Key Terms

block
Another name for a compound statement.
compound statement
A unit of code consisting of zero or more statements.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Relational Operators

KENNETH LEROY BUSBEE

Overview

A **relational operator** is a programming language construct or operator that tests or defines some kind of relation between two entities. These include numerical equality (e.g., 5 = 5) and inequalities (e.g., 4 ≥ 3).[1]

Discussion

The relational operators are often used to create a test expression that controls program flow. This type of expression is also known as a Boolean expression because they create a Boolean answer or value when evaluated. There are six common relational operators that give a Boolean value by comparing (showing the relationship) between two operands. If the operands are of different data types, implicit promotion occurs to convert the operands to the same data type.

Operator symbols and/or names can vary with different programming languages. Most programming languages use relational operators similar to the following:

Operator	Meaning
`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to
`==`	equality (equal to)
`!=` or `<>`	inequality (not equal to)

Examples:

- 9 < 25
- 9 < 3
- 9 > 14
- 9 <= 17
- 9 >= 25
- 9 == 13
- 9 != 13
- 9 !< 25

1. Wikipedia: Relational operator

- 9 <> 25

> Note: Be careful. In math you are familiar with using the symbol = to mean equal and ≠ to mean not equal. In many programming languages the ≠ is not used and the = symbol means assignment.

Key Terms

relational operator
An operator that gives a Boolean value by evaluating the relationship between two operands.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Assignment vs Equality

KENNETH LEROY BUSBEE

Overview

Assignment sets and/or re-sets the value stored in the storage location denoted by a variable name.[1] **Equality** is a relational operator that tests or defines the relationship between two entities.[2]

Discussion

Most control structures use a test expression that executes either selection (as in the: if then else) or iteration (as in the while; do while; or for loops) based on the truthfulness or falseness of the expression. Thus, we often talk about the Boolean expression that is controlling the structure. Within many programming languages, this expression must be a Boolean expression and is governed by a tight set of rules. However, in many programming languages, each data type can be used as a Boolean expression because each data type can be demoted into a Boolean value by using the rule/concept that zero and nothing represent false and all non-zero values represent true.

Within various languages, we have the potential added confusion of the equals symbol `=` as an operator that does not represent the normal math meaning of equality that we have used for most of our life. The equals symbol typically means: assignment. To get the equality concept of math we often use two equal symbols to represent the relational operator of equality. Let's consider:

```
If (pig = 'y')
    Output "Pigs are good"
Else
    Output "Pigs are bad."
```

The test expression of the control structure will always be true because the expression is an assignment (not the relational operator of `==`). It assigns the 'y' to the variable pig, then looks at the value in pig and determines that it is not zero; therefore the expression is true. And it will always be true and the else part will never be executed. This is not what the programmer had intended. The correct syntax for a Boolean expression is:

```
If (pig == 'y')
    Output "Pigs are good"
Else
    Output "Pigs are bad."
```

1. Wikipedia: Assignment (computer science)
2. Wikipedia: Relational operator

This example reminds you that you must be careful in creating your test expressions so that they are indeed a question, usually involving the relational operators. Some programming languages will generate a warning or an error when an assignment is used in a Boolean expression, and some do not.

Don't get caught using assignment for equality.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Logical Operators

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

A **logical operator** is a symbol or word used to connect two or more expressions such that the value of the compound expression produced depends only on that of the original expressions and on the meaning of the operator.[1] Common logical operators include AND, OR, and NOT.

Discussion

Within most languages, expressions that yield Boolean data type values are divided into two groups. One group uses the relational operators within their expressions and the other group uses logical operators within their expressions.

The logical operators are often used to help create a test expression that controls program flow. This type of expression is also known as a Boolean expression because they create a Boolean answer or value when evaluated. There are three common logical operators that give a Boolean value by manipulating other Boolean operand(s). Operator symbols and/or names vary with different programming languages:

Language	AND	OR	NOT
C++	`&&`	`||`	`!`
C#	`&&`	`||`	`!`
Java	`&&`	`||`	`!`
JavaScript	`&&`	`||`	`!`
Python	`and`	`or`	`not`
Swift	`&&`	`||`	`!`

The vertical dashes or piping symbol is found on the same key as the backslash \. You use the SHIFT key to get it. It is just above the Enter key on most keyboards. It may be a solid vertical line on some keyboards and show as a solid vertical line on some print fonts.

In most languages there are strict rules for forming proper logical expressions. An example is:

```
 6 > 4 && 2 <= 14
6 > 4 and 2 <= 14
```

This expression has two relational operators and one logical operator. Using the precedence of operator rules the two “relational comparison” operators will be done before the “logical and” operator. Thus:

1. Wikipedia: Logical connective

```
true && true
```
```
True and True
```

The final evaluation of the expression is: true.

We can say this in English as: It is true that six is greater than four and that two is less than or equal to fourteen.

When forming logical expressions programmers often use parentheses (even when not technically needed) to make the logic of the expression very clear. Consider the above complex Boolean expression rewritten:

```
(6 > 4) && (2 <= 14)
```
```
(6 > 4) and (2 <= 14)
```

Most programming languages recognize any non-zero value as true. This makes the following a valid expression:

```
6 > 4 && 8
```
```
6 > 4 and 8
```

But remember the order of operations. In English, this is six is greater than four and eight is not zero. Thus,

```
true && true
```
```
True and True
```

To compare 6 to both 4 and 8 would instead be written as:

```
6 > 4 && 6 > 8
```
```
6 > 4 and 6 > 8
```

This would evaluate to false as:

```
true && false
```
```
True and False
```

Truth Tables

A common way to show logical relationships is in truth tables.

Logical and (&&)

x	y	x and y
false	false	false
false	true	false
true	false	false
true	true	true

Logical or (||)

x	y	x or y
false	false	false
false	true	true
true	false	true
true	true	true

Logical not (!)	
x	**not x**
false	true
true	false

Examples

I call this example of why I hate “and” and love “or”.

Every day as I came home from school on Monday through Thursday; I would ask my mother, “May I go outside and play?” She would answer, “If your room is clean and your homework is done then you may go outside and play.” I learned to hate the word “and”. I could manage to get one of the tasks done and have some time to play before dinner, but both of them... well, I hated “and”.

On Friday my mother took a more relaxed viewpoint and when asked if I could go outside and play she responded, “If your room is clean or your homework is done then you may go outside and play.” I learned to clean my room quickly on Friday afternoon. Well, needless to say, I loved “or”.

For the next example, just imagine a teenager talking to their mother. During the conversation, mom says, “After all, your Dad is reasonable!” The teenager says, “Reasonable. (short pause) Not.”

Maybe college professors will think that all their students studied for the exam. Ha ha! Not. Well, I hope you get the point.

Examples:

- 25 < 7 || 15 > 36
- 15 > 36 || 3 < 7
- 14 > 7 && 5 <= 5
- 4 > 3 && 17 <= 7
- ! false
- ! (13 != 7)
- 9 != 7 && ! 0
- 5 > 1 && 7

More examples:

- 25 < 7 or 15 > 36
- 15 > 36 or 3 < 7
- 14 > 7 and 5 <= 5
- 4 > 3 and 17 <= 7
- not False
- not (13 != 7)
- 9 != 7 and not 0
- 5 > 1 and 7

Key Terms

logical operator

An operator used to create complex Boolean expressions.

truth tables
: A common way to show logical relationships.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Nested If Then Else

KENNETH LEROY BUSBEE

Overview

Two-way selection structures may be **nested** inside other two-way selection structures, resulting in **multi-way selection**.

Discussion

We are going to first introduce the concept of nested control structures. Nesting is a concept that places one item inside of another. Consider:

```
if expression
    true action
else
    false action
```

This is the basic form of the if then else control structure. Now consider:

```
if age is less than 18
    you can't vote
    if age is less than 16
        you can't drive
    else
        you can drive
else
    you can vote
    if age is less than 21
        you can't drink
    else
        you can drink
```

As you can see we simply included as part of the “true action” a statement and another if then else control structure. We did the same (nested another if then else) for the “false action”. In our example, we nested if then else control structures. Nesting could have an if then else within a while loop. Thus, the concept of nesting allows the mixing of the different categories of control structures.

Multiway Selection

One of the drawbacks of two-way selection is that we can only consider two choices. But what do you do if you have more than two choices? Consider the following which has four choices:

```
if age equal to 18
    you can now vote
else
    if age equal to 39
        you are middle-aged
    else
        if age equal to 65
            you can consider retirement
        else
            your age is unimportant
```

You get an appropriate message depending on the value of age. The last item is referred to as the default. If the age is not equal to 18, 39 or 65 you get the default message. To simplify the code structure, this is most often written as:

```
if age equal to 18
    you can now vote
else if age equal to 39
    you are middle-aged
else if age equal to 65
    you can consider retirement
else
    your age is unimportant
```

Key Terms

multiway selection
Using control structures to be able to select from more than two choices.
nested control structures
Placing one control structure inside of another.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Case Control Structure

KENNETH LEROY BUSBEE

Overview

A **case** or switch statement is a type of selection control mechanism used to allow the value of a variable or expression to change the control flow of program execution via a multiway branch.[1]

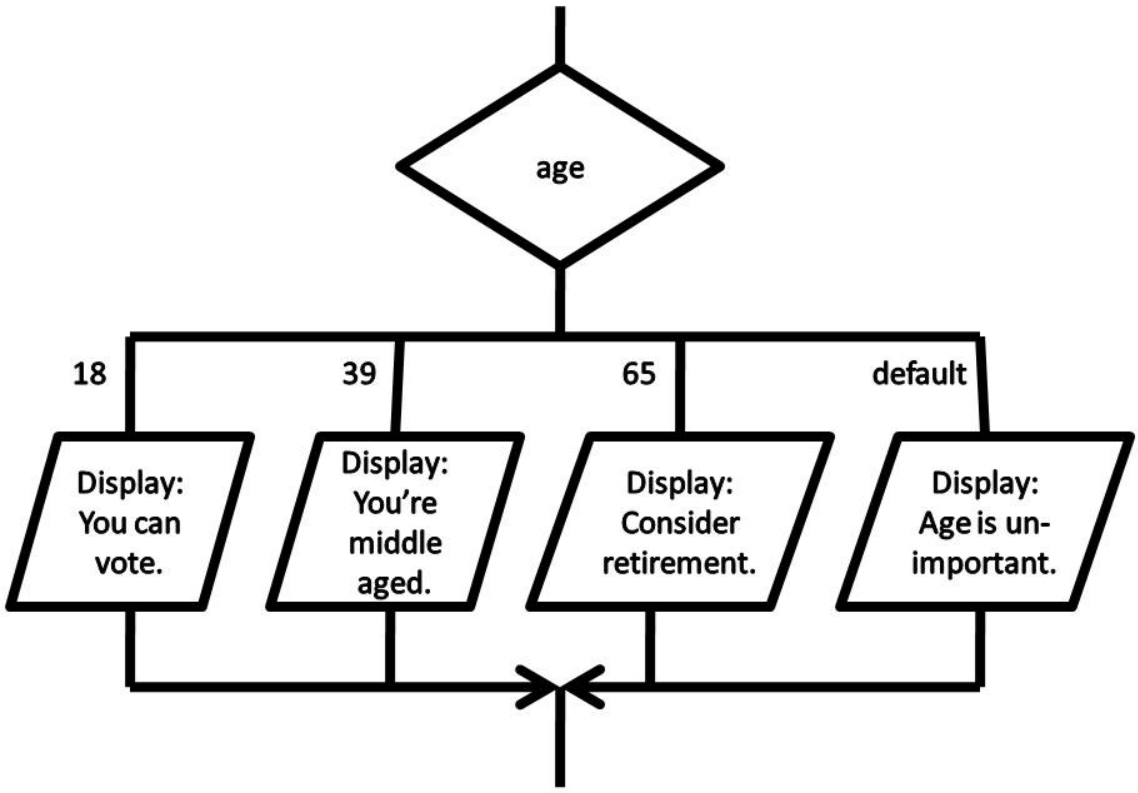
Discussion

One of the drawbacks of two-way selection is that we can only consider two choices. But what do you do if you have more than two choices? Consider the following which has four choices:

```
if age equal to 18
    you can vote
else if age equal to 39
    you're middle-aged
else if age equal to 65
    consider retirement
else
    age is unimportant
```

You get an appropriate message depending on the value of age. The last item is referred to as the default. If the age is not equal to 18, 39 or 65 you get the default message. In some situations there is no default action. Consider this flowchart example:

1. Wikipedia: Switch statement



This flowchart is of the case control structure and is used for multiway selection. The decision box holds the variable age. The logic of the case is one of equality wherein the value in the variable age is compared to the listed values in order from left to right. Thus, the value stored in age is compared to 18 or is "age equal to 18". If it is true, the logic flows down through the action and drops out at the bottom of the case structure. If the value of the test expression is false, it moves to the next listed value to the right and makes another comparison. It works exactly the same as our nested if then else structure.

Code to Accomplish Multiway Selection

Python does not support a case control structure. But using the same example as above, here is C++ / C# / Java / JavaScript / Swift code to accomplish the case control structure.

```
switch (age)
{
    case 18:
        message = "You can vote.";
        break;
    case 39:
        message = "You're middle-aged.";
```

```
        break;
    case 65:
        message = "Consider retirement.";
        break;
    default:
        message = "Age is unimportant.";
        break;
}
```

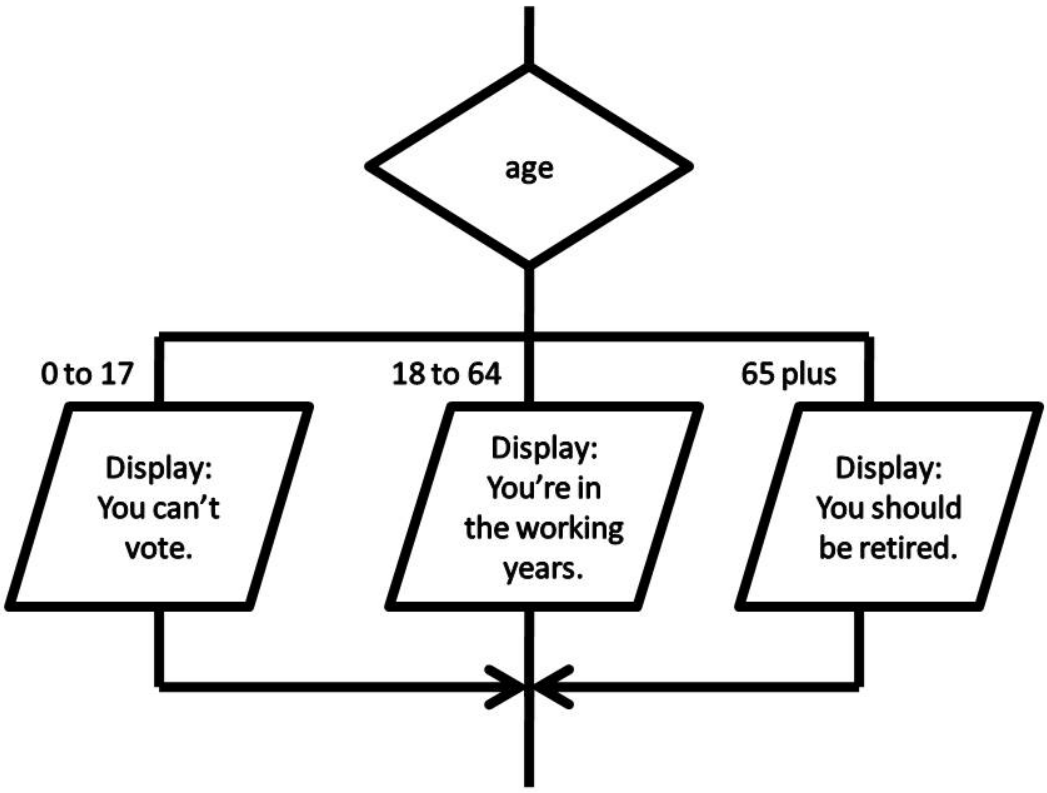
The value in the variable age is compared to the first “case”, which is the value 18 (also called the listed value) using an equality comparison or is “age equal to 18”. If it is true, the message is assigned the value “You can vote.” and the next line of code (the break) is done (which jumps us to the end of the control structure). If it is false, it moves on to the next case for comparison.

Many programming languages require the listed values for the case control structure be of the integer family of data types. This basically means either an integer or character data type. Consider this example that uses character data type (choice is a character variable):

```
switch (choice)
{
    case 'A':
        message = "You are an A student.";
        break;
    case 'B':
        message = "You are a B student.";
        break;
    case 'C':
        message = "You are a C student.";
        break;
    default:
        message = "Maybe you should study harder.";
        break;
}
```

Limitations of the Case Control Structure

Most programming languages do not allow ranges of values for case-like structures. Consider this flowcharting example that used ranges:



Consider also the following pseudocode for the same logic:

```
Case of age
  0 to 17    Display "You can't vote."
  18 to 64   Display "You're in your working years."
  65 +       Display "You should be retired."
End
```

Using the case control structure when using non-integer family or ranges of values is allowed when designing a program and documenting that design with pseudocode or flowcharting. However, the implementation in most languages would follow a nested if then else approach with complex Boolean expressions. The logic of the above examples would look like this:

```
if age > 0 and age <= to 17
    display You can't vote.
else if age is >= 18 and age <= 64
    display You're in your working years.
else
    display You should be retired.
```

Good Structured Programming Methods

Most textbook authors confirm that good structured programming techniques and habits are more important than concentrating on the technical possibilities and capabilities of the language that you are using to learn programming skills. Remember, this module is concentrating on programming fundamentals and concepts to build our initial programming skills. It is not a created with the intent to cover programming languages in detail, despite the fact that at times we have to cover language mechanics.

Key Terms

case
A control structure that does multiway selection.
switch
A control structure that can be made to act like a case control structure.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Condition Examples

DAVE BRAUNSCHWEIG

Temperature

Pseudocode

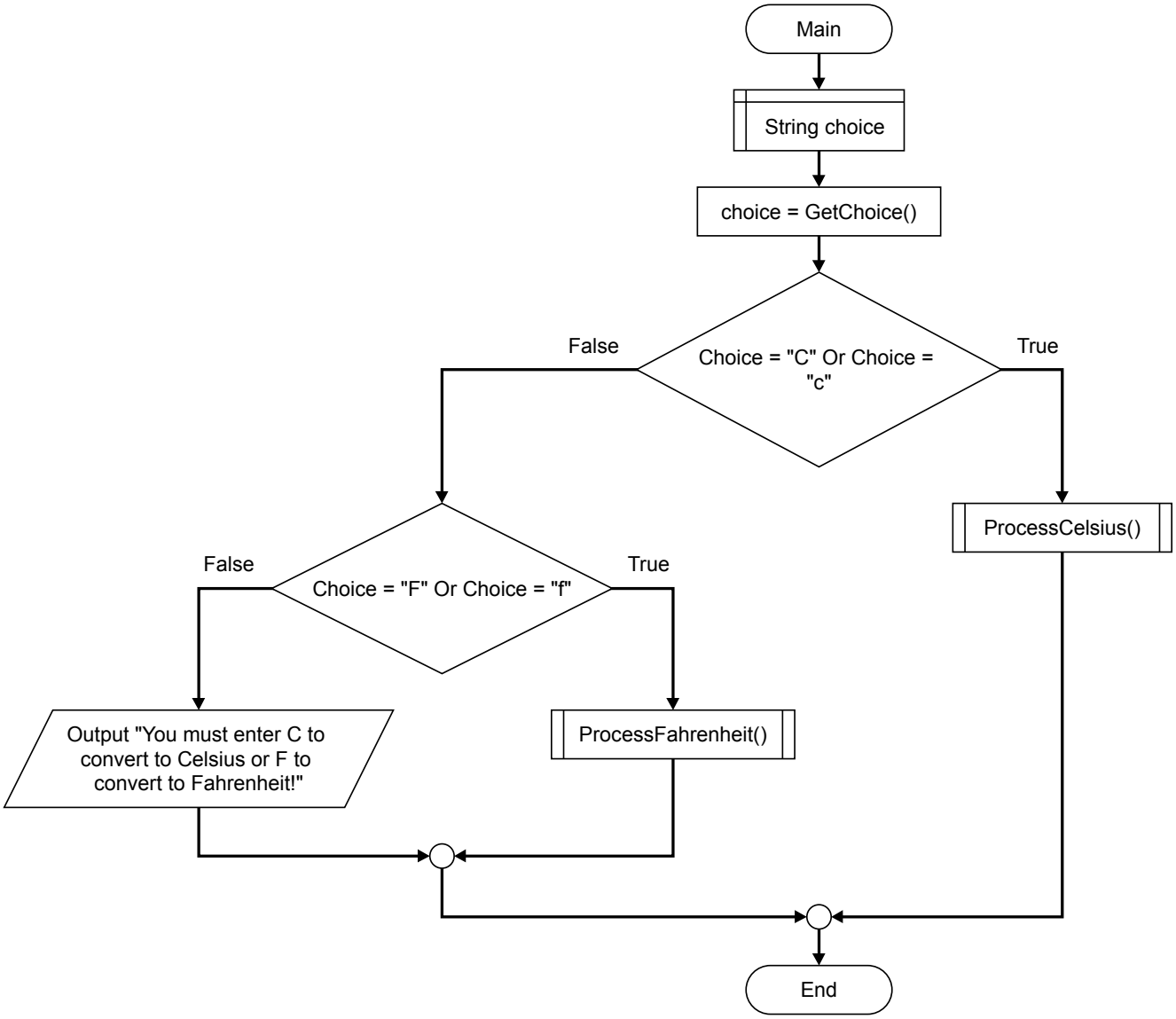
```
Function Main
    Declare String choice
    Declare Real temperature
    Declare Real result

    Assign choice = GetChoice()
    If Choice = "C" Or Choice = "c"
        Assign temperature = GetTemperature("Celsius")
        Assign result = CalculateCelsius(temperature)
        Call DisplayResult(temperature, "Fahrenheit", result, "Celsius")
    False:
        If Choice = "F" Or Choice = "f"
            Assign temperature = GetTemperature("Fahrenheit")
            Assign result = CalculateFahrenheit(temperature)
            Call DisplayResult(temperature, "Celsius", result, "Fahrenheit"
        False:
            Output "You must enter C to convert to Celsius or F to convert
        End
    End
End

Function CalculateCelsius (Real fahrenheit)
    Declare Real celsius

    Assign celsius = (fahrenheit - 32) * 5 / 9
Return Real celsius

Function CalculateFahrenheit (Real celsius)
    Declare Real fahrenheit

    Assign fahrenheit = celsius * 9 / 5 + 32
Return Real fahrenheit
```

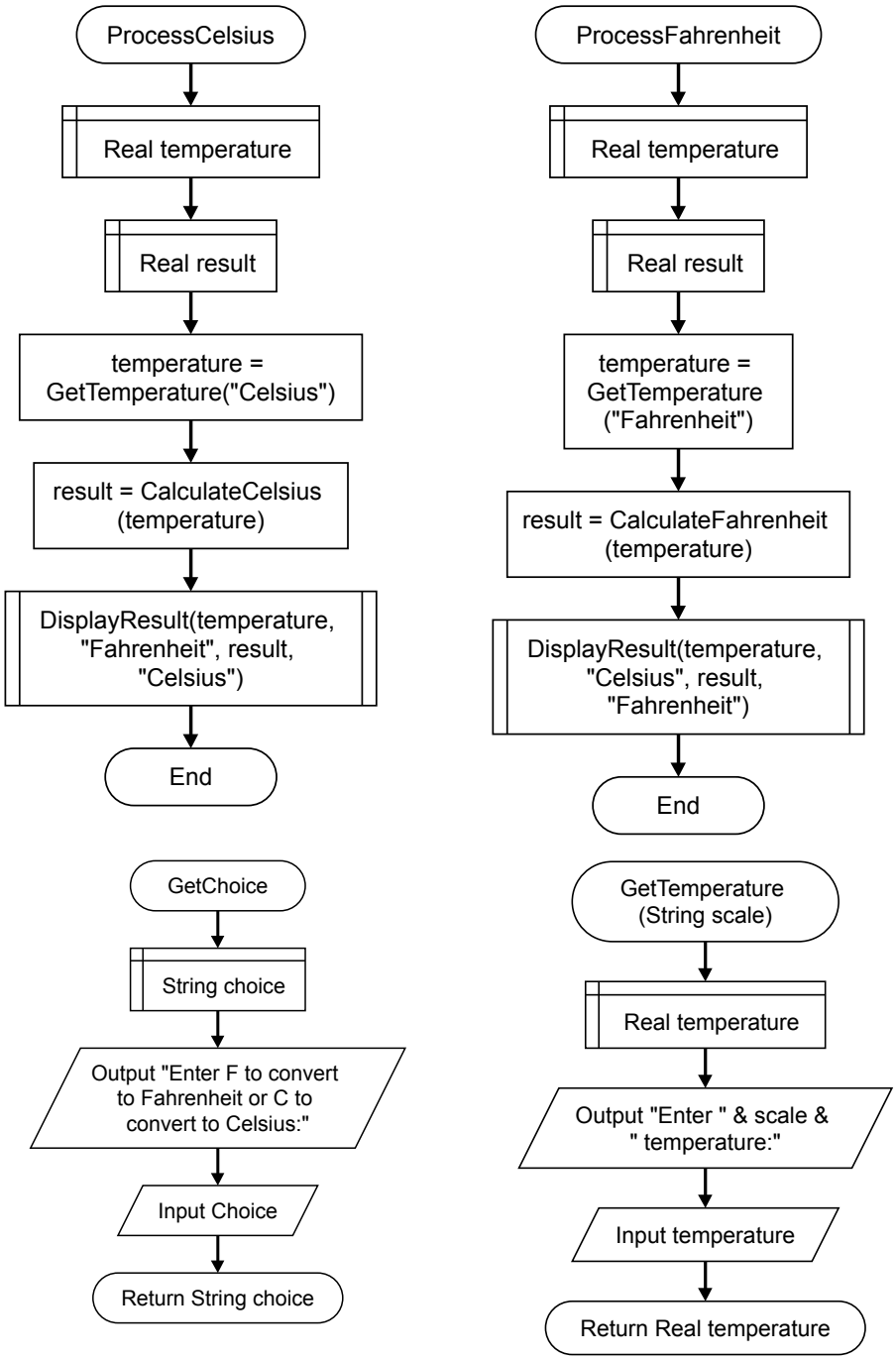
```
Function DisplayResult (Real temperature, String fromLabel, Real result, St
    Output temperature & "° " & fromLabel & " is " & result & "° " & toLabe
End

Function GetChoice
    Declare String choice

    Output "Enter F to convert to Fahrenheit or C to convert to Celsius:"
    Input Choice
Return String choice

Function GetTemperature (String label)
    Declare Real temperature

    Output "Enter " & label & " temperature:"
    Input temperature
Return Real temperature
```

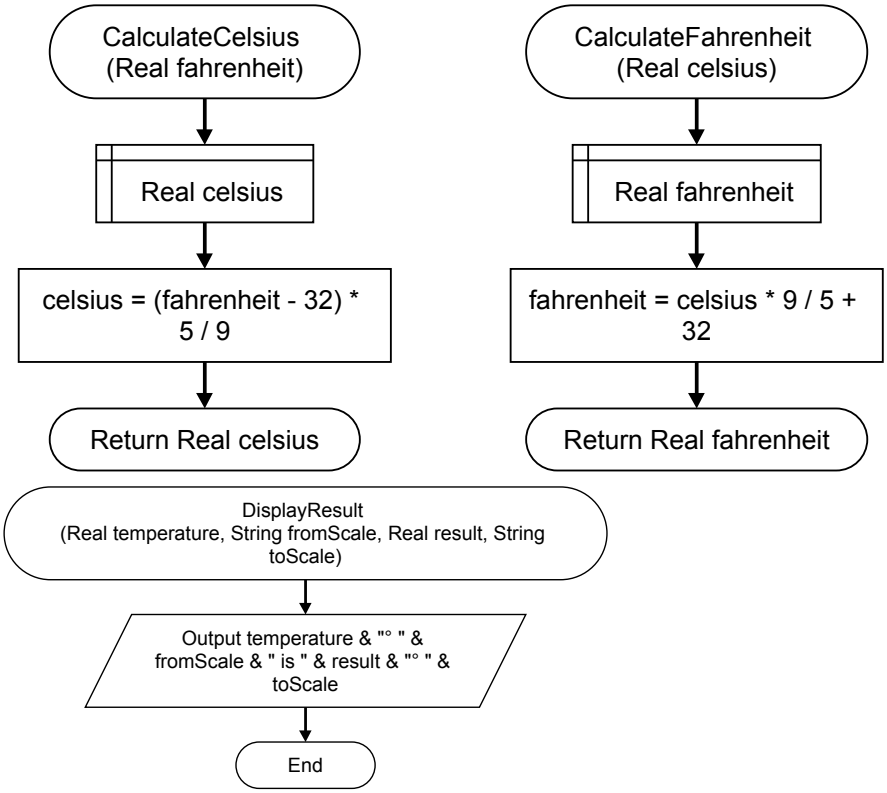
Output

```
Enter C to convert to Celsius or F to convert to Fahrenheit:
 c
Enter Fahrenheit temperature:
 100
100° Fahrenheit is 37.7777777777778° Celsius

Enter C to convert to Celsius or F to convert to Fahrenheit:
 f
Enter Celsius temperature:
 100
100° Celsius is 212° Fahrenheit

Enter C to convert to Celsius or F to convert to Fahrenheit:
 x
You must enter C to convert to Celsius or F to convert to Fahrenheit.
```

Flowchart


Main
String choice
choice = GetChoice()
Choice = "C" Or Choice = "c"
False
True
ProcessCelsius()
Choice = "F" Or Choice = "f"
False
True
Output "You must enter C to convert to Celsius or F to convert to Fahrenheit!"
ProcessFahrenheit()
End


ProcessCelsius
Real temperature
Real result
temperature = GetTemperature("Celsius")
result = CalculateCelsius (temperature)
DisplayResult(temperature, "Fahrenheit", result, "Celsius")
End
ProcessFahrenheit
Real temperature
Real result
temperature = GetTemperature ("Fahrenheit")
result = CalculateFahrenheit (temperature)
DisplayResult(temperature, "Celsius", result, "Fahrenheit")
End
GetChoice
String choice
Output "Enter F to convert to Fahrenheit or C to convert to Celsius:"
Input Choice
Return String choice
GetTemperature (String scale)
Real temperature
Output "Enter " & scale & " temperature:"
Input temperature
Return Real temperature



References

- Wikiversity: Computer Programming

C++ Examples

DAVE BRAUNSCHWEIG

Temperature

```
// This program asks the user to select Fahrenheit or Celsius conversion
// and input a given temperature. Then the program converts the given
// temperature and displays the result.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/C%2B%2B_Programming

#include <iostream>

using namespace std;

double getTemperature(string label);
double calculateCelsius(double fahrenheit);
double calculateFahrenheit(double celsius);
void displayResult(double temperature, string fromLabel, double result, str

int main() {
    // main could either be an if-else structure or a switch-case structure

    char choice;
    double temperature;
    double result;

    cout << "Enter F to convert to Fahrenheit or C to convert to Celsius:" <

    // if-else approach
    if (choice == 'C' || choice == 'c') {
        temperature = getTemperature("Fahrenheit");
        result = calculateCelsius(temperature);
        displayResult(temperature, "Fahrenheit", result, "Celsius");
    }
    else if (choice == 'F' || choice == 'f') {
        temperature = getTemperature("Celsius");
        result = calculateFahrenheit(temperature);
```

```
        displayResult(temperature, "Celsius", result, "Fahrenheit");
    }
    else {
        cout << "You must enter C to convert to Celsius or F to convert to
    }

    // switch-case approach
    switch(choice) {
        case 'C':
        case 'c':
            temperature = getTemperature("Fahrenheit");
            result = calculateCelsius(temperature);
            displayResult(temperature, "Fahrenheit", result, "Celsius");
            break;
        case 'F':
        case 'f':
            temperature = getTemperature("Celsius");
            result = calculateFahrenheit(temperature);
            displayResult(temperature, "Celsius", result, "Fahrenheit");
            break;
        default:
            cout << "You must enter C to convert to Celsius or F to convert
    }
}

double getTemperature(string label) {
    double temperature;

    cout << "Enter " << label << " temperature:" <> temperature;

    return temperature;
}

double calculateCelsius(double fahrenheit) {
    double celsius;

    celsius = (fahrenheit - 32) * 5 / 9;

    return celsius;
}

double calculateFahrenheit(double celsius) {
    double fahrenheit;
```

```
    fahrenheit = celsius * 9 / 5 + 32;

    return fahrenheit;
}

void displayResult(double temperature, string fromLabel, double result, str
    cout << temperature << "° " << fromLabel << " is " << result << "° " <<
}
```

Output

```
Enter C to convert to Celsius or F to convert to Fahrenheit:
 c
Enter Fahrenheit temperature:
 100
100° Fahrenheit is 37.7778° Celsius

Enter C to convert to Celsius or F to convert to Fahrenheit:
 f
Enter Celsius temperature:
 100
100° Celsius is 212° Fahrenheit

Enter C to convert to Celsius or F to convert to Fahrenheit:
 x
You must enter C to convert to Celsius or F to convert to Fahrenheit.
```

References

- Wikiversity: Computer Programming

C# Examples

DAVE BRAUNSCHWEIG

Temperature

```
// This program asks the user to select Fahrenheit or Celsius conversion
// and input a given temperature. Then the program converts the given
// temperature and displays the result.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/C_Sharp_Programming

using System;

public class MainClass
{
    public static void Main(String[] args)
    {
        // main could either be an if-else structure or a switch-case struc

        string choice;
        double temperature;
        double result;

        Console.WriteLine("Enter F to convert to Fahrenheit or C to convert
        choice = Console.ReadLine();

        // if-else approach
        if (choice == "C" || choice == "c")
        {
            temperature = GetTemperature("Fahrenheit");
            result = CalculateCelsius(temperature);
            DisplayResult(temperature, "Fahrenheit", result, "Celsius");
        }
        else if (choice == "F" || choice == "f")
        {
            temperature = GetTemperature("Celsius");
            result = CalculateFahrenheit(temperature);
            DisplayResult(temperature, "Celsius", result, "Fahrenheit");
```

```
        }
        else
        {
            Console.WriteLine("You must enter C to convert to Celsius or F
        }

        // switch-case approach
        switch(choice)
        {
            case "C":
            case "c":
                temperature = GetTemperature("Fahrenheit");
                result = CalculateCelsius(temperature);
                DisplayResult(temperature, "Fahrenheit", result, "Celsius")
                break;
            case "F":
            case "f":
                temperature = GetTemperature("Celsius");
                result = CalculateFahrenheit(temperature);
                DisplayResult(temperature, "Celsius", result, "Fahrenheit")
                break;
            default:
                Console.WriteLine("You must enter C to convert to Celsius o
                break;
        }
    }

    private static double GetTemperature(string label)
    {
        string input;
        double temperature;

        Console.WriteLine("Enter " + label + " temperature:");
        input = Console.ReadLine();
        temperature = Convert.ToDouble(input);

        return temperature;
    }

    private static double CalculateCelsius(double fahrenheit)
    {
        double celsius;
```

```
        celsius = (fahrenheit - 32) * 5 / 9;

        return celsius;
    }

    private static double CalculateFahrenheit(double celsius)
    {
        double fahrenheit;

        fahrenheit = celsius * 9 / 5 + 32;

        return fahrenheit;
    }

    private static void DisplayResult(double fahrenheit, string fromLabel, 
    {
        Console.WriteLine(fahrenheit.ToString() + "° " + fromLabel + " is "
    }
}
```

Output

```
Enter C to convert to Celsius or F to convert to Fahrenheit:
 c
Enter Fahrenheit temperature:
 100
100° Fahrenheit is 37.7777777777778° Celsius

Enter C to convert to Celsius or F to convert to Fahrenheit:
 f
Enter Celsius temperature:
 100
100° Celsius is 212° Fahrenheit

Enter C to convert to Celsius or F to convert to Fahrenheit:
 x
You must enter C to convert to Celsius or F to convert to Fahrenheit.
```

References

- Wikiversity: Computer Programming

Java Examples

DAVE BRAUNSCHWEIG

Temperature

```
// This program asks the user to select Fahrenheit or Celsius conversion
// and input a given temperature. Then the program converts the given
// temperature and displays the result.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/Java_Programming

import java.util.*;

class Main {
    private static Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        // main could either be an if-else structure or a switch-case struc

        String choice;
        double temperature;
        double result;

        choice = getChoice();

        // if-else approach
        if (choice.equals("C") || choice.equals("c")) {
            temperature = getTemperature("Fahrenheit");
            result = calculateCelsius(temperature);
            displayResult(temperature, "Fahrenheit", result, "Celsius");
        } else if (choice.equals("F") || choice.equals("f")) {
            temperature = getTemperature("Celsius");
            result = calculateFahrenheit(temperature);
            displayResult(temperature, "Celsius", result, "Fahrenheit");
        } else {
            System.out.println("You must enter C to convert to Celsius or F
        }
```

```
        // switch-case approach
        switch (choice) {
            case "C":
            case "c":
                temperature = getTemperature("Fahrenheit");
                result = calculateCelsius(temperature);
                displayResult(temperature, "Fahrenheit", result, "Celsius")
                break;
            case "F":
            case "f":
                temperature = getTemperature("Celsius");
                result = calculateFahrenheit(temperature);
                displayResult(temperature, "Celsius", result, "Fahrenheit")
                break;
            default:
                System.out.println("You must enter C to convert to Celsius
        }
    }

    public static String getChoice() {
        String choice;

        System.out.println("Enter C to convert to Celsius or F to convert t
        choice = input.nextLine();

        return choice;
    }

    public static double getTemperature(String label) {
        double temperature;

        System.out.println("Enter " + label + " temperature:");
        temperature = input.nextDouble();

        return temperature;
    }

    public static double calculateCelsius(double fahrenheit) {
        double celsius;

        celsius = (fahrenheit - 32) * 5 / 9;

        return celsius;
```

```
    }

    public static double calculateFahrenheit(double celsius) {
        double fahrenheit;

        fahrenheit = celsius * 9 / 5 + 32;

        return fahrenheit;
    }

    public static void displayResult(double temperature, String fromLabel, 
        System.out.println(Double.toString(temperature) + "° " + fromLabel 
    }
}
```

Output

```
Enter C to convert to Celsius or F to convert to Fahrenheit:
 c
Enter Fahrenheit temperature:
 100
100.0° Fahrenheit is 37.77777777777778° Celsius

Enter C to convert to Celsius or F to convert to Fahrenheit:
 f
Enter Celsius temperature:
 100
100.0° Celsius is 212.0° Fahrenheit

Enter C to convert to Celsius or F to convert to Fahrenheit:
 x
You must enter C to convert to Celsius or F to convert to Fahrenheit.
```

References

- Wikiversity: Computer Programming

JavaScript Examples

DAVE BRAUNSCHWEIG

Temperature

```
// This program asks the user to select Fahrenheit or Celsius conversion
// and input a given temperature. Then the program converts the given
// temperature and displays the result.
//
// References:
//   https://www.mathsisfun.com/temperature-conversion.html
//   https://en.wikibooks.org/wiki/JavaScript

main();

function main()
{
    // main could either be an if-else structure or a switch-case structure

    var choice;
    var temperature;
    var result;

    choice = getChoice();

    // if-else approach
    if (choice == "C" || choice == "c") {
        temperature = getTemperature("Fahrenheit");
        result = calculateCelsius(temperature);
        displayResult(temperature, "Fahrenheit", result, "Celsius");
    }
    else if (choice == "F" || choice == "f") {
        temperature = getTemperature("Celsius");
        result = calculateFahrenheit(temperature);
        displayResult(temperature, "Celsius", result, "Fahrenheit");
    }
    else {
        output("You must enter C to convert to Celsius or F to convert to F
    }
```

```
    // switch-case approach
    switch(choice) {
        case 'C':
        case 'c':
            temperature = getTemperature("Fahrenheit");
            result = calculateCelsius(temperature);
            displayResult(temperature, "Fahrenheit", result, "Celsius");
            break;
        case 'F':
        case 'f':
            temperature = getTemperature("Celsius");
            result = calculateFahrenheit(temperature);
            displayResult(temperature, "Celsius", result, "Fahrenheit");
            break;
        default:
            output("You must enter C to convert to Celsius or F to convert
    }
}

function getChoice() {
    var choice;

    output("Enter C to convert to Celsius or F to convert to Fahrenheit:");
    choice = input();

    return choice;
}

function getTemperature(label) {
    var temperature;

    output("Enter " + label + " temperature:");
    temperature = input();

    return temperature;
}

function calculateCelsius(fahrenheit) {
    var celsius;

    celsius = (fahrenheit - 32) * 5 / 9;

    return celsius;
```

```
}

function calculateFahrenheit(celsius) {
    var fahrenheit;

    fahrenheit = celsius * 9 / 5 + 32;

    return fahrenheit;
}

function displayResult(temperature, fromLabel, result, toLabel) {
    output(temperature.toString() + "° " + fromLabel + " is " + result + "°
}

function input(text) {
  if (typeof window === 'object') {
    return prompt(text)
  }
  else if (typeof console === 'object') {
    const rls = require('readline-sync');
    var value = rls.question(text);
    return value;
  }
  else {
    output(text);
    var isr = new java.io.InputStreamReader(java.lang.System.in);
    var br = new java.io.BufferedReader(isr);
    var line = br.readLine();
    return line.trim();
  }
}

function output(text) {
  if (typeof document === 'object') {
    document.write(text);
  }
  else if (typeof console === 'object') {
    console.log(text);
  }
  else {
    print(text);
  }
}
```

Output

```
Enter C to convert to Celsius or F to convert to Fahrenheit:
 c
Enter Fahrenheit temperature:
 100
100° Fahrenheit is 37.77777777777778° Celsius

Enter C to convert to Celsius or F to convert to Fahrenheit:
 f
Enter Celsius temperature:
 100
100° Celsius is 212° Fahrenheit

Enter C to convert to Celsius or F to convert to Fahrenheit:
 x
You must enter C to convert to Celsius or F to convert to Fahrenheit.
```

References

- Wikiversity: Computer Programming

Python Examples

DAVE BRAUNSCHWEIG

Temperature

```
# This program asks the user to select Fahrenheit or Celsius conversion
# and input a given temperature. Then the program converts the given
# temperature and displays the result.
#
# References:
#     https://www.mathsisfun.com/temperature-conversion.html
#     https://en.wikibooks.org/wiki/Python_Programming


def get_choice():
    print("Enter C to convert to Celsius or F to convert to Fahrenheit:")
    choice = input()
    return choice


def get_temperature(label):
    print(f"Enter {label} temperature:")
    temperature = float(input())
    return temperature


def calculate_celsius(fahrenheit):
    celsius = (fahrenheit - 32) * 5 / 9
    return celsius


def calculate_fahrenheit(celsius):
    fahrenheit = celsius * 9 / 5 + 32
    return fahrenheit


def display_result(temperature, from_label, result, to_label):
    print(f"{temperature}° {from_label} is {result}° {to_label}")
```

```
def main():
    choice = get_choice()
    if choice == "C" or choice == "c":
        temperature = get_temperature("Fahrenheit")
        result = calculate_celsius(temperature)
        display_result (temperature, "Fahrenheit", result, "Celsius")
    elif choice == "F" or choice == "f":
        temperature = get_temperature("Celsius")
        result = calculate_fahrenheit(temperature)
        display_result (temperature, "Celsius", result, "Fahrenheit")
    else:
        print("You must enter C to convert to Celsius or F to convert to Fa


main()
```

Output

```
Enter C to convert to Celsius or F to convert to Fahrenheit:
 c
Enter Fahrenheit temperature:
 100
100.0° Fahrenheit is 37.77777777777778° Celsius

Enter C to convert to Celsius or F to convert to Fahrenheit:
 f
Enter Celsius temperature:
 100
100.0° Celsius is 212.0° Fahrenheit

Enter C to convert to Celsius or F to convert to Fahrenheit:
 x
You must enter C to convert to Celsius or F to convert to Fahrenheit.
```

References

- Wikiversity: Computer Programming

Swift Examples

DAVE BRAUNSCHWEIG

Temperature

```
// This program asks the user for a Fahrenheit temperature,
// converts the given temperature to Celsius,
// and displays the results.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://developer.apple.com/library/content/documentation/Swift/Conc

func getChoice() -> String {
    var choice: String

    print("Enter C to convert to Celsius or F to convert to Fahrenheit:")
    choice = readLine(strippingNewline: true)!

    return choice
}

func getTemperature(label: String) -> Double {
    var temperature: Double

    print("Enter " + label + " temperature:")
    temperature = Double(readLine(strippingNewline: true)!)!

    return temperature
}

func calculateCelsius(fahrenheit: Double) -> Double {
    var celsius: Double

    celsius = (fahrenheit - 32) * 5 / 9

    return celsius
}

func calculateFahrenheit(celsius: Double) -> Double {
```

```
    var fahrenheit: Double

    fahrenheit = celsius * 9 / 5 + 32

    return fahrenheit
}

func displayResult(temperature: Double, fromLabel: String, result: Double,
    print(String(temperature) + "° " + fromLabel + " is " + String(result)
}

func main() {
    // main could either be an if-else structure or a switch-case structure

    var choice: String
    var temperature: Double
    var result: Double

    choice = getChoice()

    // if-else approach
    if choice == "C" || choice == "c" {
        temperature = getTemperature(label:"Fahrenheit")
        result = calculateCelsius(fahrenheit:temperature)
        displayResult(temperature:temperature, fromLabel:"Fahrenheit", resu
    else if choice == "F" || choice == "f" {
        temperature = getTemperature(label:"Celsius")
        result = calculateFahrenheit(celsius:temperature)
        displayResult(temperature:temperature, fromLabel:"Celsius", result:
    }
    else {
        print("You must enter C to convert to Celsius or F to convert to Fa
    }

    // switch-case approach
    switch choice {
        case "C", "c":
            temperature = getTemperature(label:"Fahrenheit")
            result = calculateCelsius(fahrenheit:temperature)
            displayResult(temperature:temperature, fromLabel:"Fahrenheit",
        case "F", "f":
            temperature = getTemperature(label:"Celsius")
            result = calculateFahrenheit(celsius:temperature)
```

```
            displayResult(temperature:temperature, fromLabel:"Celsius", res
        default:
            print("You must enter C to convert to Celsius or F to convert t
    }
}

main()
```

Output

```
Enter C to convert to Celsius or F to convert to Fahrenheit:
 c
Enter Fahrenheit temperature:
 100
100.0° Fahrenheit is 37.77777777777778° Celsius

Enter C to convert to Celsius or F to convert to Fahrenheit:
 f
Enter Celsius temperature:
 100
100.0° Celsius is 212.0° Fahrenheit

Enter C to convert to Celsius or F to convert to Fahrenheit:
 x
You must enter C to convert to Celsius or F to convert to Fahrenheit.
```

References

- Wikiversity: Computer Programming

Practice: Conditions

KENNETH LEROY BUSBEE

Review Questions

True / False

1. There are only two categories of control structures.
2. Branching control structures are rarely used in good structured programming.
3. If then else is a multiway selection control structure.
4. The while control structure is part of the branching category.
5. Pseudocode is better than flowcharting.

Answers:

1. false
2. true
3. false
4. false
5. false

Expressions

Evaluate the following Boolean expressions:

1. 25 < 7
2. 3 < 7
3. 14 > 7
4. 17 <= 7
5. 25 >= 7
6. 13 == 7
7. 9 != 7
8. 5 !> 7
9. 25 > 39 || 15 > 36
10. 19 > 26 || 13 < 17
11. 14 < 7 && 6 <= 6
12. 4 > 3 && 17 >= 7
13. ! true
14. ! (13 == 7)
15. 9 != 7 && ! 1
16. 6 < && 8

Answers:

1. 0
2. 1

3. 1
4. 0
5. 1
6. 0
7. 1
8. Error, the “not greater than” is not a valid operator.
9. 0
10. 1
11. 0
12. 1
13. 0
14. 1
15. 0
16. Error, there needs to be an operand between the operators < and &&.

Short Answer

1. List the four categories of control structures and provide a brief description of each category.
2. Create a table with the six relational operators and their meanings.

Activities

Complete the following activities using pseudocode, a flowcharting tool, or your selected programming language. Use separate functions for input, each type of processing, and output. Avoid global variables by passing parameters and returning results. Create test data to validate the accuracy of each program. Add comments at the top of the program and include references to any resources used.

1. Create a program to prompt the user for hours and rate per hour and then compute gross pay (hours * rate). Include a calculation to give 1.5 times the hourly rate for any overtime (hours worked above 40 hours).[1] For example, 50 hours worked at $10 per hour with overtime is $550.
2. Create a program that asks the user how old they are in years. Then ask the user if they would like to know how old they are in (M)onths, (D)ays, (H)ours, or (S)econds. Use if/else conditional statements to calculate and display their approximate age in the selected timeframe. Do not perform any unnecessary calculations.
3. Review MathsIsFun: US Standard Lengths. Create a program that asks the user for a distance in miles, and then ask the user if they want the distance in US measurements (yards, feet, and inches) or in metric measurements (kilometers, meters, and centimeters). Use if/else conditional statements to determine their selection and then calculate and display the results.
4. Review MathsIsFun: Area of Plane Shapes. Create a program that asks the user what shape they would like to calculate the area for. Use if/else conditional statements to determine their selection and then gather the appropriate input and calculate and display the area of the shape.
5. Review Wikipedia: Aging in dogs. Create a program to prompt the user for the name of their dog and its age in human years. Calculate and display the age of their dog in dog years, assuming the first two years equal 10.5 years each, with subsequent years equaling four

1. PythonLearn: Variables, expressions, and statements

human years. Be sure to include the dog's name in the output, such as:

```
Spike is 14.0 years old in dog years.
```

6. Create a program that helps the user determine what sock size to order based on their shoe size:

```
< 4 = extra small
4 to 6 = small
7 to 9 = medium
10 to 12 = large
13+ = extra large
```

   Use if/else conditional statements to determine their selection and then display the results. Round half-sizes up to the next whole size. One option for rounding is to add 0.5 and then convert to an integer.
7. If your programming language supports it, update one or more of the programs above to replace the if/else conditional statements with case/select conditional statements.
8. Review Wikipedia: Is functions. If your programming language supports it, update one or more of the programs above to include input validation for all numeric input.
9. If your programming language supports it, extend one or more of the programs above by adding structured exception handling statements (try-catch, try-except, etc.) to handle any runtime errors caused by the user entering invalid values for the input.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++
- Wikiversity: Computer Programming

PART V

LOOPS

Overview

This chapter introduces loops and iteration control structures.

Chapter Outline

Learning Objectives

1. Understand key terms and definitions.
2. Identify control structures based on test before iteration, test after iteration, and counting, and when to use each type.
3. Given example pseudocode, flowcharts, and source code, create a program that uses loops and iteration control structures to solve a given problem.

Iteration Control Structures

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

In **iteration control structures**, a statement or block is executed until the program reaches a certain state, or operations have been applied to every element of a collection. This is usually expressed with keywords such as `while`, `repeat`, `for`, or `do..until`.[1]

Discussion

The basic attribute of an iteration control structure is to be able to repeat some lines of code. The visual display of iteration creates a circular loop pattern when flowcharted, thus the word “loop” is associated with iteration control structures. Iteration can be accomplished with test before loops, test after loops, and counting loops. A question using Boolean concepts usually controls how often the loop will execute.

Iteration (Repetition) Control Structures

pseudocode: While

```
count assigned zero
While count < 5
    Display "I love computers!"
    Increment count
End
```

pseudocode: Do While

```
count assigned five
Do
    Display "Blast off is soon!"
    Decrement count
While count > zero
```

pseudocode: Repeat Until

1. Wikipedia: Structured programming

```
count assigned five
Repeat
    Display "Blast off is soon!"
    Decrement count
Until count < one
```

pseudocode: For

```
For x starts at 0, x < 5, increment x
    Display "Are we having fun?"
End
```

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

While Loop

KENNETH LEROY BUSBEE

Overview

A **while loop** is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.[1]

Discussion

Introduction to Test Before Loops

There are two commonly used test before loops in the iteration (or repetition) category of control structures. They are: while and for. This module covers the: while.

Understanding Iteration in General – while

The concept of iteration is connected to possibly wanting to repeat an action. Like all control structures we ask a question to control the execution of the loop. The term loop comes from the circular looping motion that occurs when using flowcharting. The basic form of the while loop is as follows:

```
initialization of the flag
while the answer to the question is true then do
    some statements or action
    some statements or action
    some statements or action
    update the flag
```

In most programming languages the question (called a test expression) is a Boolean expression. The Boolean data type has two values – true and false. Let's rewrite the structure to consider this:

```
initialization of the flag
while the expression is true then do
    some statements or action
    some statements or action
```

1. Wikipedia: While loop

```
    some statements or action
    update the flag
```

Within the while control structure there are four attributes to a properly working loop. They are:

- Initializing the flag
- Test expression
- Action or actions
- Update of the flag

The initialization of the flag is not technically part of the control structure, but a necessary item to occur before the loop is started. The English phrasing is, “While the expression is true, do the following actions”. This is looping on the true. When the test expression is false, you stop the loop and go on with the next item in the program. Notice, because this is a test before loop the action **might not happen**. It is called a test before loop because the test comes before the action. It is also sometimes called a pre-test loop, meaning the test is pre (or Latin for before) the action and update.

Human Example of the while Loop

Consider the following one-way conversation from a mother to her child.

Child: The child says nothing, but mother knows the child had Cheerios for breakfast and history tells us that the child most likely spilled some Cheerios on the floor.

Mother says: “While it is true that you see (As long as you can see) a Cheerio on the floor, pick it up and put it in the garbage.”

Note: All of the elements are present to determine the action (or flow) that the child will be doing (in this case repeating). Because the question (can you see a Cheerios) has only two possible answers (true or false) the action will continue while there are Cheerios on the floor. Either the child 1) never picks up a Cheerio because they never spilled any or 2) picks up a Cheerio and keeps picking up Cheerios one at a time while he can see a Cheerio on the floor (that is until they are all picked up).

Infinite Loops

At this point, it is worth mentioning that good programming always provides for a method to ensure that the loop question will eventually be false so that the loop will stop executing and the program continues with the next line of code. However, if this does not happen, then the program is in an infinite loop. Infinite loops are a bad thing. Consider the following code:

Pseudocode infinite loop

```
loop_response = 'y'
While loop_response == 'y'
    Output "What is your age? "
    Input user_age
    Output "What is your friend's age? "
```

```
    Input friend_age
    Output "Together your ages add up to: "
    Output user_age + friend_age
```

The programmer assigned a value to the flag before the loop which is correct. However, they forgot to update the flag. Every time the test expression is asked it will always be true. Thus, an infinite loop because the programmer did not provide a way to exit the loop (he forgot to update the flag). Consider the following code:

```
loop_response = 'y';
While loop_response = 'y'
    Output "What is your age? "
    Input user_age
    Output "What is your friend's age? "
    Input friend_age
    Output "Together your ages add up to: "
    Output user_age + friend_age
    Output "Do you want to try again? y or n "
    Input loop_response
```

No matter what the user replies during the flag update, the test expression does not do a relational comparison but does an assignment. It assigns 'y' to the variable and asks if 'y' is true? Since all non-zero values are treated as representing true, the answer to the test expression is true. Viola, you have an infinite loop.

Counting Loops

The examples above are for an event controlled loop. The flag updating is an event where someone decides if they want the loop to execute again. Often the initialization sets the flag so that the loop will execute at least once.

Another common usage of the while loop is as a counting loop. Consider:

```
counter = 0
While counter < 5
    Output "I love ice cream!"
    counter += 1
```

The variable counter is said to be controlling the loop. It is set to zero (called initialization) before entering the while loop structure and as long as it is less than 5 (five); the loop action will be executed. But part of the loop action uses the increment operator to increase counter's value by one. After executing the loop five times (once for counter's values of: 0, 1, 2, 3 and 4) the expression will be false and the next line of code in the program will execute. A counting loop is designed to execute the action (which could be more than one statement) a set of given number of times. In our example, the message is displayed five times on the monitor. It is accomplished by making sure all four attributes of the while control structure are present and working properly. The attributes are:

- Initializing the flag
- Test expression
- Action or actions
- Update of the flag

Missing an attribute might cause an infinite loop or give undesired results (does not work properly).

Infinite Loops

Consider:

```
counter = 0;
while counter < 5
    Output "I love ice cream!"
```

Missing the flag update usually causes an infinite loop.

Variations on Counting

In the following example, the integer variable age is said to be controlling the loop (that is the flag). We can assume that age has a value provided earlier in the program. Because the while structure is a test before loop; it is possible that the person's age is 0 (zero) and the first time we test the expression it will be false and the action part of the loop would never be executed.

```
While 0 < age
    Output "I love candy!"
    age -= 1
```

Consider the following variation assuming that age and counter are both integer data type and that age has a value:

```
counter = 0;
While counter < age
    Output "I love corn chips!"
    counter += 1
```

This loop is a counting loop similar to our first counting loop example. The only difference is instead of using a literal constant (in other words 5) in our expression, we used the variable age (and thus the value stored in age) to determine how many times to execute the loop. However, unlike our first counting loop example which will always execute exactly 5 times; it is possible that the person's age is 0 (zero) and the first time we test the expression it will be false and the action part of the loop would never be executed.

Key Terms

counting controlled
: Using a variable to count up or down to control a loop.

event controlled
: Using user input to control a loop.

infinite loop
: A sequence of instructions which loops endlessly, either due to the loop having no terminating condition, having one that can never be met, or one that causes the loop to start over.[2]

initialize item
: An attribute of iteration control structures.

loop attributes
: Items associated with iteration or looping control structures.

might not happen
: Indicating that test before loops might not execute the action.

while
: A test before iteration control structure.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

2. Wikipedia: Infinite loop

Do While Loop

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

A **do while loop** is a control flow statement that executes a block of code at least once, and then repeatedly executes the block, or not, depending on a given boolean condition at the end of the block.[1]

Some languages may use a different naming convention for this type of loop. For example, the Pascal language has a **repeat until loop**, which continues to run until the control expression is true (and then terminates) — whereas a "while" loop runs while the control expression is true (and terminates once the expression becomes false).[2]

Discussion

Introduction to Test After Loops

There are two commonly used test after loops in the iteration (or repetition) category of control structures. They are: do while and repeat until. This module covers both.

Understanding Iteration in General – do while

The concept of iteration is connected to possibly wanting to repeat an action. Like all control structures, we ask a question to control the execution of the loop. The term loop comes from the circular looping motion that occurs when using flowcharting. The basic form of the do while loop is as follows:

```
do
    some statements or action
    some statements or action
    some statements or action
    update the flag
while the answer to the question is true
```

In most programming languages the question (called a test expression) is a Boolean expression. The Boolean data type has two values – true and false. Let's rewrite the structure to consider this:

1. Wikipedia: Do while loop
2. Wikipedia: Do while loop

```
do
    some statements or action
    some statements or action
    some statements or action
    update the flag
while expression is true
```

Within the do while control structure there are three attributes of a properly working loop. They are:

- Action or actions
- Update of the flag
- Test expression

The English phrasing is, "You do the action while the expression is true". This is looping on the true. When the test expression is false, you stop the loop and go on with the next item in the program. Notice, because this is a test after loop the action will always happen **at least once**. It is called a test after loop because the test comes after the action. It is also sometimes called a post-test loop, meaning the test is post (or Latin for after) the action and update.

Understanding Iteration in General – repeat until

The concept of iteration is connected to possibly wanting to repeat an action. Like all control structures, we ask a question to control the execution of the loop. The term loop comes from the circular looping motion that occurs when using flowcharting. The basic form of the repeat until loop is as follows:

```
repeat
    some statements or action
    some statements or action
    some statements or action
    update the flag
until the answer to the question becomes true
```

In most programming languages the question (called a test expression) is a Boolean expression. The Boolean data type has two values – true and false. Let's rewrite the structure to consider this:

```
repeat
    some statements or action
    some statements or action
    some statements or action
    update the flag
until expression becomes true
```

Within the repeat until control structure, there are three attributes of a properly working loop. They are:

- Action or actions
- Update of the flag
- Test expression

The English phrasing is, “You repeat the action until the expression becomes true”. This is looping on the false. When the test expression becomes true, you stop the loop and go on with the next item in the program. Notice, because this is a test after loop the action will always happen **at least once**. It is called a “test after loop” because the test comes after the action. It is also sometimes called a post-test loop, meaning the test is post (or Latin for after) the action and update.

An Example

```
Do
    Output "What is your age? "
    Input user_age
    Output "What is your friend's age? "
    Input friend_age
    Output "Together your ages add up to: "
    Output age_user + friend_age
    Output "Do you want to try it again? y or n "
    Input loop_response
While loop_response == 'y'
```

The three attributes of a test after loop are present. The action part consists of the 6 lines that prompt for data and then displays the total of the two ages. The update of the flag is the displaying the question and getting the answer for the variable loop_response. The test is the equality relational comparison of the value in the flag variable to the lower case character of y.

This type of loop control is called an event controlled loop. The flag updating is an event where someone decides if they want the loop to execute again.

Using indentation with the alignment of the loop actions and flag update is the normal industry practice.

Infinite Loops

At this point, it is worth mentioning that good programming always provides for a method to ensure that the loop question will eventually be false so that the loop will stop executing and the program continues with the next line of code. However, if this does not happen, then the program is in an infinite loop. Infinite loops are a bad thing. Consider the following code:

```
loop_response = 'y'
Do
```

```
    Output "What is your age? "
    Input user_age
    Output "What is your friend's age? "
    Input friend_age
    Output "Together your ages add up to: "
    Output user_age + friend_age
While loop_response == 'y'
```

The programmer assigned a value to the flag before the loop and forgot to update the flag. Every time the test expression is asked it will always be true. Thus, an infinite loop because the programmer did not provide a way to exit the loop (he forgot to update the flag).

Consider the following code:

```
do
    Output "What is your age? "
    Input user_age
    Output "What is your friend's age? "
    Input friend_age
    Output "Together your ages add up to: "
    Output age_user + friend_age
    Output "Do you want to try it again? y or n "
    Input loop_response
While loop_response = 'y'
```

No matter what the user replies during the flag update, the test expression does not do a relational comparison but does an assignment. It assigns 'y' to the variable and asks if 'y' is true? Since all non-zero values are treated as representing true, the answer to the text question is true. Viola, you have an infinite loop.

Key Terms

action item
An attribute of iteration control structures.

at least once
Indicating that test after loops execute the action at least once.

do while
A test after iteration control structure.

infinite loop
A sequence of instructions which loops endlessly, either due to the loop having no terminating condition, having one that can never be met, or one that causes the loop to start over.[3]

repeat until
A test after iteration control structure alternative available in some programming languages.

3. Wikipedia: Infinite loop

test item
: An attribute of iteration control structures.

update item
: An attribute of iteration control structures.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Flag Concept

KENNETH LEROY BUSBEE

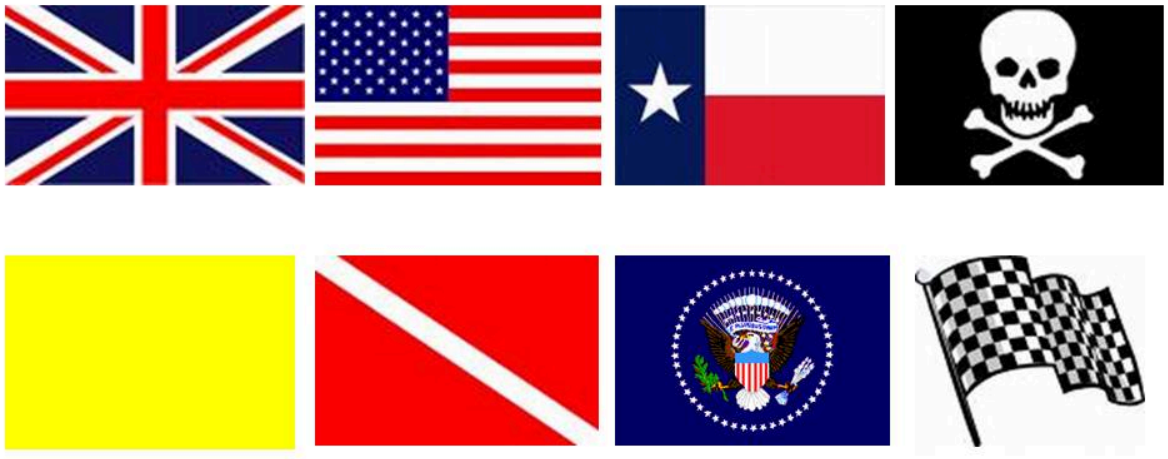
Overview

Flags are commonly used to control or to indicate the intermediate state or outcome of particular operations.[1]

Discussion

For centuries flags have been used as a signal to let others know something about the group or individual that is displaying, flying or waving the flag. There are country flags and state flags. Ships at sea flew the flag of their country. Pirates flew the skull and crossbones. A yellow flag was used for quarantine, usually the plague. Even pirates stayed away. Today, some people might recognize the flag used by scuba divers. The Presidents of most countries have a flag. At a race car event, they use the checkered flag to indicate the race is over.

1. Wikipedia: Bit field



Various Flags

Computer programming uses the concept of a flag in the same way that physical flags are used. A flag is anything that signals some information to the person looking at it.

Computer Implementation

Any variable or constant that holds data can be used as a flag. You can think of the storage location as a flagpole. The value stored within the variable conveys some meaning and you can think of it as being the flag. An example might be a variable named: gender which is of the character data type. The two values commonly stored in the variable are: 'F' and 'M', meaning female and male. Then, somewhere within a program we might look at the variable to make a decision:

flag controlling an if then control structure

```
if gender equals 'F'
    display "Are you pregnant?"
    get answer from user store in pregnant variable
```

Looking at the flag implies comparing the value in the variable to another value (a constant or the value in another variable) using a relational operator (in our above example: equality).

Control structures are "controlled" by using a **test expression** which is usually a **Boolean expression**. Thus, the flag concept of "looking" at the value in the variable and comparing it to another value is fundamental to understanding how all control structures work.

Two Flags with the Same Meaning

Sometimes we will use an iteration control structure of do while to allow us to decide if we want to do the loop action again. A variable might be named “loop_response” with the user prompted for their answer of ‘y’ for yes or ‘n’ for no. Once the answer is retrieved from the keyboard and stored in our flag variable of “loop_response” the test expression to control the loop might be:
simple flag comparison

```
loop_response equals 'y'
```

This is fine but what if the user accidentally has on the caps lock. Then the response of ‘Y’ would not have the control structure loop and perform the action again. The solution lies in looking at the flag twice. Consider:
complex flag comparison

```
loop_response equals 'y' or loop_response equals 'Y'
```

We look to see if the flag is either a lower case y or an upper case Y by using a more complex Boolean expression with both relational and logical operators.

Multiple Flags in One Byte

Within assembly language programming and in many technical programs that control special devices; the use of a single byte to represent several flags is common. This is accomplished by having each one of the 8 bits that make up the byte represent a flag. Each bit has a value of either 1 or 0 and can represent true and false, on or off, yes or no, etc.

Key Terms

flag
A variable used to store information that will normally be used to control the program.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

For Loop

KENNETH LEROY BUSBEE

Overview

A **for loop** is a control flow statement for specifying iteration, which allows code to be executed repeatedly. A for loop has two parts: a header specifying the iteration, and a body which is executed once per iteration. The header often declares an explicit loop counter or loop variable, which allows the body to know which iteration is being executed. For loops are typically used when the number of iterations is known before entering the loop. For loops can be thought of as shorthands for while loops which increment and test a loop variable.[1]

Discussion

Introduction to Test Before Loops

There are two commonly used test before loops in the iteration (or repetition) category of control structures. They are: while and for. This module covers the: for.

Understanding Iteration in General – for

In many programming languages, the for loop is used exclusively for counting; that is to repeat a loop action as it either counts up or counts down. There is a starting value and a stopping value. The question that controls the loop is a test expression that compares the starting value to the stopping value. This expression is a Boolean expression and is usually using the relational operators of either less than (for counting up) or greater than (for counting down). The term loop comes from the circular looping motion that occurs when using flowcharting. The basic form of the for loop (counting up) is as follows:

```
for
    initialization of the starting value
    starting value is less than the stopping value
    some statements or action
    some statements or action
    some statements or action
    increment the starting value
```

1. Wikipedia: For loop

It might be best to understand the for loop by understanding a while loop acting like a counting loop. Let's consider;

```
initialization of the starting value
while the starting value is less than the stopping value
    some statements or action
    some statements or action
    some statements or action
    increment the starting value
```

Within the for control structure, there are four attributes to a properly working loop. They are:

- Initializing the flag – done once
- Test expression
- Action or actions
- Update of the flag

The initialization of the flag is not technically part of the while control structure, but it is usually part of the for control structure. The English phrasing is, "For x is 1; x less than 3; do the following actions; increment x; loop back to the test expression". This is doing the action on the true. When the test expression is false, you stop the loop and go on with the next item in the program. Notice, because this is a test before loop the action **might not happen**. It is called a test before loop because the test comes before the action. It is also sometimes called a pre-test loop, meaning the test is pre (or Latin for before) the action and update.

An Example

```
For counter = 0, counter < 5, counter += 1
    Output "I love ice cream!"
```

The four attributes of a test before loop (remember the for loop is one example of a test before loop) are present.

- The initialization of the flag to a value of 0.
- The test is the less than relational comparison of the value in the flag variable to the constant value of 5.
- The action part consists of the 1 line of output.
- The update of the flag is done with the increment operator.

Using indentation with the alignment of the loop actions is the normal industry practice.

Infinite Loops

At this point, it is worth mentioning that good programming always provides for a method to

ensure that the loop question will eventually be false so that the loop will stop executing and the program continues with the next line of code. However, if this does not happen, then the program is in an infinite loop. Infinite loops are a bad thing. Consider the following code:

```
For counter = 0, counter < 5
    Output "I love ice cream!"
```

The programmer assigned a value to the flag during the initialization step which is correct. However, they forgot to update the flag (the update step is missing). Every time the test expression is asked it will always be true. Thus, an infinite loop because the programmer did not provide a way to exit the loop (he forgot to update the flag).

Key Terms

for
A test before iteration control structure typically used for counting.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Branching Statements

KENNETH LEROY BUSBEE

Overview

A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction sequence and thus deviate from its default behavior of executing instructions in order.[1] Common branching statements include `break`, `continue`, `return`, and `goto`.

Discussion

Branching statements allow the flow of execution to jump to a different part of the program. The common branching statements used within other control structures include: `break`, `continue`, `return`, and `goto`. The goto is rarely used in modular structured programming. Additionally, we will add to our list of branching items a pre-defined function commonly used in programming languages of: `exit`.

Examples

break

The break is used in one of two ways; with a switch to make it act like a case structure or as part of a looping process to break out of the loop. The following gives the appearance that the loop will execute 8 times, but the break statement causes it to stop during the fifth iteration.

```
counter = 0;
While counter < 8
    Output counter
    If counter == 4
        break
    counter += 1
```

1. Wikipedia: Branch (computer science)

continue

The following gives the appearance that the loop will print to the monitor 8 times, but the continue statement causes it not to print number 4.

```
For counter = 0, counter < 8, counter += 1
    If counter == 4
        continue
    Output counter
```

return

The return statement exits a function and returns to the statement where the function was called.

```
Function DoSometing
    statements
Return <optional return value>
```

goto

The goto structure is typically not accepted in good structured programming. However, some programming languages allow you to create a label with an identifier name followed by a colon. You use the command word `goto` followed by the label.

```
some lines of code;
goto label;                 // jumps to the label
some lines of code;
some lines of code;
some lines of code;
label: some statement;      // Declared label
some lines of code;
```

exit

Although exit is technically a pre-defined function, it is covered here because of its common usage in programming. A good example is the opening a file and then testing to see if the file was actually opened. If not, we have an error that usually indicates that we want to prematurely stop the execution of the program. The exit function terminates the running of the program and in the

process returns an integer value back to the operating system. It fits the definition of branching which is to jump to some other place in the program.

Key Terms

branching statements
: Allow the flow of execution to jump to a different part of the program.

break
: A branching statement that terminates the existing structure.

continue
: A branching statement that causes a loop to stop its current iteration and begin the next one.

exit
: A predefined function used to prematurely stop a program and return to the operating system.

goto
: An unstructured branching statement that causes the logic to jump to a different place in the program.

return
: A branching statement that causes a function to jump back to the function that called it.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Increment and Decrement Operators

KENNETH LEROY BUSBEE

Overview

Increment and decrement operators are unary operators that add or subtract one from their operand, respectively. They are commonly implemented in imperative programming languages.[1]

Discussion

The idea of increment or decrement is to either add or subtract 1 from a variable that is usually acting as a flag. Using a variable named counter; in generic terms, for example:

```
increment the counter
```

The concept is:

```
counter is assigned counter + 1
```

That is you fetch the existing value of the counter and add one then store the answer back into the variable counter. Many programming languages allow their increment and decrement operators to only be used with the integer data type. Programmers will sometimes use inc and dec as abbreviations for increment and decrement respectively.

Operator symbols and/or names vary with different programming languages. Several programming languages support increment and decrement operators:

Operator	Meaning
`++`	increment, **two plus signs**
`--`	decrement, **two minus signs**

Code Examples

Basic Concept

Within C++, C#, Java, and JavaScript programming languages, the increment and decrement operators are often used in this simple generic way. The increment operator is represented by two plus signs in a row. Examples:

```
counter = counter + 1;
```

1. Wikipedia: Increment and decrement operators

```
counter += 1;
counter++;
++counter;
```

As statements, the four examples all do the same thing. They add 1 to the value of whatever is stored in counter. The decrement operator is represented by two minus signs in a row. They would subtract 1 from the value of whatever was in the variable being decremented. The precedence of increment and decrement depends on if the operator is attached to the right of the operand (postfix) or to the left of the operand (prefix). Note that postfix and prefix do not have the same precedence.

Postfix Increment

Postfix increment says to use my existing value then when you are done with the other operators; increment me. An example:

```
int oldest = 44;
age = oldest++;
```

The first use of the oldest variable is an Rvalue context where the existing value of 44 is pulled or fetched and then assigned to the variable age; then the variable oldest is incremented with its value changing from 44 to 45. This seems to be a violation of precedence because increment is higher precedence than assignment. But that is how postfix increment works.

Prefix Increment

Prefix increment says to increment me now and use my new value in any calculation. An example:

```
int oldest = 44;
age = ++oldest;
```

The variable oldest is incremented with the new value changing it from 44 to 45; then the new value is assigned to age.

In postfix age is assigned 44 in prefix age is assigned 45. One way to help remember the difference is to think of postfix as being polite (use my existing value and return to increment me after the other operators are done) whereas prefix has an ego (I am important so increment me first and use my new value for the rest of the evaluations).

Allowable Data Types

Within some programming languages, increment and decrement can be used only on the integer data type. Other languages expand this not only to all of the integer family but also to the floating-point family (float and double). Incrementing 3.87 will change the value to 4.87. Decrementing ‘C’

will change the value to 'B'. Remember the ASCII character values are really one-byte unsigned integers (domain from 0 to 255).

Exercises

Evaluate the following items using increment or decrement:

1. True or false: x = x +1 and x+=1 and x++ all accomplish increment?
2. Given: int y = 19; and int z; what values will y and z have after: z = y–;
3. Given: double x = 7.77; and int y; what values will x and y have after: y = ++x;
4. Is this ok? Why or why not? 6 * ++(age -3)

Key Terms

decrement
Subtracting one from the value of a variable.
increment
Adding one to the value of a variable.
postfix
Placing the increment or decrement operator to the right of the operand.
prefix
Placing the increment or decrement operator to the left of the operand.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Integer Overflow

KENNETH LEROY BUSBEE

Overview

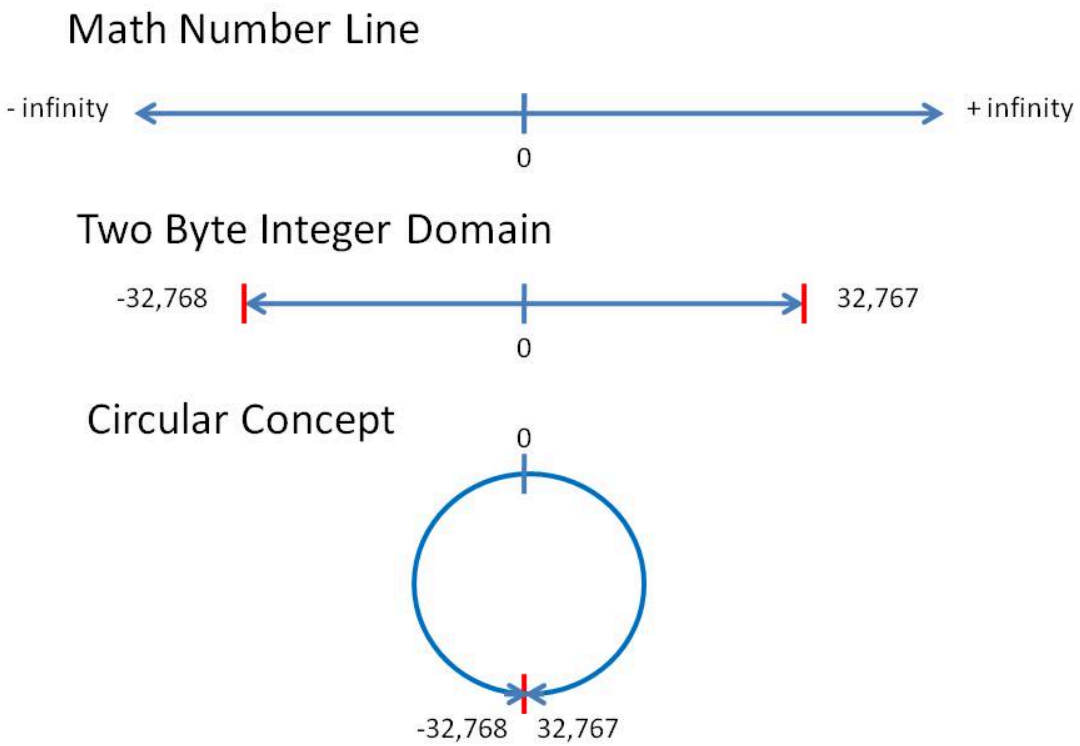
Integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of bits – either larger than the maximum or lower than the minimum representable value.[1]

The most common result of an overflow is that the least significant representable bits of the result are stored; the result is said to wrap around the maximum (i.e. modulo power of two). An overflow condition may give results leading to unintended behavior. In particular, if the possibility has not been anticipated, overflow can compromise a program's reliability and security.[2]

Discussion

There are times when character and integer data types are lumped together because they both act the same (often called the integer family). Maybe we should say they act differently than the floating-point data types. The integer family values jump from one value to another. There is nothing between 6 and 7 nor between 'A' and 'B'. It could be asked why not make all your numbers floating-point data types. The reason is twofold. First, some things in the real world are not fractional. A dog, even with only 3 legs, is still one dog not three-fourths of a dog. Second, the integer data type is often used to control program flow by counting (counting loops). The integer family has a circular wrap-around feature. Using a two-byte integer, the next number bigger than 32767 is negative 32768 (character acts the same way going from 255 to 0. We could also reverse that to be the next smaller number than negative 32768 is positive 32767. This can be shown by using a normal math line, limiting the domain and then connecting the two ends to form a circle.

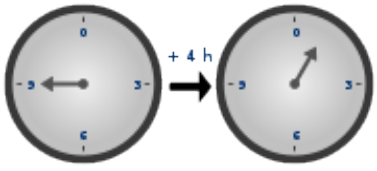
1. Wikipedia: Integer overflow
2. Wikipedia: Integer overflow



This circular nature of the integer family works for both integer and character data types. In theory, it should work for the Boolean data type as well; but in most programming languages it does not for various technical reasons.

"In mathematics, modular arithmetic (sometimes called clock arithmetic) is a system of arithmetic for integers where numbers "wrap around" after they reach a certain value — the modulus. ...

A familiar use of modular arithmetic is its use in the 12-hour clock the arithmetic of time-keeping in which the day is divided into two 12 hour periods. If the time is 7:00 now, then 8 hours later it will be 3:00. Regular addition would suggest that the later time should be 7 + 8 = 15, but this is not the answer because clock time "wraps around" every 12 hours; there is no "15 o'clock". Likewise, if the clock starts at 12:00 (noon) and 21 hours elapse, then the time will be 9:00 the next day, rather than 33:00. Since the hour number starts over when it reaches 12, this is arithmetic modulo 12.



Time-keeping on a clock gives an example of modular arithmetic." (Modular arithmetic from Wikipedia)

The use of the modulus operator in integer division is tied to the concepts used in modular arithmetic.

Implications When Executing Loops

If a programmer sets up a counting loop incorrectly, usually one of three things happen:

- Infinite loop – usually caused by missing update attribute.
- Loop never executes – usually, the text expression is wrong with the direction of the less than or greater than relationship needing to be switched.
- Loop executes more times than desired – update not properly handled. Usually, the direction of counting (increment or decrement) need to be switched.

Let's give an example of the loop executing for what appears to be for infinity (the third item on our list).

```
for int x = 0, x < 10, x--
    Output x
```

The above code accidentally decrements and the value of x goes in a negative way towards -2147483648 (the largest negative value in a normal four-byte signed integer data type). It might take a while (thus it might appear to be in an infinite loop) for it to reach the negative 2 billion-plus value, before finally decrementing to positive 2147483647 which would, incidentally, stop the loop execution.

Key Terms

circular nature
Connecting the negative and positive ends of the domain of an integer family data type.
loop control
Making sure the attributes of a loop are properly handled.
modular arithmetic
A system of arithmetic for integers where numbers "wrap around".

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Nested For Loops

KENNETH LEROY BUSBEE

Overview

Nested for loops places one for loop inside another for loop. The inner loop is repeated for each iteration of the outer loop.

Discussion

Nested Control Structures

We are going to first introduce the concept of nested control structures. Nesting is a concept that places one item inside of another. Consider:

```
if expression
    true action
else
    false action
```

This is the basic form of the if then else control structure. Now consider:

```
if age is less than 18
    you can't vote
    if age is less than 16
        you can't drive
    else
        you can drive
else
    you can vote
    if age is less than 21
        you can't drink
    else
        you can drink
```

As you can see we simply included as part of the “true action” a statement and another if then else control structure. We did the same (nested another if then else) for the “false action”. In our example, we nested if then else control structures. Nesting could have an if then else within a while loop. Thus, the concept of nesting allows the mixing of the different categories of control structures.

Many complex logic problems require using nested control structures. By nesting control structures (or placing one inside another) we can accomplish almost any **complex logic** problem.

An Example – Nested for loops

Here is an example of a 10 by 10 multiplication table:

```
      1 |   2 |   3 |   4 |   5 |   6 |   7 |   8 |   9 |  10 |
   ---------------------------------------------------------------
 1 !   1 |   2 |   3 |   4 |   5 |   6 |   7 |   8 |   9 |  10 |
 2 !   2 |   4 |   6 |   8 |  10 |  12 |  14 |  16 |  18 |  20 |
 3 !   3 |   6 |   9 |  12 |  15 |  18 |  21 |  24 |  27 |  30 |
 4 !   4 |   8 |  12 |  16 |  20 |  24 |  28 |  32 |  36 |  40 |
 5 !   5 |  10 |  15 |  20 |  25 |  30 |  35 |  40 |  45 |  50 |
 6 !   6 |  12 |  18 |  24 |  30 |  36 |  42 |  48 |  54 |  60 |
 7 !   7 |  14 |  21 |  28 |  35 |  42 |  49 |  56 |  63 |  70 |
 8 !   8 |  16 |  24 |  32 |  40 |  48 |  56 |  64 |  72 |  80 |
 9 !   9 |  18 |  27 |  36 |  45 |  54 |  63 |  72 |  81 |  90 |
10 !  10 |  20 |  30 |  40 |  50 |  60 |  70 |  80 |  90 | 100 |
```

We might also see that the answers could be designed as a collection of cells (each cell being exactly six spaces wide). The pseudocode to produce part of the table is:

```
For row = 1, row <= 3, row += 1
    For column = 1, column <= 3, column += 1
        Output row * column
        Output "\t"
    Output "\n"
```

Key Terms

complex logic
Often solved with nested control structures.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Loop Examples

DAVE BRAUNSCHWEIG

Counting

Pseudocode

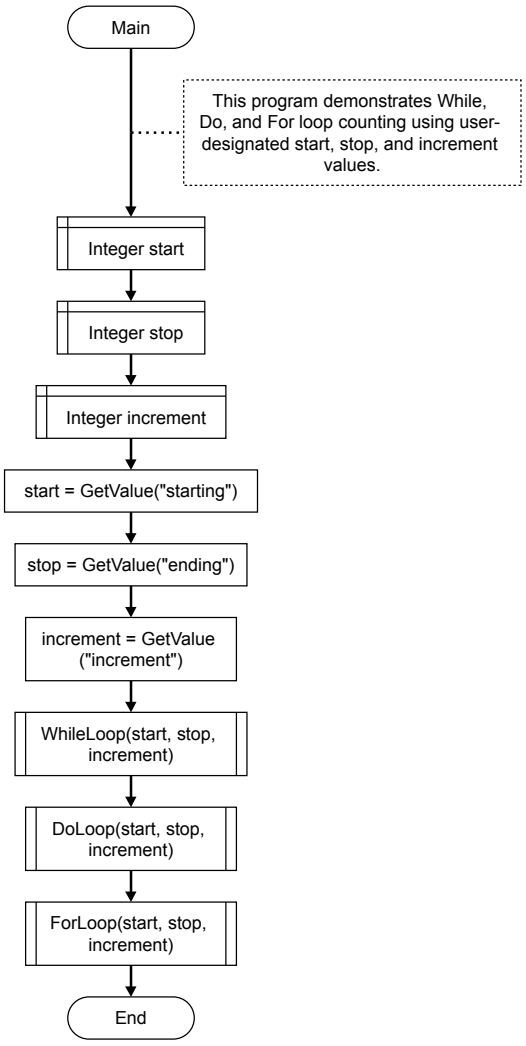
```
... This program demonstrates While, Do, and For loop counting using user-d

Function Main
    Declare Integer start
    Declare Integer stop
    Declare Integer increment

    Assign start = GetValue("starting")
    Assign stop = GetValue("ending")
    Assign increment = GetValue("increment")
    Call WhileLoop(start, stop, increment)
    Call DoLoop(start, stop, increment)
    Call ForLoop(start, stop, increment)
End

Function GetValue (String name)
    Declare Integer value

    Output "Enter " & name & " value:"
    Input value
Return Integer value

Function WhileLoop (Integer start, Integer stop, Integer increment)
    Output "While loop counting from " & start & " to " & stop & " by " & i
    Declare Integer count

    Assign count = start
    While count <= stop
        Output count
        Assign count = count + increment
    End
End
```

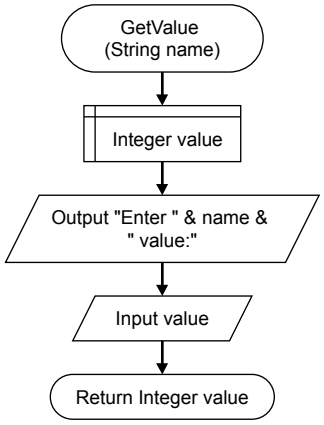
```
Function DoLoop (Integer start, Integer stop, Integer increment)
    Output "Do loop counting from " & start & " to " & stop & " by " & incr
    Declare Integer count

    Assign count = start
    Loop
        Output count
        Assign count = count + increment
    Do count <= stop
End

Function ForLoop (Integer start, Integer stop, Integer increment)
    Output "For loop counting from " & start & " to " & stop & " by " & inc
    Declare Integer count

    For count = start to stop step increment
        Output count
    End
End
```

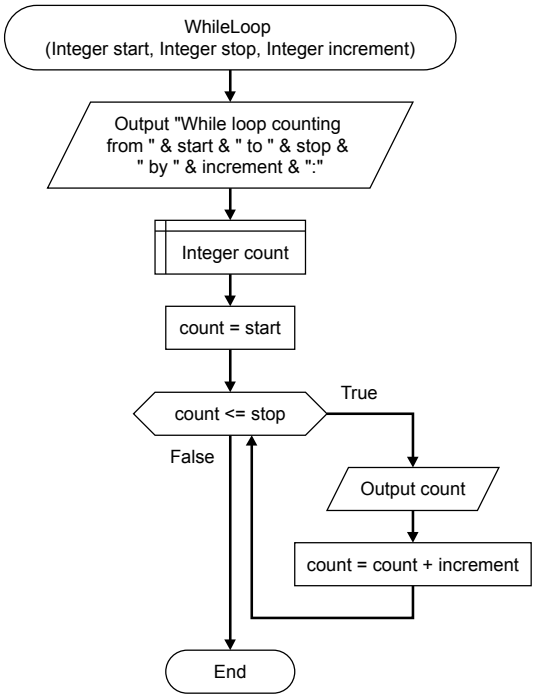
Output

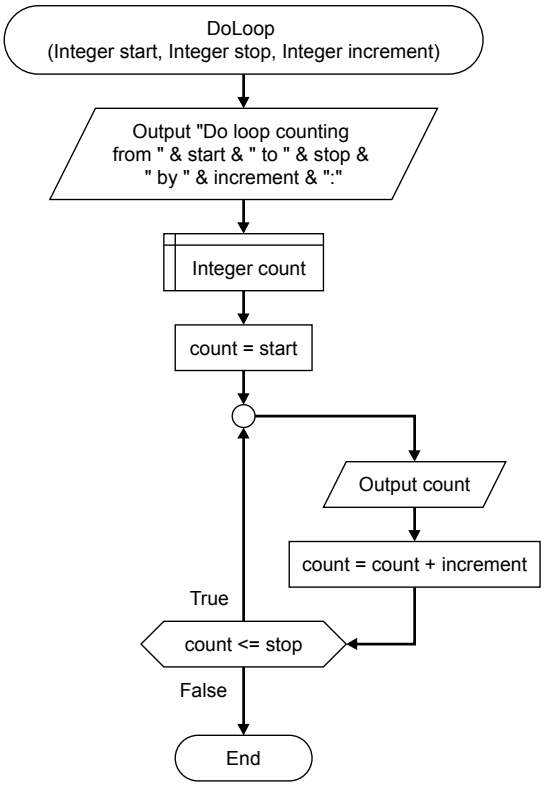
```
Enter starting value:
1
Enter ending value:
3
Enter increment value:
1
While loop counting from 1 to 3 by 1:
1
2
3
Do loop counting from 1 to 3 by 1:
1
2
3
For loop counting from 1 to 3 by 1:
1
2
3
```

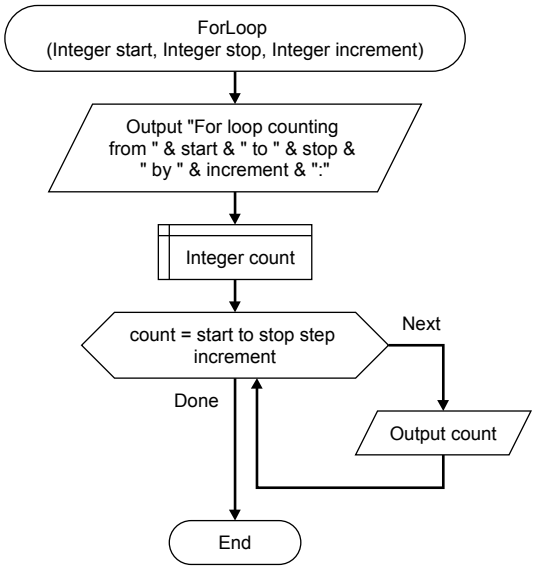
Flowchart


Main
This program demonstrates While, Do, and For loop counting using user-designated start, stop, and increment values.
Integer start
Integer stop
Integer increment
start = GetValue("starting")
stop = GetValue("ending")
increment = GetValue ("increment")
WhileLoop(start, stop, increment)
DoLoop(start, stop, increment)
ForLoop(start, stop, increment)
End


GetValue (String name)
Integer value
Output "Enter " & name & " value:"
Input value
Return Integer value







References

- [Wikiversity: Computer Programming]

C++ Examples

DAVE BRAUNSCHWEIG

Counting

```
// This program demonstrates While, Do, and For loop counting using
// user-designated start, stop, and increment values.
//
// References:
//     https://en.wikibooks.org/wiki/C%2B%2B_Programming

#include

using namespace std;

int getValue(string name);
void whileLoop(int start, int stop, int increment);
void doLoop(int start, int stop, int increment);
void forLoop(int start, int stop, int increment);

int main() {
    int start = getValue("starting");
    int stop = getValue("ending");
    int increment = getValue("increment");

    whileLoop(start, stop, increment);
    doLoop(start, stop, increment);
    forLoop(start, stop, increment);

    return 0;
}

int getValue(string name) {
    int value;

    cout << "Enter " << name << " value:" <> value;

    return value;
}
```

```
void whileLoop(int start, int stop, int increment) {
    cout << "While loop counting from " << start << " to " <<
        stop << " by " << increment << ":" << endl;

    int count = start;
    while (count <= stop) {
        cout << count << endl;
        count = count + increment;
    }
}

void doLoop(int start, int stop, int increment) {
    cout << "Do loop counting from " << start << " to " <<
        stop << " by " << increment << ":" << endl;

    int count = start;
    do {
        cout << count << endl;
        count = count + increment;
    } while (count <= stop);
}

void forLoop(int start, int stop, int increment) {
    cout << "For loop counting from " << start << " to " <<
        stop << " by " << increment << ":" << endl;

    for (int count = start; count <= stop; count += increment) {
        cout << count << endl;
    }
}
```

Output

```
Enter starting value:
1
Enter ending value:
3
Enter increment value:
1
While loop counting from 1 to 3 by 1:
1
2
```

```
3
Do loop counting from 1 to 3 by 1:
1
2
3
For loop counting from 1 to 3 by 1:
1
2
3
```

References

- Wikiversity: Computer Programming

C# Examples

DAVE BRAUNSCHWEIG

Counting

```
// This program demonstrates While, Do, and For loop counting using
// user-designated start, stop, and increment values.
//
// References:
//     https://en.wikibooks.org/wiki/C_Sharp_Programming

using System;

public class Loops
{
    public static void Main(string[] args)
    {
        int start = GetValue("starting");
        int stop = GetValue("ending");
        int increment = GetValue("increment");

        WhileLoop(start, stop, increment);
        DoLoop(start, stop, increment);
        ForLoop(start, stop, increment);
    }

    public static int GetValue(string name)
    {
        Console.WriteLine("Enter " + name + " value:");
        string input = Console.ReadLine();
        int value = Convert.ToInt32(input);

        return value;
    }

    public static void WhileLoop(int start, int stop, int increment)
    {
        Console.WriteLine("While loop counting from " + start + " to " +
            stop + " by " + increment + ":");
```

```
        int count = start;
        while (count <= stop)
        {
            Console.WriteLine(count);
            count = count + increment;
        }
    }

    public static void DoLoop(int start, int stop, int increment)
    {
        Console.WriteLine("Do loop counting from " + start + " to " +
            stop + " by " + increment + ":");

        int count = start;
        do
        {
            Console.WriteLine(count);
            count = count + increment;
        }
        while (count <= stop);
    }

    public static void ForLoop(int start, int stop, int increment)
    {
        Console.WriteLine("For loop counting from " + start + " to " +
            stop + " by " + increment + ":");

        for (int count = start; count <= stop; count += increment)
        {
            Console.WriteLine(count);
        }
    }
}
```

Output

```
Enter starting value:
1
Enter ending value:
3
Enter increment value:
1
```

```
While loop counting from 1 to 3 by 1:
1
2
3
Do loop counting from 1 to 3 by 1:
1
2
3
For loop counting from 1 to 3 by 1:
1
2
3
```

References

- Wikiversity: Computer Programming

Java Examples

DAVE BRAUNSCHWEIG

Counting

```
// This program demonstrates While, Do, and For loop counting using
// user-designated start, stop, and increment values.
//
// References:
//     https://en.wikibooks.org/wiki/Java_Programming

import java.util.*;

public class Main {
    private static Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        int start = getValue("starting");
        int stop = getValue("ending");
        int increment = getValue("increment");

        whileLoop(start, stop, increment);
        doLoop(start, stop, increment);
        forLoop(start, stop, increment);
    }

    public static int getValue(String name) {
        System.out.println("Enter " + name + " value:");
        int value = input.nextInt();

        return value;
    }

    public static void whileLoop(int start, int stop, int increment) {
        System.out.println("While loop counting from " + start + " to " +
            stop + " by " + increment + ":");

        int count = start;
        while (count <= stop) {
            System.out.println(count);
```

```
            count = count + increment;
        }
    }

    public static void doLoop(int start, int stop, int increment) {
        System.out.println("Do loop counting from " + start + " to " +
            stop + " by " + increment + ":");

        int count = start;
        do {
            System.out.println(count);
            count = count + increment;
        } while (count <= stop);
    }

    public static void forLoop(int start, int stop, int increment) {
        System.out.println("For loop counting from " + start + " to " +
            stop + " by " + increment + ":");

        for (int count = start; count <= stop; count += increment) {
            System.out.println(count);
        }
    }
}
```

Output

```
Enter starting value:
1
Enter ending value:
3
Enter increment value:
1
While loop counting from 1 to 3 by 1:
1
2
3
Do loop counting from 1 to 3 by 1:
1
2
3
For loop counting from 1 to 3 by 1:
```

```
1
2
3
```

References

- Wikiversity: Computer Programming

JavaScript Examples

DAVE BRAUNSCHWEIG

Counting

```
// This program demonstrates While, Do, and For loop counting using
// user-designated start, stop, and increment values.
//
// References:
//   https://en.wikibooks.org/wiki/JavaScript

main()

function main() {
    var start = getValue("starting");
    var stop = getValue("ending");
    var increment = getValue("increment");

    whileLoop(start, stop, increment);
    doLoop(start, stop, increment);
    forLoop(start, stop, increment);
}

function getValue(name) {
    output("Enter " + name + " value:");
    var value = Number(input());
    return value;
}

function whileLoop(start, stop, increment) {
    output("While loop counting from " + start + " to " + stop +
        " by " + increment + ":");

    var count = start;
    while (count <= stop) {
        output(count);
        count = count + increment;
    }
}
```

```
function doLoop(start, stop, increment) {
    output("Do loop counting from " + start + " to " + stop +
        " by " + increment + ":");

    var count = start;
    do {
        output(count);
        count = count + increment;
    } while (count <= stop);
}

function forLoop(start, stop, increment) {
    output("For loop counting from " + start + " to " + stop +
        " by " + increment + ":");

    for (var count = start; count <= stop; count += increment) {
        output(count);
    }
}

function input(text) {
  if (typeof window === 'object') {
    return prompt(text)
  }
  else if (typeof console === 'object') {
    const rls = require('readline-sync');
    var value = rls.question(text);
    return value;
  }
  else {
    output(text);
    var isr = new java.io.InputStreamReader(java.lang.System.in);
    var br = new java.io.BufferedReader(isr);
    var line = br.readLine();
    return line.trim();
  }
}

function output(text) {
  if (typeof document === 'object') {
    document.write(text);
  }
  else if (typeof console === 'object') {
```

```
    console.log(text);
  }
  else {
    print(text);
  }
}
```

Output

```
Enter starting value:
1
Enter ending value:
3
Enter increment value:
1
While loop counting from 1 to 3 by 1:
1
2
3
Do loop counting from 1 to 3 by 1:
1
2
3
For loop counting from 1 to 3 by 1:
1
2
3
```

References

- Wikiversity: Computer Programming

Python Examples

DAVE BRAUNSCHWEIG

Counting

```
# This program demonstrates While, Do, and For loop counting using
# user-designated start, stop, and increment values.
#
# References:
#     https://en.wikibooks.org/wiki/Python_Programming


def get_value(name):
    print("Enter " + name + " value:")
    value = int(input())
    return value


def while_loop(start, stop, increment):
    print("While loop counting from " + str(start) + " to " +
        str(stop) + " by " + str(increment) + ":")
    count = start
    while count <= stop:
        print(count)
        count = count + increment


def do_loop(start, stop, increment):
    print("Do loop counting from " + str(start) + " to " +
        str(stop) + " by " + str(increment) + ":")
    count = start
    while True:
        print(count)
        count = count + increment
        if not(count <= stop):
            break
```

```
def for_loop(start, stop, increment):
    print("For loop counting from " + str(start) + " to " +
```

```
        str(stop) + " by " + str(increment) + ":")
    for count in range(start, stop + increment, increment):
        print(count)


def main():
    start = get_value("starting")
    stop = get_value("ending")
    increment = get_value("increment")
    while_loop(start, stop, increment)
    do_loop(start, stop, increment)
    for_loop(start, stop, increment)


main()
```

Output

```
Enter starting value:
1
Enter ending value:
3
Enter increment value:
1
While loop counting from 1 to 3 by 1:
1
2
3
Do loop counting from 1 to 3 by 1:
1
2
3
For loop counting from 1 to 3 by 1:
1
2
3
```

References

- Wikiversity: Computer Programming

Swift Examples

DAVE BRAUNSCHWEIG

Counting

```
// This program demonstrates While, Do, and For loop counting using
// user-designated start, stop, and increment values.
//
// References:
//     https://developer.apple.com/library/content/documentation/Swift/Conc

import Foundation

func getValue(name: String) -> Int {
    var value : Int

    print("Enter " + name + " value:")
    value = Int(readLine()!)!
    return value
}

func whileLoop(start: Int, stop: Int, increment: Int) {
    print("While loop counting from " + String(start) + " to " +
        String(stop) + " by " + String(increment) + ":")

    var count : Int

    count = start
    while count <= stop {
        print(count)
        count = count + increment
    }
}

func doLoop(start: Int, stop: Int, increment: Int) {
    print("Do loop counting from " + String(start) + " to " +
        String(stop) + " by " + String(increment) + ":")

    var count : Int
```

```
    count = start
    repeat {
        print(count)
        count = count + increment
    } while count <= stop
}

func forLoop(start: Int, stop: Int, increment: Int) {
    print("For loop counting from " + String(start) + " to " +
        String(stop) + " by " + String(increment) + ":")

    for count in stride(from: start, through: stop, by: increment) {
        print(count)
    }
}

func main() {
    var start : Int
    var stop : Int
    var increment : Int

    start = getValue(name: "starting")
    stop = getValue(name: "ending")
    increment = getValue(name: "increment")

    whileLoop(start: start, stop: stop, increment: increment)
    doLoop(start: start, stop: stop, increment: increment)
    forLoop(start: start, stop: stop, increment: increment)
}

main()
```

Output

```
Enter starting value:
1
Enter ending value:
3
Enter increment value:
1
While loop counting from 1 to 3 by 1:
1
```

```
2
3
Do loop counting from 1 to 3 by 1:
1
2
3
For loop counting from 1 to 3 by 1:
1
2
3
```

References

- Wikiversity: Computer Programming

Practice: Loops

KENNETH LEROY BUSBEE

Review Questions

True / False

1. The do while and repeat until structure act exactly the same.
2. Students sometimes confuse assignment and equality.
3. The repeat until looping control structure is available in all programming languages.
4. Because flags are often used, they are usually a special data type.
5. The do while is a test before loop.
6. Only for loops can be counting loops.
7. The integer data type has modular arithmetic attributes.
8. The escape code of \n is part of formatting output.
9. Nested for loops is not allowed in the C++ programming language.
10. Counting loops use all four of the loop attributes.

Answers:

1. false
2. true
3. false
4. false
5. false
6. false
7. true
8. true
9. false
10. true

Activities

Complete the following activities using pseudocode, a flowcharting tool, or your selected programming language. Use separate functions for input, each type of processing, and output. Avoid global variables by passing parameters and returning results. Create test data to validate the accuracy of each program. Add comments at the top of the program and include references to any resources used.

While Loops

Complete the following using a while loop structure.

1. Create a program that uses a loop to generate a list of multiplication expressions for a given

value. Ask the user to enter the value and the number of expressions to be displayed. For example, a list of three expressions for the value 1 would be:

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
```

A list of five expressions for the value 3 would be:

```
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
```

2. Review MathsIsFun: Definition of Average. Create a program that asks the user to enter grade scores. Start by asking the user how many scores they would like to enter. Then use a loop to request each score and add it to a total. Finally, calculate and display the average for the entered scores.
3. Review MathsIsFun: Pi. Write a program that uses the Nilakantha series to calculate Pi based on a given number of iterations entered by the user.

Do While / Repeat Until Loops

Complete the following using a do while / repeat until loop structure.

1. Review MathsIsFun: Definition of Average. Create a program that asks the user to enter grade scores. Use a loop to request each score and add it to a total. Continue accepting scores until the user enters a negative value. Finally, calculate and display the average for the entered scores.
2. Review Khan Academy: A guessing game. Write a program that allows the user to think of a number between 0 and 100, inclusive. Then have the program try to guess the user's number. Start at the midpoint (50) and ask the user if their number is (h)igher, (l)ower, or (e)qual to the guess. If they indicate lower, guess the new midpoint (25). If they indicate higher, guess the new midpoint (75). Continue efficiently guessing higher or lower until they indicate equal, then print the number of guesses required to guess their number and end the program.
3. Add a do while / repeat until loop to any activity from a previous chapter. Continue running the program while the user wants to continue or until the user wants to stop.
4. Add an input validation loop to any activity from a previous chapter. Verify that the input is valid before returning the value. Ask the user to input the value again while the input is invalid.

For Loops

Complete the following using a for loop structure.

1. Create a program that uses a loop to generate a list of multiplication expressions for a given value. Ask the user to enter the value and the number of expressions to be displayed. For example, a list of three expressions for the value 1 would be:

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
```

A list of five expressions for the value 3 would be:

```
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
```

2. Review MathsIsFun: Definition of Average. Create a program that asks the user to enter grade scores. Start by asking the user how many scores they would like to enter. Then use a loop to request each score and add it to a total. Finally, calculate and display the average for the entered scores.
3. Review MathsIsFun: Pi. Write a program that uses the Nilakantha series to calculate Pi based on a given number of iterations entered by the user.

Nested Loops

Complete the following using a nested loop structure.

1. Review MathsIsFun: 10x Printable Multiplication Table. Create a program that uses nested loops to generate a multiplication table. Rather than simply creating a 10 by 10 table, ask the user to enter the starting and ending values. Include row and column labels. For example, the output from 1 to 3 might look like:

```
    1   2   3
1   1   2   3
2   2   4   6
3   3   6   9
```

   The output from 3 to 5 might look like:

```
    3   4   5
3   9  12  15
4  12  16  20
5  15  20  25
```

2. Add a do while / repeat until loop to any activity from this chapter. Continue running the program while the user wants to continue or until the user wants to stop.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++
- Wikiversity: Computer Programming

PART VI

ARRAYS

Overview

This chapter introduces arrays, which may be referred to as lists in some programming languages.

Chapter Outline

Learning Objectives

1. Understand key terms and definitions.
2. Identify static and dynamic arrays and the code structures necessary to process each type.
3. Identify single-dimension arrays and multi-dimensional arrays and the code structures necessary to process each type.
4. Given example pseudocode, flowcharts, and source code, create a program that uses arrays or lists to solve a given problem.

Arrays and Lists

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

An array is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key.[1]

Depending on the language, array types may overlap (or be identified with) other data types that describe aggregates of values, such as lists and strings. Array types are often implemented by array data structures, but sometimes by other means, such as hash tables, linked lists, or search trees.[2] In Python, the built-in array data structure is a list.

Discussion

An array is a sequenced collection of elements of the same data type with a single identifier name. Python lists are similar to arrays in other languages but are not restricted to a single data type. The term 'array' as used in this chapter will generally also apply to Python lists unless otherwise noted.

Arrays can have multiple axes (more than one axis). Each axis is a **dimension**. Thus a single-dimension array is also known as a **list**. A two-dimension array is commonly known as a **table** (a spreadsheet like Excel is a two dimension array). In real life, there are occasions to have data organized into multiple-dimension arrays. Consider a theater ticket with section, row, and seat (three dimensions). Most single-dimension arrays are visualized vertically.

Most programmers are familiar with a special type of array called a string. Strings are basically a single dimension array of characters. Unlike other single dimension arrays, we usually envision a string as a horizontal stream of characters and not vertically as a list.

We refer to the individual values as members (or elements) of the array. Programming languages implement the details of arrays differently. Because there is only one identifier name assigned to the array, we have operators that allow us to reference or access the individual members of an array. The operator commonly associated with referencing array members is the **index** operator. It is important to learn how to define an array and initialize its members.

1. Wikipedia: Array data structure
2. Wikipedia: Array data type

Defining an Array

Language	Example
C++	`int ages[] = {49, 48, 26, 19, 16};`
C#	`int[] ages = {49, 48, 26, 19, 16};`
Java	`int[] ages = {49, 48, 26, 19, 16};`
JavaScript	`var ages = [49, 48, 26, 19, 16];`
Python	`ages = [49, 48, 26, 19, 16]`
Swift	`var ages:[Int] = [49, 48, 26, 19, 16]`

This is the **defining of storage space**. The square brackets `[]` are used here to create the array with five integer members and the identifier name of ages. The assignment with braces (that is a block) establishes the initial values assigned to the members of the array. Note the use of the sequence or comma operator. We could have also done something similar to:

Language	Example	Initial Values
C++	`int ages[5];`	undefined
C#	`int[] ages = new int[5];`	0
Java	`int[] ages = new int[5];`	0
JavaScript	`var ages = Array(5);`	undefined
Python	`ages = [None] * 5`	None

This would have declared the storage space of five integers with the identifier name of ages but their initial values would have been unknown values or initialized as indicated, depending on the programming language. We could assign values later in our program by doing the following (leaving off the semicolons in Python):

```
ages[0] = 49;
ages[1] = 48;
ages[2] = 26;
ages[3] = 19;
ages[4] = 16;
```

Note: The members of the array go from 0 to 4; **NOT** 1 to 5. This is explained in more detail on the next page.

Key Terms

dimension
: An axis of an array.

list
: A single dimension array.

table
: A two-dimension array.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Index Notation

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

Index notation is used to specify the elements of an array.[1] Most current programming languages use square brackets `[]` as the array index operator. Older programming languages, such as FORTRAN, COBOL, and BASIC, often use parentheses `()` as the array index operator.

Discussion

Example:

Language	Example
C++	`int ages[] = {49, 48, 26, 19, 16};` `int myAge = ages[2];`
C#	`int[] ages = {49, 48, 26, 19, 16};` `int myAge = ages[2];`
Java	`int[] ages = {49, 48, 26, 19, 16};` `int myAge = ages[2];`
JavaScript	`var ages = [49, 48, 26, 19, 16];` `int myAge = ages[2];`
Python	`ages = [49, 48, 26, 19, 16]` `my_age = ages[2]`
Swift	`var ages:[Int] = [49, 48, 26, 19, 16]` `var my_age = ages[2]`

As an operator, square brackets either provide the value held by the member of the array (Rvalue) or change the value of member (Lvalue). In the above example, the member that is two offsets from the front of the array (the value 26) is assigned to the variable named myAge. The dereference operator of [2] means to go the 2nd **offset** from the front of the ages array and get the value stored there. In this case, the value would be 26. In most current programming languages, the array members (or elements) are referenced starting at **zero**. The more common way for people to reference a list is by starting with position **one**. Consider:

1. Wikipedia: Index notation

Position	Index	Miss America	Other Contests
zero offsets from the front	`ages[0]`	Winner	1st Place
one offset from the front	`ages[1]`	1st Runner Up	2nd Place
two offsets from the front	`ages[2]`	2nd Runner Up	3rd Place
three offsets from the front	`ages[3]`	3rd Runner Up	4th Place
four offsets from the front	`ages[4]`	4th Runner Up	5th Place

Saying that my cousin is the 2nd Runner-Up in the Miss America contest sounds so much better than saying that she was in 3rd Place. We would be talking about the same position in the array of the five finalists.

```
ages[3] = 20;
```

This is an example of changing an array's value by assigning 20 to the 4th member of the array and replacing the value 19 with 20. This is an Lvalue context because the array is on the left side of the assignment operator.

Key Terms

array member
An element or value in an array.
index
An operator that allows us to reference a member of an array.
offset
The method of referencing array members by starting at zero.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Displaying Array Members

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

To **display** all **array members**, visit each element using a `for` loop and output the element using index notation and the loop control variable.

Discussion

Accessing Array Members

Assume an integer array named ages with five values of 49, 48, 26, 19, and 16, respectively. In pseudocode, this might be written as:

```
Declare Integer Array ages[5]
Assign ages = [49, 48, 26, 19, 16]
```

To display all elements of the array in order, we might write:

```
Output ages[0]
Output ages[1]
Output ages[2]
Output ages[3]
Output ages[4]
```

While this works for short arrays, it is not very efficient, and quickly becomes overwhelming for longer arrays. One of the principles of software development is **don't repeat yourself (DRY)**. Violations of the DRY principle are typically referred to as WET solutions, which is commonly taken to stand for either "write everything twice", "we enjoy typing" or "waste everyone's time".[1]

Rather than repeating ourselves, we can use a for loop to visit each element of the array and use the loop control variable as the array index. Consider the following pseudocode:

```
Declare Integer Array ages[5]
Declare Integer index

Assign ages = [49, 48, 26, 19, 16]

For index = 0 to 4
    Output ages[index]
End
```

1. Wikipedia: Don't repeat yourself

This approach is much more efficient from a programming perspective, and also results in a smaller program. But there is still one more opportunity for improvement. Most programming languages support a built-in method for determining the size of an array. To reduce potential errors and required maintenance, the loop control should be based on the size of the array rather than a hard-coded value. Consider:

```
Declare Integer Array ages[5]
Declare Integer index

Assign ages = [49, 48, 26, 19, 16]

For index = 0 to Size(ages) - 1
    Output ages[index]
End
```

This method allows for **flexible coding**. By writing the for loop in this fashion, we can change the declaration of the array by adding or subtracting members and we don't need to change our for loop code.

Key Terms

don't repeat yourself
: A principle of software development aimed at reducing repetition of software patterns, replacing it with abstractions, or repetition of the same data, using data normalization to avoid redundancy.[2]

flexible coding
: Using the size of an array to determine the number of loop iterations required.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++
- Wikiversity: Computer Programming

2. Wikipedia: Don't repeat yourself

Arrays and Functions

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

In modular programming, specific task functions are often created and used or reused for array processing. Array processing functions are usually passed the array and any data necessary to process the array for the given task.

It should be noted that arrays are passed by reference in most current programming languages. Array processing functions must take care not to alter the array unless intended.

Discussion

Arrays are an important complex data type used in almost all programming. We continue to concentrate on simple one dimension arrays also called a list. Most programmers develop a series of user-defined specific task functions that can be used with an array for normal processing. These functions are usually passed the array along with the number of elements within the array. Some functions also pass another piece of data needed for that particular function's task.

This module covers the displaying the array members on the monitor via calling an **array function** dedicated to that task.

Pseudocode

```
Function Main
    Declare Integer Array ages[5]

    Assign ages = [49, 48, 26, 19, 16]
    Call DisplayArray(ages)
End

Function DisplayArray (Integer Array array)
    Declare Integer index

    For index = 0 to Size(array) - 1
        Output array[index]
    End
End
```

Output

```
49
48
26
19
16
```

Key Terms

array function
A user-defined specific task function designed to process an array.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++
- Wikiversity: Computer Programming

Math Statistics with Arrays

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

Statistics is a branch of mathematics dealing with the collection, organization, analysis, interpretation, and presentation of data. Common statistical methods include mean (or average) and standard deviation.[1]

Discussion

Arrays are an important complex data type used in almost all programming. We continue to concentrate on simple one dimension arrays also called a list. Most programmers develop a series of user-defined specific task functions that can be used with an array for normal processing. These functions are usually passed the array along with the number of elements within the array. Some functions also pass another piece of data needed for that particular functions task.

This module covers the totaling of the members of an integer array member. The Latin name for totaling is summa, sometimes shortened to the word **sum**. In the example below, the sum function totals the array passed to it. Other mathematical functions often associated with statistics such as: average, count, minimum, maximum, standard deviation, etc. are often developed for processing arrays.

Pseudocode

```
Function Main
    Declare Integer Array ages[5]
    Declare Integer total

    Assign ages = [49, 48, 26, 19, 16]

    Assign total = sum(ages)

    Output "Total age is: " & total
End

Function sum (Integer Array array)
    Declare Integer total
    Declare Integer index
```

1. Wikipedia: Statistics

```
    Assign total = 0
    For index = 0 to Size(array) - 1
        Assign total = total + array[index]
    End
Return Integer total
```

Output

```
Total age is: 158
```

Key Terms

sum
Latin for summa or a total.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Searching Arrays

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

Linear search or sequential search is a method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.[1]

Discussion

Finding a specific member of an array means searching the array until the member is found. It's possible that the member does not exist and the programmer must handle that possibility within the logic of his or her algorithm.

"The linear search is a very simple algorithm. Sometimes called a sequential search, it uses a loop to sequentially step through an array, starting with the first element. It compares each element with the value being searched for, and stops when either the value is found or the end of the array is encountered. If the value being searched for is not in the array, the algorithm will search to the end of the array."[2]

Two specific linear searches can be made for the maximum (largest) value in the array or the minimum (smallest) value in the array. Maximum and minimum are also known as max and min. Note that the following max and min functions assume an array size >= 1.

Pseudocode

```
Function Main
    Declare Integer Array ages[5]
    Declare Integer maximum
    Declare Integer minimum

    Assign ages = [49, 48, 26, 19, 16]

    Assign maximum = max(ages)
    Assign minimum = min(ages)

    Output "Maximum age is: " & maximum
```

1. Wikipedia: Linear search
2. Tony Gaddis, Judy Walters, and Godfrey Muganda, Starting Out with C++ Early Objects Sixth Edition (United States of America: Pearson – Addison Wesley, 2008) 559.

```
    Output "Minimum age is: " & minimum
End

Function max (Integer Array array)
    Declare Integer maximum
    Declare Integer index

    Assign maximum = array[0]
    For index = 1 to Size(array) - 1
        If maximum < array[index]
            Assign maximum = array[index]
        End
    End
Return Integer maximum

Function min (Integer Array array)
    Declare Integer minimum
    Declare Integer index

    Assign minimum = array[0]
    For index = 1 to Size(array) - 1
        If minimum > array[index]
            Assign minimum = array[index]
        End
    End
Return Integer minimum
```

Output

```
Maximum age is: 49
Minimum age is: 16
```

Key Terms

linear search
Using a loop to sequentially step through an array.

maximum
Aka max or the largest member of an array.

minimum
Aka min or the smallest member of an array.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++
- Wikiversity: Computer Programming

Sorting Arrays

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

A **sorting** algorithm is an algorithm that puts elements of a list in a certain order. The most frequently used orders are numerical order and lexicographical order.[1] Most current programming languages include built-in or standard library functions for sorting arrays.

Discussion

Sorting is the process through which data are arranged according to their values. The following examples show standard library and/or built-in array sorting methods for different programming languages.

Language	Sort Example
C++	`#include <algorithm>` `sort(array, array + sizeof(array) / sizeof(int));`
C#	`System.Array.Sort(array)`
Java	`import java.util.Arrays;` `Arrays.sort(array);`
JavaScript	`array.sort();`
Python	`array.sort()`
Swift	`array.sort()`

Key Terms

sorting
: Arranging data according to their values.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

1. Wikipedia: Sorting algorithm

Parallel Arrays

DAVE BRAUNSCHWEIG

Overview

A group of **parallel arrays** is a form of implicit data structure that uses multiple arrays to represent a singular array of records. It keeps a separate, homogeneous data array for each field of the record, each having the same number of elements. Then, objects located at the same index in each array are implicitly the fields of a single record.[1]

Discussion

A data structure is a data organization and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data. Data structure options include arrays, linked lists, records, and classes.[2]

Parallel arrays use two or more arrays to represent a collection of data where each corresponding array index is a matching field for a given record. For example, if there are two arrays, one for names and one for ages, the array elements at names[2] and ages[2] would describe the name and age of the third person.

Pseudocode

```
Function Main
    Declare String Array names[5]
    Declare Integer Array ages[5]

    Assign names = ["Lisa", "Michael", "Ashley", "Jacob", "Emily"]
    Assign ages = [49, 48, 26, 19, 16]

    DisplayArrays(names, ages)
End

Function DisplayArrays (String Array names, Integer Array ages)
    Declare Integer index

    For index = 0 to Size(array) - 1
```

1. Wikipedia: Parallel array
2. Wikipedia: Data structure

```
        Output names[index] & " is " & ages[index] & " years old"
    End
End
```

Output

```
Lisa is 49 years old
Michael is 48 years old
Ashley is 26 years old
Jacob is 19 years old
Emily is 16 years old
```

Key Terms

parallel array
An implicit data structure that uses multiple arrays to represent a singular array of records.

References

Multidimensional Arrays

KENNETH LEROY BUSBEE

Overview

The number of indices needed to specify an element is called the dimension or dimensionality of the array. A two-dimensional array, or table, may be stored as a one-dimensional array of one-dimensional arrays (rows of columns) and accessed with double indexing (`array[row][column]` in typical notation).[1]

Discussion

An array is a sequenced collection of elements of the same data type with a single identifier name. As such, the array data type belongs to the "Complex" category or family of data types. Arrays can have multiple axes (more than one axis). Each axis is a **dimension**. Thus a single dimension array is also known as a **list**. A two-dimension array is commonly known as a **table** (a spreadsheet like Excel is a two dimension array). In real life, there are occasions to have data organized into multiple dimensioned arrays. Consider a theater ticket with section, row, and seat (three dimensions).

We refer to the individual values as members (or elements) of the array. Multidimensional arrays use one set of square brackets per dimension or axis of the array. For example, a table which has two dimensions would use two sets of square brackets to define the array variable and two sets of square brackets for the index operators to access the members of the array. Programming languages implement the details of arrays differently. The total number of dimensions allowed in an array is language-specific and also limited by available memory.

Pseudocode

```
Function Main
    Declare String Array game[3][3]

    Assign game = [ ["X", "O", "X"], ["O", "O", "O"], ["X", "O", "X"] ]

    DisplayGame(game)
End

Function DisplayGame (String Array game)
    Declare Integer row
    Declare Integer column
```

1. Wikipedia: Array data type

```
    Output "Tic-Tac-Toe"
    For row = 0 to 2
        For column = 0 to 2
            Output game[row][column]
            If column < 2 Then
                Output " | "
            End
        End
    End
End
```

Output

```
Tic-Tac-Toe
X | O | X
O | O | O
X | O | X
```

Key Terms

array member
An element or value in an array.
dimension
An axis of an array.
index
An operator that allows us to reference a member of an array.
list
A single dimension array.
offset
The method of referencing array members by starting at zero.
table
A two-dimension array.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Fixed and Dynamic Arrays

DAVE BRAUNSCHWEIG

Overview

A **fixed array** is an array for which the size or length is determined when the array is created and/or allocated.[1]

A **dynamic array** is a random access, variable-size list data structure that allows elements to be added or removed. It is supplied with standard libraries in many modern programming languages. Dynamic arrays overcome a limit of static arrays, which have a fixed capacity that needs to be specified at allocation.[2]

Discussion

Static arrays have their size or length determined when the array is created and/or allocated. For this reason, they may also be referred to as fixed-length arrays or fixed arrays. Array values may be specified when the array is defined, or the array size may be defined without specifying array contents. Depending on the programming language, an uninitialized array may contain default values, or it may contain whatever values were left in memory from previous allocation.

Language	Defined Values	Fixed-Length with Undefined or Default Values
C++	`int values[] = {0, 1, 2};`	`int values[3];`
C#	`int[] values = {0, 1, 2};`	`int[] values = = new int[3];`
Java	`int[] values = {0, 1, 2};`	`int[] values = = new int[3];`
JavaScript	`var values = [0, 1, 2];`	`var values = new Array(3);`
Python	`values = [0, 1, 2]`	`values = [None] * 3`
Swift	`var values:[Int] = [0, 1, 2]`	`var values: [Int] = [Int](repeating: 0, count: 3)`

Dynamic arrays allow elements to be added and removed at runtime. Most current programming languages include built-in or standard library functions for creating and managing dynamic arrays.

1. Wikipedia: Array data type
2. Wikipedia: Dynamic array

Language	Class	Add	Remove
C++	`#include <list>` `std::list`	`insert`	`erase`
C#	`System.Collections.Generic.List`	`Add`	`Remove`
Java	`java.util.ArrayList`	`add`	`remove`
JavaScript	`Array`	`push`, `splice`	`pop`, `splice`
Python	`List`	`append`	`remove`
Swift	`Array`	`append`	`remove`

Key Terms

dynamic array
: A data structure consisting of a collection of elements that allows individual elements to be added or removed.

fixed array
: A data structure consisting of a collection of elements for which the size or length is determined when the data structure is defined or allocated.

References

C++ Examples

DAVE BRAUNSCHWEIG

Arrays

```
// This program demonstrates array processing, including:
// display, total, max, min, parallel arrays, sort,
// fixed arrays, dynamic arrays, and multidimensional arrays.

#include <iostream>
#include <list>
#include <algorithm>

using namespace std;

void displayArray(int [], int);
int sum(int [], int);
int max(int [], int);
int min(int [], int);
void displayParallel(string [], int [], int);
void fixedArray();
void dynamicArray();
void displayMultidimensional();

int main() {
    string names[] = {"Lisa", "Michael", "Ashley", "Jacob", "Emily"};
    int ages[] = {49, 48, 26, 19, 16};

    displayArray(ages, sizeof(ages) / sizeof(int));

    int total = sum(ages, sizeof(ages) / sizeof(int));
    int maximum = max(ages, sizeof(ages) / sizeof(int));
    int minimum = min(ages, sizeof(ages) / sizeof(int));

    cout << "total: " << total << endl;
    cout << "maximum: " << maximum << endl;
    cout << "minimum: " << minimum << endl;

    displayParallel(names, ages, sizeof(ages) / sizeof(int));
```

```
    sort(ages, ages + sizeof(ages) / sizeof(int));
    displayArray(ages, sizeof(ages) / sizeof(int));

    fixedArray();
    dynamicArray();
    displayMultidimensional();

    return 0;
}

void displayArray(int arry[], int size) {
    for (int index = 0; index < size; index++) {
        cout << "array[" << index << "] = " << arry[index] << endl;
    }
}

int sum(int arry[], int size) {
    int total = 0;
    for (int index = 0; index < size; index++) {
        total += arry[index];
    }
    return total;
}

int max(int arry[], int size) {
    int maximum = arry[0];
    for (int index = 1; index < size; index++) {
        if (maximum < arry[index]) {
            maximum = arry[index];
        }
    }
    return maximum;
}

int min(int arry[], int size) {
    int minimum = arry[0];
    for (int index = 1; index  arry[index]) {
            minimum = arry[index];
        }
    }
    return minimum;
}
```

```
void displayParallel(string names[], int ages[], int size) {
    for (int index = 0; index < size; index++) {
        cout << names[index] << " is " << ages[index] << " years old" << en
    }
}

void fixedArray() {
    int arry[5];
    srand (time(NULL));
    for (int index = 0; index < 5; index++) {
        int number = rand() % 100;
        arry[index] = number;
    }

    displayArray(arry, 5);
}

void dynamicArray() {
    list arry;
    srand (time(NULL));
    for (int index = 0; index < 5; index++) {
        int number = rand() % 100;
        arry.push_back(number);
    }

    for (list::iterator it = arry.begin(); it != arry.end(); it++) {
        cout << *it << endl;
    }
}

void displayMultidimensional() {
    string game[3][3] = {
        {"X", "O", "X"},
        {"O", "O", "O"},
        {"X", "O", "X"} };

    for (int row = 0; row < 3; row++) {
        for (int column = 0; column < 3; column++) {
            cout << (game[row][column]);
            if (column < 2) {
                cout << " | ";
            }
        }
```

```
        cout << endl;
    }
}
```

Output

```
array[0] = 49
array[1] = 48
array[2] = 26
array[3] = 19
array[4] = 16
total: 158
maximum: 49
minimum: 16
Lisa is 49 years old
Michael is 48 years old
Ashley is 26 years old
Jacob is 19 years old
Emily is 16 years old
array[0] = 16
array[1] = 19
array[2] = 26
array[3] = 48
array[4] = 49
array[0] = 30
array[1] = 14
array[2] = 67
array[3] = 59
array[4] = 96
30
14
67
59
96
X | O | X
O | O | O
X | O | X
```

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

C# Examples

DAVE BRAUNSCHWEIG

Arrays

```
// This program demonstrates array processing, including:
// display, total, max, min, parallel arrays, sort,
// fixed arrays, dynamic arrays, and multidimensional arrays.

using System;
using System.Collections.Generic;

class Arrays {
    public static void Main (string[] args)
    {
        String[] names = {"Lisa", "Michael", "Ashley", "Jacob", "Emily"};
        int[] ages = {49, 48, 26, 19, 16};

        DisplayArray(ages);

        int total = sum(ages);
        int maximum = max(ages);
        int minimum = min(ages);

        Console.WriteLine("total: " + total);
        Console.WriteLine("maximum: " + maximum);
        Console.WriteLine("minimum: " + minimum);

        DisplayParallel(names, ages);

        System.Array.Sort(ages);
        DisplayArray(ages);

        FixedArray();
        DynamicArray();
        DisplayMultidimensional();
    }

    public static void DisplayArray(int[] array)
    {
```

```
        for (int index = 0; index < array.Length; index++)
        {
            Console.WriteLine("array[" + index + "] = " + array[index]);
        }
    }

    public static int sum(int[] array)
    {
        int total = 0;
        for (int index = 0; index < array.Length; index++)
        {
            total += array[index];
        }
        return total;
    }

    public static int max(int[] array)
    {
        int maximum = array[0];
        for (int index = 1; index < array.Length; index++)
        {
            if (maximum < array[index])
            {
                maximum = array[index];
            }
        }
        return maximum;
    }

    public static int min(int[] array)
    {
        int minimum = array[0];
        for (int index = 1; index  array[index])
            {
                minimum = array[index];
            }
        }
        return minimum;
    }

    public static void DisplayParallel(String[] names, int[] ages)
    {
        for (int index = 0; index < names.Length; index++)
```

```
        {
            Console.WriteLine(names[index] + " is " +
                ages[index] + " years old");
        }
    }

    public static void FixedArray()
    {
        int[] array = new int[5];

        Random random = new Random();
        for (int index = 0; index < array.Length; index++)
        {
            int number = random.Next(0, 100);
            array[index] = number;
        }
        DisplayArray(array);
    }

    public static void DynamicArray()
    {
        List array = new List();

        Random random = new Random();
        for (int index = 0; index < 5; index++)
        {
            int number = random.Next(0, 100);
            array.Add(number);
        }
        for (int index = 0; index < array.Count; index++)
        {
            Console.WriteLine("array[" + index + "] = " + array[index]);
        }
    }

    public static void DisplayMultidimensional()
    {
        String[,] game = new String[,]
        {
            {"X", "O", "X"},
            {"O", "O", "O"},
            {"X", "O", "X"}
        };
```

```
        for (int row = 0; row < 3; row++)
        {
            for (int column = 0; column < 3; column++)
            {
                Console.Write(game[row, column]);
                if (column < 2)
                {
                    Console.Write(" | ");
                }
            }
            Console.WriteLine();
        }
    }
}
```

Output

```
array[0] = 49
array[1] = 48
array[2] = 26
array[3] = 19
array[4] = 16
total: 158
maximum: 49
minimum: 16
Lisa is 49 years old
Michael is 48 years old
Ashley is 26 years old
Jacob is 19 years old
Emily is 16 years old
array[0] = 16
array[1] = 19
array[2] = 26
array[3] = 48
array[4] = 49
array[0] = 65
array[1] = 45
array[2] = 78
array[3] = 32
array[4] = 4
array[0] = 24
```

```
array[1] = 62
array[2] = 97
array[3] = 40
array[4] = 82
X | O | X
O | O | O
X | O | X
```

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Java Examples

DAVE BRAUNSCHWEIG

Arrays

```
// This program demonstrates array processing, including:
// display, total, max, min, parallel arrays, sort,
// fixed arrays, dynamic arrays, and multidimensional arrays.

import java.util.*;

class Main {
    public static void main(String[] args) {
        String[] names = {"Lisa", "Michael", "Ashley", "Jacob", "Emily"};
        int[] ages = {49, 48, 26, 19, 16};

        displayArray(ages);

        int total = sum(ages);
        int maximum = max(ages);
        int minimum = min(ages);

        System.out.println("total: " + total);
        System.out.println("maximum: " + maximum);
        System.out.println("minimum: " + minimum);

        displayParallel(names, ages);

        Arrays.sort(ages);
        displayArray(ages);

        fixedArray();
        dynamicArray();
        displayMultidimensional();
    }

    public static void displayArray(int[] array) {
        for (int index = 0; index < array.length; index++) {
            System.out.println("array[" + index + "] = " + array[index]);
        }
```

```
    }

    public static int sum(int[] array) {
        int total = 0;
        for (int index = 0; index < array.length; index++) {
            total += array[index];
        }
        return total;
    }

    public static int max(int[] array) {
        int maximum = array[0];
        for (int index = 1; index < array.length; index++) {
            if (maximum < array[index]) {
                maximum = array[index];
            }
        }
        return maximum;
    }

    public static int min(int[] array) {
        int minimum = array[0];
        for (int index = 1; index  array[index]) {
                minimum = array[index];
            }
        }
        return minimum;
    }

    public static void displayParallel(String[] names, int[] ages) {
        for (int index = 0; index < names.length; index++) {
            System.out.println(names[index] + " is " +
                ages[index] + " years old");
        }
    }

    public static void fixedArray() {
        int[] array = new int[5];

        for (int index = 0; index < array.length; index++) {
            int number = (int) Math.floor(Math.random() * 100);
            array[index] = number;
        }
```

```
        displayArray(array);
    }

    public static void dynamicArray() {
        ArrayList array = new ArrayList();

        for (int index = 0; index < 5; index++) {
            int number = (int) Math.floor(Math.random() * 100);
            array.add(number);
        }
        System.out.println(array);
    }

    public static void displayMultidimensional() {
        String[][] game = {
            {"X", "O", "X"},
            {"O", "O", "O"},
            {"X", "O", "X"} };

        for (int row = 0; row < 3; row++) {
            for (int column = 0; column < 3; column++) {
                System.out.print(game[row][column]);
                if (column < 2) {
                    System.out.print(" | ");
                }
            }
            System.out.println();
        }
    }
}
```

Output

```
array[0] = 49
array[1] = 48
array[2] = 26
array[3] = 19
array[4] = 16
total: 158
maximum: 49
minimum: 16
Lisa is 49 years old
```

```
Michael is 48 years old
Ashley is 26 years old
Jacob is 19 years old
Emily is 16 years old
array[0] = 16
array[1] = 19
array[2] = 26
array[3] = 48
array[4] = 49
array[0] = 28
array[1] = 30
array[2] = 28
array[3] = 75
array[4] = 21
[56, 50, 63, 82, 15]
X | O | X
O | O | O
X | O | X
```

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

JavaScript Examples

DAVE BRAUNSCHWEIG

Arrays

```
// This program demonstrates array processing, including:
// display, total, max, min, parallel arrays, sort,
// fixed arrays, dynamic arrays, and multidimensional arrays.

main()

function main() {
    var names = ['Lisa', 'Michael', 'Ashley', 'Jacob', 'Emily'];
    var ages = [49, 48, 26, 19, 16];

    displayArray(names);
    displayArray(ages);

    var total = sum(ages);
    var maximum = max(ages);
    var minimum = min(ages);

    output('total: ' + total);
    output('maximum: ' + maximum);
    output('minimum: ' + minimum);

    displayParallel(names, ages);

    ages.sort();
    displayArray(ages);

    fixedArray();
    dynamicArray();
    displayMultidimensional();
}

function displayArray(array) {
    for (var index = 0; index < array.length; index++) {
        output('array[' + index + '] = ' + array[index]);
    }
```

```
}

function sum(array) {
    var total = 0;
    for (var index = 0; index < array.length; index++) {
        total += array[index];
    }
    return total;
}

function max(array) {
    var maximum = array[0];
    for (var index = 1; index < array.length; index++) {
        if (maximum < array[index]) {
            maximum = array[index];
        }
    }
    return maximum;
}

function min(array) {
    var minimum = array[0];
    for (var index = 1; index < array.length; index++) {
        if (minimum > array[index]) {
            minimum = array[index];
        }
    }
    return minimum;
}

function displayParallel(names, ages) {
    for (var index = 0; index < names.length; index++) {
        output(names[index] + ' is ' + ages[index] + ' years old');
    }
}

function fixedArray() {
    var array = new Array(5);

    for (var index = 0; index < array.length; index++) {
        var number = Math.floor(Math.random() * 100);
        array[index] = number;
    }
```

```
    displayArray(array);
}

function dynamicArray() {
    var array = [];

    for (var index = 0; index < 5; index++) {
        var number = Math.floor(Math.random() * 100);
        array.push(number);
    }
    displayArray(array);
}

function displayMultidimensional() {
    var game = [
        ['X', 'O', 'X'],
        ['O', 'O', 'O'],
        ['X', 'O', 'X'] ];

    for (var row = 0; row < 3; row++) {
        var line = '';
        for (var column = 0; column < 3; column++) {
            line += game[row][column];
            if (column < 2) {
                line += ' | ';
            }
        }
        output(line);
    }
}

function output(text) {
  if (typeof document === 'object') {
    document.write(text);
  }
  else if (typeof console === 'object') {
    console.log(text);
  }
  else {
    print(text);
  }
}
```

Output

```
array[0] = Lisa
array[1] = Michael
array[2] = Ashley
array[3] = Jacob
array[4] = Emily
array[0] = 49
array[1] = 48
array[2] = 26
array[3] = 19
array[4] = 16
total: 158
maximum: 49
minimum: 16
Lisa is 49 years old
Michael is 48 years old
Ashley is 26 years old
Jacob is 19 years old
Emily is 16 years old
array[0] = 16
array[1] = 19
array[2] = 26
array[3] = 48
array[4] = 49
array[0] = 55
array[1] = 4
array[2] = 46
array[3] = 88
array[4] = 49
array[0] = 28
array[1] = 95
array[2] = 13
array[3] = 60
array[4] = 60
X | O | X
O | O | O
X | O | X
```

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Python Examples

DAVE BRAUNSCHWEIG

Arrays

```
# This program demonstrates array processing, including:
# display, total, max, min, parallel arrays, sort,
# fixed arrays, dynamic arrays, and multidimensional arrays.


def display_array(array):
    for index in range(len(array)):
        print('array[' + str(index) + '] = ' +
            str(array[index]))


def sum(array):
    total = 0
    for index in range(len(array)):
        total += array[index]
    return total


def max(array):
    maximum = array[0]
    for index in range(1, len(array)):
        if maximum  array[index]:
            minimum = array[index]
    return minimum


def display_parallel(names, ages):
    for index in range(len(names)):
        print(names[index] + ' is ' +
            str(ages[index]) + ' years old')


def fixed_array():
    import random
```

```
    array = [None] * 5
    for index in range(len(array)):
        array[index] = random.randint(0, 100)
    display_array(array)


def dynamic_array():
    import random

    array = []
    for index in range(5):
        array.append(random.randint(0, 100))
    display_array(array)


def display_multidimensional():
    game = [
        ['X', 'O', 'X'],
        ['O', 'O', 'O'],
        ['X', 'O', 'X'] ]

    for row in range (0, 3):
        for column in range(0, 3):
            print(game[row][column], end='')
            if column < 2:
                print(' | ', end='')
        print()


def main():
    names = ['Lisa', 'Michael', 'Ashley', 'Jacob', 'Emily']
    ages = [49, 48, 26, 19, 16]

    display_array(names)
    display_array(ages)

    total = sum(ages)
    maximum = max(ages)
    minimum = min(ages)

    print('total: ' + str(total))
    print('maximum: ' + str(maximum))
    print('minimum: ' + str(minimum))
```

```
    display_parallel(names, ages)

    ages.sort()
    display_array(ages)

    fixed_array()
    dynamic_array()
    display_multidimensional()


main()
```

Output

```
array[0] = Lisa
array[1] = Michael
array[2] = Ashley
array[3] = Jacob
array[4] = Emily
array[0] = 49
array[0] = Lisa
array[1] = Michael
array[2] = Ashley
array[3] = Jacob
array[4] = Emily
array[0] = 49
array[1] = 48
array[2] = 26
array[3] = 19
array[4] = 16
total: 158
maximum: 49
minimum: 16
Lisa is 49 years old
Michael is 48 years old
Ashley is 26 years old
Jacob is 19 years old
Emily is 16 years old
array[0] = 16
array[1] = 19
array[2] = 26
```

```
array[3] = 48
array[4] = 49
array[0] = 18
array[1] = 14
array[2] = 59
array[3] = 99
array[4] = 61
array[0] = 85
array[1] = 4
array[2] = 35
array[3] = 45
array[4] = 93
X | O | X
O | O | O
X | O | X
```

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Swift Examples

DAVE BRAUNSCHWEIG

Arrays

```
// This program demonstrates array processing, including:
// display, total, max, min, parallel arrays, sort,
// fixed arrays, dynamic arrays, and multidimensional arrays.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://developer.apple.com/library/content/documentation/Swift/Conc

import Foundation

func displayArray(array: [Int]) {
    for index in 0...array.count - 1 {
        print("array[" + String(index) + "] = " + String(array[index]))
    }
}

func displayParallel(names:[String], ages:[Int]) {
    for index in 0...names.count - 1 {
        print(names[index] + " is " + String(ages[index]))
    }
}

func fixedArray() {
    var array: [Int] = [Int](repeating: 0, count: 5)

    srand(UInt32(time(nil)))
    for index in 0...4 {
        array[index] = random() % 100
    }
    print(array)
}

func dynamicArray() {
    var array: [Int] = []
```

```
    srand(UInt32(time(nil)))
    for _ in 0...4 {
        array.append(random() % 100)
    }
    print(array)
}

func displayMultidimensional() {
    var game: [[String]]

    game = [
        ["X", "O", "X"],
        ["O", "X", "O"],
        ["X", "O", "X"]
    ]

    for row in 0...2 {
        for column in 0...2 {
            print(game[row][column], terminator:"")
            if column < 2 {
                print(" | ", terminator:"")
            }
        }
        print()
    }
}

func main() {
    var names: [String]
    var ages: [Int]
    var total: Int
    var maximum: Int
    var minimum: Int

    names = ["Lisa", "Michael", "Ashley", "Jacob", "Emily"]
    ages = [49, 48, 26, 19, 16]

    displayArray(array:ages)
    total = ages.reduce(0, +)
    maximum = ages.max()!
    minimum = ages.min()!

    print("total:", total)
```

```
    print("maximum:", maximum)
    print("minimum:", minimum)

    displayParallel(names:names, ages:ages)

    ages.sort()
    displayArray(array:ages)

    fixedArray()
    dynamicArray()
    displayMultidimensional()
}

main()
```

Output

```
array[0] = 49
array[1] = 48
array[2] = 26
array[3] = 19
array[4] = 16
total: 158
maximum: 49
minimum: 16
Lisa is 49
Michael is 48
Ashley is 26
Jacob is 19
Emily is 16
array[0] = 16
array[1] = 19
array[2] = 26
array[3] = 48
array[4] = 49
[89, 41, 22, 56, 60]
[89, 41, 22, 56, 60]
X | O | X
O | X | O
X | O | X
```

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++
- Wikiversity: Computer Programming

Practice: Arrays

KENNETH LEROY BUSBEE

Review Questions

True / False

1. The array data type is one of the standard data types in C++.
2. Arrays can have more than one dimension.
3. For loops are often used to display the members of an array.
4. When defining an array, it is preferable to specify how many members are in the array.
5. Arrays are rarely used to represent data.
6. Linear searches require complex algorithms.
7. Functions are often created for searching for the max and min values in an array.
8. The bubble sort is an easy way to arrange data an array.
9. There is only one method of bubble sorting.
10. Sorting an array is frequently done.

Answers:

1. false
2. true
3. true
4. false
5. false
6. false
7. true
8. true
9. false
10. true

Short Answer

1. Briefly explain what an array is and list the two common operators used with arrays.
2. Give a short explanation of bubble sorting.

Activities

Complete the following activities using pseudocode, a flowcharting tool, or your selected programming language. Use separate functions for input, each type of processing, and output. Avoid global variables by passing parameters and returning results. Create test data to validate the accuracy of each program. Add comments at the top of the program and include references to any resources used.

Defined-Value Arrays

1. Review MathsIsFun: Leap Years. Create a program that asks the user for a year, and then calculate whether or not the given year is a leap year. Build an array where each entry is the number of days in the corresponding month (January = 31, February = 28 or 29 depending on year, March = 31, etc.). Build a parallel string array with the names of each month. Then ask the user to enter a month number, and look up the corresponding month name and number of days and display the information. Continue accepting input and displaying results until the user enters a number less than 1 or greater than 12.
2. Review Wikipedia: Zeller's congruence. Create a program that asks the user for their birthday (year, month, and day) and then calculates and displays the day of the week on which they were born. Use an array lookup to convert the numeric day of the week to the correct string representation (Monday, Tuesday, Wednesday, etc.).

Fixed-Length Arrays

1. Review MathsIsFun: Definition of Average. Create a program that asks the user to enter grade scores. Start by asking the user how many scores they would like to enter. Then use a loop to request each score and add it to a static (fixed-size) array. After the scores are entered, calculate and display the high, low, and average for the entered scores.
2. Review Wikipedia: Monty Hall problem. Create a program that uses an array to simulate the three doors. Use 0 (zero) to indicate goats and 1 (one) to indicate the car. Clear each "door" and then use a random number function to put the number 1 in one of the array elements. Then use the random number function to randomly select one of the three elements. Run the simulation in a loop 100 times to confirm a 1/3 chance of winning. Then run the simulation again, this time switching the selection after a 0 (goat) is removed from the remaining choices. Run the simulation in a loop 100 times to confirm a 2/3 chance of winning by switching.
3. If your programming language supports it, use a built-in sort function to sort the grade scores from the activity above and display the array in order from highest score to lowest score.

Dynamic Arrays / Lists

1. If your programming language supports it, update the grade scores program above to replace the static array with a dynamic array, and extend the array as each item is added to the array. Continue accepting scores until the user enters a negative value.
2. Review Khan Academy: A guessing game. Write a program that allows the user to think of a number between 0 and 100, inclusive. Then have the program try to guess their number. Start at the midpoint (50) and ask the user if their number is (h)igher, (l)ower, or (e)qual to the guess. If they indicate lower, guess the new midpoint (25). If they indicate higher, guess the new midpoint (75). Record each guess in an an array and continue efficiently guessing higher or lower until they indicate equal, then display the list of guesses required to guess their number and end the program.
3. If your programming language supports it, use a built-in sort function to sort the grade scores from the activity above and display the array in order from highest score to lowest score.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++
- Wikiversity: Computer Programming

PART VII

STRINGS AND FILES

Overview

This chapter introduces string and file processing.

Chapter Outline

Learning Objectives

1. Understand key terms and definitions.
2. Given example pseudocode, flowcharts, and source code, create a program that processes strings to solve a given problem.
3. Given example pseudocode, flowcharts, and source code, create a program that processes a text file to solve a given problem.

Strings

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

A **string** is traditionally a sequence of characters, either as a literal constant or as some kind of variable. The latter may allow its elements to be mutated and the length changed, or it may be fixed (after creation). A string is generally considered a data type and is often implemented as an array data structure of bytes (or words) that stores a sequence of elements, typically characters, using some character encoding.[1]

Discussion

Recall from String Data Type earlier in the book that, depending on programming language and precise data type used, a variable declared to be a string may either cause storage in memory to be statically allocated for a predetermined maximum length or employ dynamic allocation to allow it to hold a variable number of elements. When a string appears literally in source code, it is known as a string literal or an anonymous string.[2]

Most data is more complex than just one character, integer, etc. Programming languages develop other methods to represent and store data that are more complex. A complex data type of array is the first most students encounter. An array is a sequenced collection of elements of the same data type with a single identifier name. This definition perfectly describes our string data type concept. The simplest array is called a one-dimensional array; also know as a list because we usually list the members or elements vertically. However, strings are viewed as a one-dimensional array that visualize as listed horizontally. Strings are an array of character data.

In the "C" programming language all strings were handled as an array of characters that end in an ASCII null character (the value 0 or the first character in the ASCII character code set). This approach required programmers to manually process string length and manage string storage. Buffer overflows were common. A **buffer overflow**, or buffer overrun, is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations.[3]

Most current programming languages implement strings as a data type or class where strings are stored as a length controlled array. String length and storage are handled by the compiler or interpreter, reducing program errors.

1. Wikipedia: String (computer science)
2. Wikipedia: String (computer science)
3. Wikipedia: Buffer overflow

Language	Reserved Word
C++	`string`
C#	`String`
Java	`String`
JavaScript	`String`
Python	`str()`
Swift	`String`

Key Terms

array
A sequenced collection of elements of the same data type with a single identifier name.

buffer overflow
An anomaly where a program overruns a memory storage location and overwrites adjacent memory locations.

concatenation
Combining two strings into one string.

string class
A complex data item that uses object oriented programming.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

String Functions

DAVE BRAUNSCHWEIG

Overview

String functions are used in computer programming languages to manipulate a string or query information about a string.[1]

Discussion

Most current programming languages include built-in or library functions to process strings. Common examples include case conversion, comparison, concatenation, find, join, length, reverse, split, substring, and trim.

Function	C++	C#	Java
case	`tolower()`, `toupper()`, etc.	`ToLower()`, `ToUpper()`, etc.	`toLowerCase()`, `toUpperCase()`, etc.
comparison	`<`, `>`, `==`, etc.	`<`, `>`, `==`, etc.	`<`, `>`, `==`, etc.
concatenation	`+`, `+=`	`+`, `+=`	`+`, `+=`
find	`find()`	`IndexOf()`	`indexOf()`
join	N/A	`Join()`	`join()`
length	`length()`	`Length`	`length()`
replace	`replace()`	`Replace()`	`replace()`
reverse	`reverse()`	`Reverse()`	N/A
split	`strtok()`	`Split()`	`split()`
substring	`substr()`	`Substring()`	`substring()`
trim	N/A	`Trim()`	`trim()`

1. Wikipedia: Comparison of programming languages (string functions)

Function	JavaScript	Python	Swift
case	`toLowerCase()`, `toUpperCase()`, etc.	`lower()`, `upper()`, etc.	`lowercased()`, `uppercased()`
comparison	`<`, `>`, `==`, etc.	`<`, `>`, `==`, etc.	`<`, `>`, `==`, etc.
concatenation	`+`, `+=`	`+`, `+=`	`+`, `+=`
find	`indexOf()`	`find()`	`firstIndex()`
join	`join()`	`join()`	`joined()`
length	`length`	`len()`	`count`
replace	`replace()`	`replace()`	`replacingOccurrences()`
reverse	N/A	`string[::-1]`	`reversed()`
split	`split()`	`split()`	`split()`
substring	`substring()`	`string[start:end]`	`string[start...end]`
trim	`trim()`	`strip()`	`trimmingCharacters()`

Key Terms

concatenate
: Join character strings end-to-end.[2]

trim
: Remove leading and trailing spaces from a string.[3]

References

2. Wikipedia: Concatenation
3. Wikipedia: Trimming (computer programming)

String Formatting

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

String formatting uses a process of string interpolation (variable substitution) to evaluate a string literal containing one or more placeholders, yielding a result in which the placeholders are replaced with their corresponding values.[1]

Discussion

Most current programming languages provide one or more string formatting functions that use a template string with placeholders and optional alignment, width, and precision indicators to generate formatted output.

Language	Function	Examples
C++	`snprintf()`	`snprintf(str, sizeof(str), "Hello %s!", name);` `snprintf(str, sizeof(str), "$%.2f", value);`
C#	`Format()`	`String.Format("Hello {0}!", name);` `String.Format("{0:$0.00}", value);`
Java	`format()`	`String.format("Hello %s!", name);` `String.format("$%.2f", value);`
JavaScript	template literal	`` `Hello ${name}`; `` `` `$${value.toFixed(2)}`; ``
Python	`format()`	`"Hello {}!".format(name)` `"${:.2f}".format(value)`
Swift	interpolation `String()`	`"Hello \(name)!"` `String(format:"%.2f", value)`

String interpolation, like string concatenation, may lead to security problems. If user input data is improperly escaped or filtered, the system may be exposed to code injection.[2]

1. Wikipedia: String interpolation
2. Wikipedia: String interpolation

Key Terms

code injection
: The exploitation of a computer bug that is caused by processing invalid data.[3]

formatting
: Modifying the way the output is displayed.

string interpolation
: Evaluating a string literal containing one or more placeholders, yielding a result in which the placeholders are replaced with their corresponding values.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

3. Wikipedia: Code injection

File Input and Output

KENNETH LEROY BUSBEE

Overview

A computer file is a computer resource for recording data discretely in a computer storage device. Just as words can be written to paper, so can information be written to a computer file.

There are different types of computer files, designed for different purposes. A file may be designed to store a picture, a written message, a video, a computer program, or a wide variety of other kinds of data. Some types of files can store several types of information at once.

By using computer programs, a person can open, read, change, and close a computer file. Computer files may be reopened, modified, and copied an arbitrary number of times.[1]

Discussion

In computer programming, standard **streams** are pre-connected input and output communication channels between a computer program and its environment when it begins execution. The three input/output (I/O) connections are called standard input (**stdin** – keyboard), standard output (**stdout** – originally a printer) and standard error (**stderr** – monitor). Streams may be redirected to other devices and/or files. In current environments, stdout is usually redirected to the monitor.[2]

Computer files are stored on secondary storage devices and used to maintain program data over time. Most programming languages have built-in functions or libraries to support processing files as text streams. We need to understand how to open, read, write and close text files. The following File Input/Output terms are explained:

Text File – A file consisting of characters from the ASCII character code set. Text files (also known as ASCII text files) contain character data. When we create a text file we usually think of it consisting of a series of lines. On each line are several characters (including spaces, punctuation, etc.) and we generally end the line with a return (a character within the ASCII character code set). The return is also known as the new line character. You are most likely already familiar with the escape code of \n which is used within many programming languages to indicate a return character when used within a literal string.

A typical text file consisting of lines can be created by text editors (Notepad) or word processing programs (Microsoft Word). When using a word processor you must usually specify the output file as text (.txt) when saving it. Most source code files are ASCII text files with a unique file extension; such as C++ using .cpp, C# using .cs, Python using .py, etc. Thus, most compiler/Integrated Development Environment software packages can be used to create ASCII text files.

Filename – The name and its extension. Most operating systems have restrictions on which characters can be used in filenames. Example Lab_05.txt

Because some operating systems do not allow spaces, we suggest that you use the underscore where needed for spacing in a filename.

Path (Filespec) – The location of a file along with its filename. Filespec is short for file specification. Most operating systems have a set of rules on how to specify the drive and directory

1. Wikipedia: Computer file
2. Wikipedia: Standard streams

(or path through several directory levels) along with the filename. Example: C:\myfiles\cosc_1436\Lab_05.txt

Because some operating systems do not allow spaces, we suggest that you use the **underscore** where needed when creating folders or sub-directories.

Open – Your program requesting the operating system to let it have access to an existing file or to open a new file. In most current programming languages, a file data type exists and is used for file processing. A file variable will be used to store the device token that the operating system assigns to the file being opened. An open function or method is used to retrieve the device token, and typically requires at least two parameters: the path and the mode (read, write, append, or a combination thereof). Corresponding pseudocode would be:

```
Declare File datafile

datafile = open(filespec, mode)
```

The open function provides a return value of a **device token** from the operating system and it is stored in the variable named data.

It is considered good programming practice to determine if the file was opened properly. The reason the operating system usually can't open a file is because the filespec is wrong (misspelled or not typed case consistent in some operating systems) or the file is not stored in the location specified. Accessing files stored on a network or the Internet may fail due to a network error.

Verifying that a file was opened properly is processed with a condition control structure. That structure may be either be an if-then-else statement or a try-catch / try-except error handler, depending on the programming language used.

Read – Moving data from a device that has been opened into a memory location defined in your program. For example:

```
text = read(datafile)
```

or

```
text = datafile.read()
```

Write – Moving data from a memory location defined in your program to a device that has been opened. For example:

```
write(datafile, text)
```

or

```
datafile.write(text)
```

Close – Your program requesting the operating system to release a file that was previously opened. There are two reasons to close a file. First, it releases the file and frees up the associated operation system resources. Second, if closing a file that was opened for output; it will clear the out the operating system's buffer and ensure that all of the data is physically stored in the output file. For example:

```
close(datafile)
```

or

```
datafile.close()
```

Using / With – A wrapper around a processing block that will automatically close opened resources, available in some programming languages. For example:

```
// C#
using (datafile = open(filespec, mode))
{
```

```
    //...
}
```

or

```
# Python3
with open(filespec, mode) as datafile:
    # ...
```

Key Terms

close
: Your program requesting the operating system to release a file that was previously opened.

device token
: A key value provided by the operating system to associate a device to your program.

filename
: The name and its extension.

filespec
: The location of a file along with its filename.

open
: Your program requesting the operating system to let it have access to an existing file or to open a new file.

read
: Moving data from a device that has been opened into a memory location defined in your program.

stream
: A sequence of data elements made available over time.[3]

stdin
: Standard input stream, typically the keyboard. [4]

stderr
: Standard output error stream, typically the monitor.[5]

stdout
: Standard output stream, originally a printer, but now typically the monitor.[6]

text file
: A file consisting of characters from the ASCII character code set.

using / with
: A wrapper around a processing block that will automatically close opened resources.

write
: Moving data from a memory location defined in your program to a device that has been opened.

3. Wikipedia: Stream (computing)
4. Wikipedia: Standard streams
5. Wikipedia: Standard streams
6. Wikipedia: Standard streams

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Loading an Array from a Text File

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

Loading an array from a text file requires several steps, including: opening the file, reading the records, parsing (splitting) the records into fields, adding the fields to an array, and closing the file. The file may be read all at once and then parsed, or processed line by line. The array must either be at least as large as the number of records in the file, or must be generated dynamically.

Discussion

Loading an array from a file presents an interesting dilemma. The problem resolves around how many elements you should plan for in the array. Let's say 100, but what if the file has fewer or more than 100 values. How can the program handle it correctly?
Either:

1. Read the file and count the number of records.
2. Create a static array of that size.
3. Read the file again and add each record to the array.

Or:

1. Read the file and dynamically add the records to the array.

Processing Records

There are two options for file processing:

1. Read the entire file into memory, split the records and then process each record.
2. Read the file line by line and process one record at a time.

Which of these approaches is better will depend on the size of the file and the types of file and string processing supported by your programming language. Reading the entire file at once may be faster for small files. Very large files must be processed line by line.

Processing Fields

Processing fields requires splitting records based on the given file format. For example, a comma-separated-values file might be formatted as:

```
Celsius,Fahrenheit
```

```
0.0,32.0
1.0,33.8
2.0,35.6
```

The first line contains field names separated by commas. Following lines contain a value for each of the fields, separated by commas. Note that all text file input is strings. Each line must be split on the field separator (comma), and then numeric fields must be converted to integer or floating point values for processing.

Pseudocode

Static Array Processing

```
Open file
Read header
While Not End-Of-File
    Read line
    Increment record count
Close file

Declare array with length based on record count
Read Header
While Not End-Of-File
    Read line
    Split line into field(s)
    Convert numeric values to numeric data types
    Add field(s) to array or parallel arrays
Close file
```

Dynamic Array Processing

```
Declare empty array
Open file
Read Header
While Not End-Of-File
    Read line
    Split line into field(s)
    Convert numeric values to numeric data types
    Add field(s) to array or parallel arrays
Close file
```

Key Terms

dynamic memory
Aka stack created memory associated with local scope.
static memory
Aka data area memory associated with global scope.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

C++ Examples

DAVE BRAUNSCHWEIG

Strings

```
#include <algorithm>
#include <iostream>
#include <string>

using namespace std;

string toLower(string);
string toUpper(string);

int main() {
    string str = "Hello";

    cout << "string: " << str << endl;
    cout << "tolower: " << toLower(str) << endl;
    cout << "toupper: " << toUpper(str) << endl;
    cout << "string.find('e'): " << str.find('e') << endl;
    cout << "string.length(): " << str.length() << endl;
    cout << "string.replace(0, 1, \"j\"): " << str.replace(0, 1, "j") << en
    cout << "string.substr(2, 2): " << str.substr(2, 2) << endl;

    string name = "Bob";
    double value = 123.456;
    cout << name << " earned $" << fixed << setprecision (2) << value << en

}

string toLower(string str) {
    transform(str.begin(), str.end(), str.begin(), ::tolower);

    return str;
}

string toUpper(string str) {
    transform(str.begin(), str.end(), str.begin(), ::toupper);
```

```
    return str;
}
```

Output

```
string: Hello
tolower: hello
toupper: HELLO
string.find('e'): 1
string.length(): 5
string.replace(0, 1, "j"): jello
string.substr(2, 2): ll
Bob earned $123.46
```

Files

```
// This program creates a file, adds data to the file, displays the file,
// appends more data to the file, displays the file, and then deletes the f
// It will not run if the file already exists.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/C%2B%2B_Programming

#include <fstream>
#include <iomanip>
#include <iostream>
#include <string>

using namespace std;

double calculateFahrenheit(double celsius);
void createFile(string);
void readFile(string);
void appendFile(string);
void deleteFile(string);
int fileExists(string);

int main() {
    string FILENAME = "~file.txt";
```

```
    if(fileExists(FILENAME)) {
        cout << "File already exists." << endl;
    } else {
        createFile(FILENAME);
        readFile(FILENAME);
        appendFile(FILENAME);
        readFile(FILENAME);
        deleteFile(FILENAME);
    }
}

double calculateFahrenheit(double celsius) {
    double fahrenheit;

    fahrenheit = celsius * 9 / 5 + 32;
    return fahrenheit;
}

void createFile(string filename) {
    fstream file;
    float celsius;
    float fahrenheit;

    file.open(filename, fstream::out);
    if (file.is_open()) {
        file << "Celsius,Fahrenheit\n";
        for(celsius = 0; celsius <= 50; celsius++) {
            fahrenheit = calculateFahrenheit(celsius);
            file << fixed << setprecision (1) << celsius << "," << fahrenhe
        }
        file.close();
    } else {
        cout << "Error creating " << filename << endl;
    }
}

void readFile(string filename)
{
    fstream file;
    string line;

    file.open(filename, fstream::in);
```

```
    if (file.is_open()) {
        while (getline(file, line))
        {
            cout << line << endl;
        }
        file.close();
        cout << endl;
    } else {
        cout << "Error reading " << filename << endl;
    }
}

void appendFile(string filename)
{
    fstream file;
    float celsius;
    float fahrenheit;

    file.open(filename, fstream::out | fstream::app);
    if (file.is_open()) {
        for(celsius = 51; celsius <= 100; celsius++) {
            fahrenheit = calculateFahrenheit(celsius);
            file << fixed << setprecision (1) << celsius << "," << fahrenhe
        }
        file.close();
    } else {
        cout << "Error appending to " << filename << endl;
    }
}

void deleteFile(string filename)
{
    remove(filename.c_str());
}

int fileExists(string filename)
{
    FILE *file;

    file = fopen (filename.c_str(), "r");
    if (file != NULL)
    {
        fclose (file);
```

```
    }
    return (file != NULL);
}
```

Output

```
Celsius,Fahrenheit
0.0,32.0
1.0,33.8
2.0,35.6
...
98.0,208.4
99.0,210.2
100.0,212.0
```

References

- Wikiversity: Computer Programming

C# Examples

DAVE BRAUNSCHWEIG

Strings

```
// This program demonstrates string functions.

using System;

class Strings
{
    public static void Main (string[] args)
    {
        String str = "Hello";

        Console.WriteLine("string: " + str);
        Console.WriteLine("string.ToLower(): " + str.ToLower());
        Console.WriteLine("string.ToUpper(): " + str.ToUpper());
        Console.WriteLine("string.IndexOf('e'): " + str.IndexOf('e'));
        Console.WriteLine("string.Length: " + str.Length);
        Console.WriteLine("string.Replace('H', 'j'): " + str.Replace('H', '
        Console.WriteLine("string(Substring(2, 2)): " + str.Substring(2, 2)
        Console.WriteLine("string.Trim(): " + str.Trim());

        String name = "Bob";
        double value = 123.456;
        Console.WriteLine(String.Format("{0} earned {1:$0.00}", name, value
    }
}
```

Output

```
string: Hello
string.ToLower(): hello
string.ToUpper(): HELLO
string.IndexOf('e'): 1
string.Length: 5
string.Replace('H', 'j'): jello
string(Substring(2, 2)): ll
```

```
string.Trim(): Hello
Bob earned $123.46
```

Files

```
// This program creates a file, adds data to the file, displays the file,
// appends more data to the file, displays the file, and then deletes the f
// It will not run if the file already exists.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/C_Sharp_Programming

using System;

public class Files
{
    public static void Main(String[] args)
    {
        string FILENAME = "~file.txt";

        if(System.IO.File.Exists(FILENAME))
        {
            System.Console.WriteLine("File already exists.\n");
        }
        else
        {
            CreateFile(FILENAME);
            ReadFile(FILENAME);
            AppendFile(FILENAME);
            ReadFile(FILENAME);
            DeleteFile(FILENAME);
        }
    }

    private static double CalculateFahrenheit(double celsius)
    {
        double fahrenheit;

        fahrenheit = celsius * 9 / 5 + 32;
        return fahrenheit;
```

```
    }

    private static void CreateFile(string filename)
    {
        System.IO.StreamWriter file;
        double celsius;
        double fahrenheit;

        try
        {
            using(file = System.IO.File.CreateText(filename))
            {
                file.WriteLine("Celsius,Fahrenheit");
                for(celsius = 0; celsius <= 50; celsius++)
                {
                    fahrenheit = CalculateFahrenheit(celsius);
                    file.WriteLine(String.Format("{0:F1},{1:F1}", celsius,
                }
            }
        }
        catch(Exception exception)
        {
            Console.WriteLine("Error creating " + filename);
            Console.WriteLine(exception.Message);
        }
    }

    private static void ReadFile(string filename)
    {
        System.IO.StreamReader file;
        string line;

        try
        {
            using (file = System.IO.File.OpenText(filename))
            {
                while (true)
                {
                    line = file.ReadLine();
                    if (line == null)
                    {
                        break;
                    }
```

```
                Console.WriteLine(line);
            }
        }
        Console.WriteLine("");
    }
    catch(Exception exception)
    {
        Console.WriteLine("Error reading " + filename);
        Console.WriteLine(exception.Message);
    }
}

private static void AppendFile(string filename)
{
    System.IO.StreamWriter file;
    double celsius;
    double fahrenheit;

    try
    {
        using (file = System.IO.File.AppendText(filename))
        {
            for(celsius = 51; celsius <= 100; celsius++)
            {
                fahrenheit = CalculateFahrenheit(celsius);
                file.WriteLine(String.Format("{0:F1},{1:F1}", celsius,
            }
        }
    }
    catch(Exception exception)
    {
        Console.WriteLine("Error appending to " + filename);
        Console.WriteLine(exception.Message);
    }
}

private static void DeleteFile(string filename)
{
    try
    {
        System.IO.File.Delete(filename);
    }
    catch(Exception exception)
```

```
        {
            Console.WriteLine("Error deleting " + filename);
            Console.WriteLine(exception.Message);
        }
    }
}
```

Output

```
Celsius,Fahrenheit
0.0,32.0
1.0,33.8
2.0,35.6
...
98.0,208.4
99.0,210.2
100.0,212.0
```

References

- Wikiversity: Computer Programming

Java Examples

DAVE BRAUNSCHWEIG

Strings

```
// This program demonstrates string functions.

class Main {
    public static void main(String[] args) {
        String str = "Hello";

        System.out.println("string: " + str);
        System.out.println("string.toLowerCase(): " + str.toLowerCase());
        System.out.println("string.toUpperCase(): " + str.toUpperCase());
        System.out.println("string.indexOf('e'): " + str.indexOf('e'));
        System.out.println("string.length(): " + str.length());
        System.out.println("string.replace('H', 'j'): " + str.replace('H',
        System.out.println("string(substring(2,4): " + str.substring(2, 4))
        System.out.println("string.trim(): " + str.trim());

        String name = "Bob";
        double value = 123.456;
        System.out.println(String.format("%s earned $%.2f", name, value));
    }
}
```

Output

```
string: Hello
string..toLowerCase(): hello
string.toUpperCase(): HELLO
string.indexOf('e'): 1
string.length(): 5
string.replace('H', 'j'): jello
string(substring(2,4): ll
string.trim(): Hello
Bob earned $123.46
```

Files

```
// This program creates a file, adds data to the file, displays the file,
// appends more data to the file, displays the file, and then deletes the f
// It will not run if the file already exists.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/Java_Programming

import java.util.*;

class Main {
    public static void main(String[] args) {
        String FILENAME = "~file.txt";

        if(fileExists(FILENAME)) {
            System.out.println("File already exists.\n");
        } else {
            createFile(FILENAME);
            readFile(FILENAME);
            appendFile(FILENAME);
            readFile(FILENAME);
            deleteFile(FILENAME);
        }
    }

    private static double calculateFahrenheit(double celsius) {
        double fahrenheit;

        fahrenheit = celsius * 9 / 5 + 32;
        return fahrenheit;
    }

    private static void createFile(String filename) {
        try {
            java.io.File file = new java.io.File(filename);
            java.io.BufferedWriter writer =
                new java.io.BufferedWriter(new java.io.FileWriter(file));
            double celsius;
            double fahrenheit;

            writer.write("Celsius,Fahrenheit\n");
```

```
            for(celsius = 0; celsius <= 50; celsius++) {
                fahrenheit = calculateFahrenheit(celsius);
                writer.write(celsius + "," + fahrenheit + "\n");
            }
            writer.close();
        } catch(Exception exception) {
            System.out.println("Error creating " + filename);
            exception.printStackTrace();
        }
    }

    private static void readFile(String filename) {
        try {
            java.io.File file = new java.io.File(filename);
            java.io.BufferedReader reader =
                new java.io.BufferedReader(new java.io.FileReader(file));
            String line;

            while(true) {
                line = reader.readLine();
                if (line == null) {
                    break;
                }
                System.out.println(line);
            }
            reader.close();
            System.out.println("");
        } catch(Exception exception) {
            System.out.println("Error reading " + filename);
            exception.printStackTrace();
        }
    }

    private static void appendFile(String filename)
    {
        try {
            java.io.File file = new java.io.File(filename);
            java.io.BufferedWriter writer =
                new java.io.BufferedWriter(new java.io.FileWriter(file, tru
            double celsius;
            double fahrenheit;

            for(celsius = 51; celsius <= 100; celsius++) {
```

```
                fahrenheit = calculateFahrenheit(celsius);
                writer.write(celsius + "," + fahrenheit + "\n");
            }
            writer.close();
        } catch(Exception exception) {
            System.out.println("Error appending to " + filename);
            exception.printStackTrace();
        }
    }

    private static void deleteFile(String filename) {
        java.io.File file;

        try {
            file = new java.io.File(filename);
            file.delete();
        } catch(Exception exception) {
            System.out.println("Error deleting " + filename);
            exception.printStackTrace();
        }

    }

    private static boolean fileExists(String filename) {
        java.io.File file;

        file = new java.io.File(filename);
        return file.exists();
    }
}
```

Output

```
Celsius,Fahrenheit
0.0,32.0
1.0,33.8
2.0,35.6
...
98.0,208.4
99.0,210.2
100.0,212.0
```

References

- Wikiversity: Computer Programming

JavaScript Examples

DAVE BRAUNSCHWEIG

Strings

```
// This program demonstrates string functions.

main();

function main()
{
    var str = "Hello";

    output("string: " + str);
    output("string.toLowerCase(): " + str.toLowerCase());
    output("string.toUpperCase(): " + str.toUpperCase());
    output("string.indexOf('e'): " + str.indexOf('e'));
    output("string.length: " + str.length);
    output("string.replace('H', 'j'): " + str.replace('H', 'j'));
    output("string(substring(2,4): " + str.substring(2, 4));
    output("string.trim(): " + str.trim());

    var name = "Bob";
    var value = 123.456;
    output(`string.format(): ${name} earned $${value.toFixed(2)}`);
}

function output(text) {
  if (typeof document === 'object') {
    document.write(text);
  }
  else if (typeof console === 'object') {
    console.log(text);
  }
  else {
    print(text);
  }
}
```

Output

```
string: Hello
string..toLowerCase(): hello
string.toUpperCase(): HELLO
string.indexOf('e'): 1
string.length: 5
string.replace('H', 'j'): jello
string(substring(2,4): ll
string.trim(): Hello
string.format(): Bob earned $123.46
```

Files

Note: For security reasons, JavaScript in a browser requires the user to select the file to be processed. This example is based on node.js rather than browser-based JavaScript.

```
// This program creates a file, adds data to the file, displays the file,
// appends more data to the file, displays the file, and then deletes the f
// It will not run if the file already exists.

function calculateFahrenheit(celsius) {
    fahrenheit = celsius * 9 / 5 + 32
    return fahrenheit
}

function createFile(filename) {
    var fs = require('fs')

    fs.writeFile(filename, "Celsius,Fahrenheit\n",  function(err) {
        if (err) return console.error(err);
    });

    for(var celsius = 0; celsius <= 50; celsius++) {
        var fahrenheit = calculateFahrenheit(celsius);
        fs.appendFile(filename, celsius.toFixed(1) + "," +
            fahrenheit.toFixed(1) + "\n", function (err) {
            if (err) {
                return console.error(err);
            }
        });
    }
```

```
}

function readFile(filename) {
    var file = require('readline').createInterface( {
      input: require('fs').createReadStream(filename)
    });

    file.on('line', function (line) {
        console.log(line);
    });
}

function appendFile(filename) {
    var fs = require('fs')

    for(var celsius = 51; celsius <= 100; celsius++) {
        var fahrenheit = calculateFahrenheit(celsius);
        fs.appendFile(filename, celsius.toFixed(1) + "," +
            fahrenheit.toFixed(1) + "\n", function (err) {
            if (err) {
                return console.error(err);
            }
        });
    }
}

function deleteFile(filename) {
    var fs = require("fs");

    fs.unlink(filename, function(err) {
        if (err) {
            return console.error(err);
        }
    });
}

function fileExists(filename) {
    var fs = require('fs');
    return fs.existsSync(filename);
}

function main() {
    var filename = "~file.txt";
```

```
    if(fileExists(filename)) {
        console.log("File already exists.")
    } else {
        createFile(filename);
        readFile(filename);
        appendFile(filename);
        readFile(filename);
        deleteFile(filename);
    }
}

main();
```

Output

```
Celsius,Fahrenheit
0.0,32.0
1.0,33.8
2.0,35.6
...
98.0,208.4
99.0,210.2
100.0,212.0
```

References

- Wikiversity: Computer Programming

Python Examples

DAVE BRAUNSCHWEIG

Strings

```
# This program demonstrates string functions.


def main():
    string = "Hello"

    print("string: " + string)
    print("string.lower(): " + string.lower())
    print("string.upper(): " + string.upper())
    print("string.find('e'): " + str(string.find('e')))
    print("len(string): " + str(len(string)))
    print("string.replace('H', 'j'): " + string.replace('H', 'j'))
    print("string[::-1]: " + string[::-1])
    print("string[2:4]: " + string[2:4])
    print("string.strip('H'): " + string.strip('H'))

    name = "Bob"
    value = 123.456
    print("string.format(): {0} earned ${1:.2f}".format(name, value))


main()
```

Output

```
string: Hello
string.lower(): hello
string.upper(): HELLO
string.find('e'): 1
len(string): 5
string.replace('H', 'j'): jello
string[::-1]: olleH
string[2:4]: ll
string.strip('H'): ello
```

```
string.format(): Bob earned $123.46
```

Files

```
# This program creates a file, adds data to the file, displays the file,
# appends more data to the file, displays the file, and then deletes the fi
# It will not run if the file already exists.
#
# References:
#     https://www.mathsisfun.com/temperature-conversion.html
#     https://en.wikibooks.org/wiki/Python_Programming


import os
import sys


def calculate_fahrenheit(celsius):
    fahrenheit = celsius * 9 / 5 + 32
    return fahrenheit


def create_file(filename):
    try:
        with open(filename, "w") as file:
            file.write("Celsius,Fahrenheit\n")
            for celsius in range(0, 51):
                fahrenheit = calculate_fahrenheit(celsius)
                file.write("{:.1f},{:.1f}\n".format(celsius, fahrenheit))
    except:
        print("Error creating", filename)
        print(sys.exc_info()[1])


def read_file(filename):
    try:
        with open(filename, "r") as file:
            for line in file:
                line = line.strip()
                print(line)
        print()
```

```
    except:
        print("Error reading", filename)
        print(sys.exc_info()[1])


def append_file(filename):
    try:
        with open(filename, "a") as file:
            for celsius in range(51, 101):
                fahrenheit = calculate_fahrenheit(celsius)
                file.write("{:.1f},{:.1f}\n".format(celsius, fahrenheit))
    except:
        print("Error appending to", filename)
        print(sys.exc_info()[1])


def delete_file(filename):
    try:
        os.remove(filename)
    except:
        print("Error deleting", filename)
        print(sys.exc_info()[1])


def main():
    filename = "~file.txt"

    if os.path.isfile(filename):
        print("File already exists.")
    else:
        create_file(filename)
        read_file(filename)
        append_file(filename)
        read_file(filename)
        delete_file(filename)


main()
```

Output

```
Celsius,Fahrenheit
```

```
0.0,32.0
1.0,33.8
2.0,35.6
3.0,37.4
...
98.0,208.4
99.0,210.2
100.0,212.0
```

References

- Wikiversity: Computer Programming

Swift Examples

DAVE BRAUNSCHWEIG

Strings

```
// This program demonstrates string functions.

import Foundation

func main() {
    let string:String = "Hello"

    print("string: " + string)
    print("string.lowercased(): " + string.lowercased())
    print("string.uppercased(): " + string.uppercased())
    print("find(string, \"e\"): " + String(find(string:string, character:"e
    print("string.count: " + String(string.count))
    print("string.replacingOccurrences(of:\"H\", with:\"j\"): " + string.re
    print("string.reversed(): " + String(string.reversed()))
    print("substring(2, 2): " + substring(string:string, start:2, length:2)
    print("string.trimmingCharacters(\"H\"): " + string.trimmingCharacters(

    let name:String = "Bob"
    let value:Double = 123.456
    print("\(name) earned $" + String(format:"%.2f", value))
}

func find(string:String, character:Character) -> Int {
    var result: Int

    if let index = string.firstIndex(of:character) {
        result = string.distance(from: string.startIndex, to: index)
    } else {
        result = -1
    }
    return result
}

func substring(string:String, start:Int, length:Int) -> String {
    let startIndex = string.index(string.startIndex, offsetBy: start)
```

```
    let endIndex = string.index(string.startIndex, offsetBy: start + length
    return String(string[startIndex...endIndex])
}

main()
```

Output

```
string: Hello
string.lowercased(): hello
string.uppercased(): HELLO
find(string, "e"): 1
string.count: 5
string.replacingOccurrences(of:"H", with:"j"): jello
string.reversed(): olleH
substring(2, 2): ll
string.trimmingCharacters("H"): ello
Bob earned $123.46
```

Files

```
// This program creates a file, adds data to the file, displays the file,
// appends more data to the file, displays the file, and then deletes the f
// It will not run if the file already exists.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://developer.apple.com/library/content/documentation/Swift/Conc

import Foundation

func fileExists(filename:String) -> Bool {
    let fileManager = FileManager.default
    return fileManager.fileExists(atPath:filename)
}

func calculateFahrenheit(celsius:Double) -> Double {
    var fahrenheit: Double
    fahrenheit = celsius * 9 / 5 + 32
    return fahrenheit
```

```
}

func createFile(filename:String) {
    var text: String
    var fahrenheit: Double

    text = "Celsius,Fahrenheit\n"
    for celsius in stride(from: 0.0, through: 50.0, by: 1.0) {
        fahrenheit = calculateFahrenheit(celsius:celsius)
        text += String(celsius) + "," + String(fahrenheit) + "\n"
    }

    do {
        try text.write(toFile: filename, atomically: true, encoding: .utf8)
    } catch {
        print("Error creating ", filename)
        print(error.localizedDescription)
    }
}

func readFile(filename:String) {
    var text = ""

    do {
        text = try String(contentsOfFile: filename, encoding: .utf8)

        let lines = text.components(separatedBy:"\n")
        for line in lines {
            print(line)
        }
    } catch {
        print("Error reading " + filename)
        print(error.localizedDescription)
    }
}

func appendFile(filename:String) {
    var text: String
    var fahrenheit: Double

    do {
        text = try String(contentsOfFile: filename, encoding: .utf8)
```

```
        for celsius in stride(from: 51.0, through: 100.0, by: 1.0) {
            fahrenheit = calculateFahrenheit(celsius:celsius)
            text += String(celsius) + "," + String(fahrenheit) + "\n"
        }

        try text.write(toFile: filename, atomically: true, encoding: .utf8)
    } catch {
        print("Error appending to ", filename)
        print(error.localizedDescription)
    }
}

func deleteFile(filename:String) {
    do {
        let fileManager = FileManager.default
        try fileManager.removeItem(atPath:filename)
    } catch {
        print("Error deleting", filename)
        print(error.localizedDescription)
    }
}

func main() {
    let filename:String = "~file.txt"

    if (fileExists(filename:filename)) {
        print("File already exists.")
    }
    else {
        createFile(filename:filename)
        readFile(filename:filename)
        appendFile(filename:filename)
        readFile(filename:filename)
        deleteFile(filename:filename)
    }
}

main()
```

Output

```
Celsius,Fahrenheit
```

```
0.0,32.0
1.0,33.8
2.0,35.6
3.0,37.4
...
98.0,208.4
99.0,210.2
100.0,212.0
```

References

- Wikiversity: Computer Programming

Practice: Strings and Files

KENNETH LEROY BUSBEE

Review Questions

True / False

1. The character data type in C++ uses the double quote marks, like: char grade = “A”;
2. Sizeof is an operator that tells you how many bytes a data type occupies in storage.
3. Typedef helps people who can’t hear and is one of the standard accommodation features of a programming language for people with a learning disability.
4. The sequence operator should be used when defining variables in order to save space.
5. Programming can be both enjoyable and frustrating.
6. Text files are hard to create.
7. A filespec refers to a very small (like a spec dust) file.
8. A device token is a special non zero value the operating system gives your program and is associated with the file that you requested to be opened.
9. Programmers should not worry about closing a file.
10. Where you define an item, that is global or local scope, is rarely important.

Answers:

1. false
2. true
3. false
4. false
5. true
6. false
7. false
8. true
9. false
10. false

Short Answer

1. Describe the normal operations allowed with the string data type.
2. Describe why unary positive is worthless.
3. Describe how unary negative works.

Activities

Complete the following activities using pseudocode, a flowcharting tool, or your selected programming language. Use separate functions for input, each type of processing, and output. Avoid global variables by passing parameters and returning results. Create test data to validate the

accuracy of each program. Add comments at the top of the program and include references to any resources used.

String Activities

1. Create a program that asks the user for a single line of text containing a first name and last name, such as `Firstname Lastname`. Use string functions/methods to parse the line and print out the name in the form last name, first initial, such as `Lastname, F.` Include a trailing period after the first initial. Handle invalid input errors, such as extra spaces or missing name parts.
2. Create a program that asks the user for a line of text. Use string functions/methods to delete leading, trailing, and duplicate spaces, and then print the line of text backwards. For example:
```
   the   cat   in   the   hat
tah eht ni tac eht
```
3. Create a program that asks the user for a line of comma-separated-values. It could be a sequence of test scores, names, or any other values. Use string functions/methods to parse the line and print out each item on a separate line. Remove commas and any leading or trailing spaces from each item when printed.
4. Create a program that asks the user for a line of text. Then ask the user for the number of characters to print in each line, the number of lines to be printed, and a scroll direction, right or left. Using the given line of text, duplicate the text as needed to fill the given number of characters per line. Then print the requested number of lines, shifting the entire line's content by one character, left or right, each time the line is printed. The first or last character will be shifted / appended to the other end of the string. For example:
```
Repeat this. Repeat this.
epeat this. Repeat this. R
peat this. Repeat this. Re
```

File Activities

Note: Each of the following activities uses code only to read the file. It is not necessary to use code to create the file.

1. Using a text editor or IDE, copy the following list of names and grade scores and save it as a text file named `scores.txt`:
```
Name,Score
Joe Besser,70
Curly Joe DeRita,0
Larry Fine,80
Curly Howard,65
Moe Howard,100
Shemp Howard,85
```
Create a program that displays high, low, and average scores based on input from `scores.txt`. Verify that the file exists and then use string functions/methods to parse the file content and add each score to an array. Display the array contents and then calculate and display the high, low, and average score. Format the average to two decimal places. Note that the program must work for any given number of scores in the file. Do not assume there will

always be six scores.

2. Create a program that displays high, low, and average scores based on input from `scores.txt`. Verify that the file exists and then use string functions/methods to parse the file content and add each score to an array. Display the array contents and then calculate and display the high, low, and average score. Format the average to two decimal places. Include error handling in case the file is formatted incorrectly. Note that the program must work for any given number of scores in the file. Do not assume there will always be six scores.
3. Create a program that asks the user for the name of a text/HTML file that contains HTML tags, such as:
   `<p><strong>This is a bold paragraph.</strong></p>`
   Verify that the file exists and then use string methods to search for and remove all HTML tags from the text, saving each removed tag in an array. Display the untagged text and then display the array of removed tags. For example:
   ```
   This is a bold paragraph.
   <p>
   <strong>
   </strong>
   </p>
   ```
4. Using a text editor or IDE, create a text file of names and addresses to use for testing based on the following format:
   ```
   Firstname Lastname
   123 Any Street
   City, State/Province/Region PostalCode
   ```
   Include a blank line between addresses, and include at least three addresses in the file. Create a program that verifies that the file exists, and then processes the file and displays each address as a single line of comma-separated values in the form:
   `Lastname, Firstname, Address, City, State/Province/Region, PostalCode`

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++
- Wikiversity: Computer Programming

PART VIII

OBJECT-ORIENTED PROGRAMMING

Overview

This chapter introduces object-oriented programming, with a focus on understanding object-oriented concepts and terminology. It includes short examples of objects and classes in different programming languages.

Chapter Outline

Learning Objectives

1. Understand key terms and definitions.
2. Gain exposure to object-oriented programming.
3. Given example source code, create a program that uses object-oriented programming concepts to solve a given problem.

Objects and Classes

DAVE BRAUNSCHWEIG

Overview

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). There is significant diversity of OOP languages, but the most popular ones are class-based, meaning that objects are instances of classes, which typically also determine their type.[1]

Discussion

Thus far, we have focused on procedural programming. Based on structured programming, procedures (routines, subroutines, or functions) contain a series of computational steps to be carried out. Any given procedure might be called at any point during a program's execution, including by other procedures or itself. The focus of procedural programming is to break down a programming task into a collection of variables, data structures, and subroutines.[2] Small programs and scripts tend to be easier to develop using a simple procedural approach.

Object-oriented programming instead breaks down a programming task into objects that expose behavior (methods) and data (members or attributes) using interfaces. The most important distinction is that while procedural programming uses procedures to operate on separate data structures, object-oriented programming bundles the two together, so an "object", which is an instance of a class, operates on its "own" data structure.[3] Larger programs benefit from better code and data isolation and reuse provided by an object-oriented approach.

Objects and classes are often designed to represent real-world objects. Consider a door as an example of a real-world object. Most doors have limited functionality. They may be opened and closed, and locked and unlocked. In procedural programming, we might design functions to open, close, lock, and unlock a door, such as:

```
Procedural Programming - Functions
OpenDoor(door)
CloseDoor(door)
LockDoor(door)
UnlockDoor(door)
```

Object-oriented programming combines code and data, so that, rather than having separate

1. Wikipedia: Object-oriented programming
2. Wikipedia: Object-oriented programming
3. Wikipedia: Object-oriented programming

functions act on doors, we design doors that have methods that can act on themselves. Methods represent something the object can do, and are typically defined using verbs. Object-oriented door pseudocode might look like:

```
Object-Oriented Programming - Methods
door.Open()
door.Close()
door.Lock()
door.Unlock()
```

Objects may also have attributes, something the object is or has, and are typically defined using nouns or adjectives. Door attributes might include:

```
Object-Oriented Programming - Attributes
door.Height
door.Width
door.Color
door.Closed
door.Locked
```

When we write code to define a generic door, we would create a door class. The door class would contain all of the methods a door can perform and all of the attributes a door might have. We would then create instances of the class (objects) to represent specific doors, such as a front door, back door, or room door on a house, or a left door and right door on a car.

Key Terms

attribute
: A specification that defines a property of an object.[4]

class
: An extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions or methods).[5]

instance
: :A concrete occurrence of an object.[6]

method
: A specification that defines a procedure or behavior of an object.[7]

object
: A particular instance of a class where the object can be a combination of variables, functions, and data structures.[8]

4. Wikipedia: Attribute (computing)
5. Wikipedia: Class (computer programming)
6. Wikipedia: Instance (computer science)
7. Wikipedia: Method (computer programming)
8. Wikipedia: Object (computer science)

this, self, or Me
: Keywords used in some computer programming languages to refer to the object, class, or other entity that the currently running code is part of.[9]

References

- Wikibooks: Object-Oriented Programming
- Wikipedia: Object-oriented programming
- Wikiversity: Computer Programming

9. Wikipedia: this (computer programming)

Encapsulation

DAVE BRAUNSCHWEIG

Overview

Encapsulation is one of the fundamentals of OOP (object-oriented programming). It refers to the bundling of data with the methods that operate on that data. Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties' direct access to them. Publicly accessible methods are generally provided in the class (so-called getters and setters) to access the values, and other client classes call these methods to retrieve and modify the values within the object.[1]

Discussion

The most important principle of object orientation is *encapsulation*: the idea that data inside the object should only be accessed through a public *interface* – that is, the object's methods.

If we want to use the data stored in an object to perform an action or calculate a derived value, we define a method associated with the object which does this. Then whenever we want to perform this action we call the method on the object. We consider it bad practice to retrieve the information from inside the object and write separate code to perform the action outside of the object.

Encapsulation is a good idea for several reasons:

- the functionality is defined *in one place* and not in multiple places.
- it is defined in a logical place – the place where the data is kept.
- data inside our object is not modified unexpectedly by external code in a completely different part of our program.
- when we use a method, we only need to know what result the method will produce – we don't need to know details about the object's internals in order to use it. We could switch to using another object which is completely different on the inside, and not have to change any code because both objects have the same interface.

We can say that the object "knows how" to do things with its own data, and it's a bad idea for us to access its internals and do things with the data ourselves. If an object doesn't have an interface method which does what we want to do, we should add a new method or update an existing one.

Some languages have features which allow us to enforce encapsulation strictly. In Java or C++, we can define access permissions on object attributes, and make it illegal for them to be accessed from outside the object's methods. In Java it is also considered good practice to write setters and getters for all attributes, even if the getter simply retrieves the attribute and the setter just assigns it the value of the parameter which you pass in.

In Python, encapsulation is not enforced by the language, but there is a convention that we can use to indicate that a property is intended to be private and is not part of the object's public interface: we begin its name with an underscore. Python also supports the use of a property decorator to replace a simple attribute with a method without changing the object's interface.

1. Wikipedia: Encapsulation (computer programming)

Key Terms

abstraction
: A technique for arranging complexity of computer systems so that functionality may be separated from specific implementation details.[23]

accessor
: A method used to return the value of a private member variable, also known as a getter method.[4]

encapsulation
: A language mechanism for restricting direct access to some of an object's components.[5]

information hiding
: The principle of segregation of the design decisions in a computer program from other parts of the program. See encapsulation.[6]

mutator
: A method used to control changes to a private member variable, also known as a setter method.[7]

private
: An access modifier that restricts visibility of a property or method to the class in which it is defined.[8]

public
: An access modifier that opens visibility of a property or method to all other classes.[9]

References

- Read the Docs: Object-Oriented Programming in Python
- Wikiversity: Computer Programming

2. Wikipedia: Object-oriented programming
3. Wikipedia: Abstraction (computer science)
4. Wikipedia: Mutator method
5. Wikipedia: Encapsulation (computer programming)
6. Wikipedia: Information hiding
7. Wikipedia: Mutator method
8. Wikipedia: Access modifiers
9. Wikipedia: Access modifiers

Inheritance and Polymorphism

DAVE BRAUNSCHWEIG

Overview

In object-oriented programming, **inheritance** is the mechanism of basing an object or class upon another object (prototypical inheritance) or class (class-based inheritance), retaining similar implementation. In most class-based object-oriented languages, an object created through inheritance (a "child object") acquires all the properties and behaviors of the parent object (except: constructors, destructor, overloaded operators and friend functions of the base class). Inheritance allows programmers to create classes that are built upon existing classes, to specify a new implementation while maintaining the same behaviors (realizing an interface), to reuse code and to independently extend original software via public classes and interfaces.[1]

Discussion

Inheritance is a way of arranging objects in a hierarchy from the most general to the most specific. An object which *inherits* from another object is considered to be a *subtype* of that object. An example might include Instructor and Student, each of which inherit from Person. When we can describe the relationship between two objects using the phrase *is-a*, that relationship is inheritance.

We also often say that a class is a *subclass* or *child class* of a class from which it inherits, or that the other class is its *superclass* or *parent class*. We can refer to the most generic class at the base of a hierarchy as a *base class*.

Inheritance can help us to represent objects which have some differences and some similarities in the way they work. We can put all the functionality that the objects have in common in a base class, and then define one or more subclasses with their own custom functionality.

Inheritance is also a way of reusing existing code easily. If we already have a class which does *almost* what we want, we can create a subclass in which we partially override some of its behavior, or perhaps add some new functionality.

In some statically typed languages inheritance is very popular because it allows the programmer to work around some of the restrictions of static typing. If an instructor and a student are both a kind of person, we can write a function which accepts a parameter of type Person and have it work on both instructor and student objects because they both inherit from Person. This is known as *polymorphism*.

Key Terms

inheritance
: An object or class being based on another object or class, using the same implementation or specifying a new implementation to maintain the same behavior.[2]

1. Wikipedia: Inheritance (object-oriented programming)

polymorphism
: The provision of a single interface to entities of different types.[3]

References

- Read the Docs: Object-Oriented Programming in Python
- Wikiversity: Computer Programming

2. Wikipedia: Inheritance (object-oriented programming)
3. Wikipedia: Polymorphism (computer science)

C++ Examples

DAVE BRAUNSCHWEIG

Objects

```
// This class converts temperature between Celsius and Fahrenheit.
// It may be used by assigning a value to either Celsius or Fahrenheit
// and then retrieving the other value, or by calling the ToCelsius or
// ToFahrenheit methods directly.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/C%2B%2B_Programming

#include <iostream>

using namespace std;

class Temperature {
    public:
        double getCelsius(void);
        void setCelsius(double value);
        double getFahrenheit(void);
        void setFahrenheit(double value);
        double toCelsius(double fahrenheit);
        double toFahrenheit(double celsius);

    private:
        double celsius;
        double fahrenheit;
};

double Temperature::getCelsius(void) {
    return celsius;
}

void Temperature::setCelsius(double value) {
    celsius = value;
    fahrenheit = toFahrenheit(celsius);
}
```

```
double Temperature::getFahrenheit(void) {
    return fahrenheit;
}

void Temperature::setFahrenheit(double value) {
    fahrenheit = value;
    celsius = toCelsius(fahrenheit);
}

double Temperature::toCelsius(double fahrenheit) {
    return (fahrenheit - 32) * 5 / 9;
}

double Temperature::toFahrenheit(double celsius) {
    return celsius * 9 / 5 + 32;
}

// This program creates instances of the Temperature class to convert Celsi
// and Fahrenheit temperatures.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/C%2B%2B_Programming

int main() {
    Temperature temp1;
    temp1.setCelsius(100.0);
    cout << "temp1.celsius = " << temp1.getCelsius() << endl;
    cout << "temp1.fahrenheit = " << temp1.getFahrenheit() << endl;
    cout << endl;

    Temperature temp2;
    temp2.setFahrenheit(100.0);
    cout << "temp2.fahrenheit = " << temp2.getFahrenheit() << endl;
    cout << "temp2.celsius = " << temp2.getCelsius() << endl;
}
```

Output

```
temp1.celsius = 100
temp1.fahrenheit = 212
```

```
temp2.fahrenheit = 100
temp2.celsius = 37.7778
```

References

- Wikiversity: Computer Programming

C# Examples

DAVE BRAUNSCHWEIG

Objects

```
// This program creates instances of the Temperature class to convert Celsi
// and Fahrenheit temperatures.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/C_Sharp_Programming

using System;

public class Objects
{
    public static void Main(String[] args)
    {
        Temperature temp1 = new Temperature(celsius: 0);
        Console.WriteLine("temp1.Celsius = " + temp1.Celsius.ToString());
        Console.WriteLine("temp1.Fahrenheit = " + temp1.Fahrenheit.ToString
        Console.WriteLine("");

        temp1.Celsius = 100;
        Console.WriteLine("temp1.Celsius = " + temp1.Celsius.ToString());
        Console.WriteLine("temp1.Fahrenheit = " + temp1.Fahrenheit.ToString
        Console.WriteLine("");

        Temperature temp2 = new Temperature(fahrenheit: 0);
        Console.WriteLine("temp2.Fahrenheit = " + temp2.Fahrenheit.ToString
        Console.WriteLine("temp2.Celsius = " + temp2.Celsius.ToString());
        Console.WriteLine("");

        temp2.Fahrenheit = 100;
        Console.WriteLine("temp2.Fahrenheit = " + temp2.Fahrenheit.ToString
        Console.WriteLine("temp2.Celsius = " + temp2.Celsius.ToString());
    }
}

// This class converts temperature between Celsius and Fahrenheit.
```

```
// It may be used by assigning a value to either Celsius or Fahrenheit
// and then retrieving the other value, or by calling the ToCelsius or
// ToFahrenheit methods directly.

public class Temperature
{
    double _celsius;
    double _fahrenheit;

    public double Celsius
    {
        get
        {
            return _celsius;
        }

        set
        {
            _celsius = value;
            _fahrenheit = ToFahrenheit(value);
        }
    }

    public double Fahrenheit
    {
        get
        {
            return _fahrenheit;
        }

        set
        {
            _fahrenheit = value;
            _celsius = ToCelsius(value);
        }
    }

    public Temperature(double? celsius = null, double? fahrenheit = null)
    {
        if (celsius.HasValue)
        {
            this.Celsius = Convert.ToDouble(celsius);
        }
```

```
        if (fahrenheit.HasValue)
        {
            this.Fahrenheit = Convert.ToDouble(fahrenheit);
        }
    }

    public double ToCelsius(double fahrenheit)
    {
        return (fahrenheit - 32) * 5 / 9;
    }

    public double ToFahrenheit(double celsius)
    {
        return celsius * 9 / 5 + 32;
    }
}
```

Output

```
temp1.Celsius = 0
temp1.Fahrenheit = 32

temp1.Celsius = 100
temp1.Fahrenheit = 212

temp2.Fahrenheit = 0
temp2.Celsius = -17.7777777777778

temp2.Fahrenheit = 100
temp2.Celsius = 37.7777777777778
```

References

- Wikiversity: Computer Programming

Java Examples

DAVE BRAUNSCHWEIG

Objects

```
// This program creates instances of the Temperature class to convert Celsi
// and Fahrenheit temperatures.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/Java_Programming

import java.util.*;

class Main {
    public static void main(String[] args) {
        Temperature temp1 = new Temperature();
        temp1.setCelsius(100.0);
        System.out.println("temp1.celsius = " + temp1.getCelsius().toString
        System.out.println("temp1.fahrenheit = " + temp1.getFahrenheit().to
        System.out.println("");

        Temperature temp2 = new Temperature();
        temp2.setFahrenheit(100.0);
        System.out.println("temp2.fahrenheit = " + temp2.getFahrenheit().to
        System.out.println("temp2.celsius = " + temp2.getCelsius().toString
    }
}

// This class converts temperature between Celsius and Fahrenheit.
// It may be used by assigning a value to either Celsius or Fahrenheit
// and then retrieving the other value, or by calling the ToCelsius or
// ToFahrenheit methods directly.

class Temperature {
    Double celsius;
    Double fahrenheit;

    public Double getCelsius() {
        return celsius;
```

```
    }

    public void setCelsius(Double value) {
        celsius = value;
        fahrenheit = toFahrenheit(celsius);
    }

    public Double getFahrenheit() {
        return fahrenheit;
    }

    public void setFahrenheit(Double value) {
        fahrenheit = value;
        celsius = toCelsius(fahrenheit);
    }

    public Double toCelsius(Double fahrenheit) {
        return (fahrenheit - 32) * 5 / 9;
    }

    public Double toFahrenheit(Double celsius) {
        return celsius * 9 / 5 + 32;
    }
}
```

Output

```
temp1.celsius = 100.0
temp1.fahrenheit = 212.0

temp2.fahrenheit = 100.0
temp2.celsius = 37.77777777777778
```

References

- Wikiversity: Computer Programming

JavaScript Examples

DAVE BRAUNSCHWEIG

Objects

```
// This class converts temperature between Celsius and Fahrenheit.
// It may be used by assigning a value to either Celsius or Fahrenheit
// and then retrieving the other value, or by calling the ToCelsius or
// ToFahrenheit methods directly.

class Temperature {
    constructor() {
        this._celsius = 0;
        this._fahrenheit = 32;
    }

    get celsius() {
        return this._celsius;
    }

    set celsius(value) {
        this._celsius = value;
        this._fahrenheit = this.toFahrenheit(value);
    }

    get fahrenheit() {
        return this._fahrenheit;
    }

    set fahrenheit(value) {
        this._fahrenheit = value;
        this._celsius = this.toCelsius(value);
    }

    toCelsius(fahrenheit) {
        return (fahrenheit - 32) * 5 / 9
    }

    toFahrenheit(celsius) {
        return celsius * 9 / 5 + 32
```

```
    }
}

// This program creates instances of the Temperature class to convert Celsi
// and Fahrenheit temperatures.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/JavaScript

main()

function main() {
    var temp1 =  new Temperature();
    temp1.celsius = 0
    output("temp1.celsius = " + temp1.celsius);
    output("temp1.fahrenheit = " + temp1.fahrenheit);
    output("");

    temp1.celsius = 100;
    output("temp1.celsius = " + temp1.celsius);
    output("temp1.fahrenheit = " + temp1.fahrenheit);
    output("");

    var temp2 = new Temperature();
    temp2.fahrenheit = 0
    output("temp2.fahrenheit = " + temp2.fahrenheit);
    output("temp2.celsius = " + temp2.celsius);
    output("");

    temp2.fahrenheit = 100;
    output("temp2.fahrenheit = " + temp2.fahrenheit);
    output("temp2.celsius = " + temp2.celsius);
}

function output(text) {
  if (typeof document === 'object') {
    document.write(text);
  }
  else if (typeof console === 'object') {
    console.log(text);
  }
  else {
```

```
    print(text);
  }
}
```

Output

```
temp1.celsius = 0
temp1.fahrenheit = 32

temp1.celsius = 100
temp1.fahrenheit = 212

temp2.fahrenheit = 0
temp2.celsius = -17.77777777777778

temp2.fahrenheit = 100
temp2.celsius = 37.77777777777778
```

References

- Wikiversity: Computer Programming

Python Examples

DAVE BRAUNSCHWEIG

Objects

```
# This class converts temperature between Celsius and Fahrenheit.
# It may be used by assigning a value to either Celsius or Fahrenheit
# and then retrieving the other value, or by calling the to_celsius or
# to_fahrenheit methods directly.
#
# References:
#     https://www.mathsisfun.com/temperature-conversion.html
#     https://en.wikibooks.org/wiki/Python_Programming

class Temperature:
    _celsius = None
    _fahrenheit = None

    @property
    def celsius(self):
        return self._celsius

    @celsius.setter
    def celsius(self, value):
        self._celsius = float(value)
        self._fahrenheit = self.to_fahrenheit(self._celsius)

    @property
    def fahrenheit(self):
        return self._fahrenheit

    @fahrenheit.setter
    def fahrenheit(self, value):
        self._fahrenheit = float(value)
        self._celsius = self.to_celsius(self._fahrenheit)

    def __init__(self, celsius=None, fahrenheit=None):
        if celsius != None:
            self._celsius = celsius
            self._fahrenheit = self.to_fahrenheit(celsius)
```

```
        if fahrenheit != None:
            self._fahrenheit = fahrenheit
            self._celsius = self.to_celsius(fahrenheit)

    def to_celsius(self, fahrenheit):
        return (fahrenheit - 32) * 5 / 9

    def to_fahrenheit(self, celsius):
        return celsius * 9 / 5 + 32


# This program creates instances of the Temperature class to convert Cesius
# and Fahrenheit temperatures.

def main():
    temp1 = Temperature(celsius=0)
    print("temp1.celsius =", temp1.celsius)
    print("temp1.fahrenheit =", temp1.fahrenheit)
    print("")

    temp1.celsius = 100
    print("temp1.celsius =", temp1.celsius)
    print("temp1.fahrenheit =", temp1.fahrenheit)
    print("")

    temp2 = Temperature(fahrenheit=0)
    print("temp2.fahrenheit =", temp2.fahrenheit)
    print("temp2.celsius =", temp2.celsius)
    print("")

    temp2.fahrenheit = 100
    print("temp2.fahrenheit =", temp2.fahrenheit)
    print("temp2.celsius =", temp2.celsius)

main()
```

Output

```
temp1.celsius = 0
temp1.fahrenheit = 32.0

temp1.celsius = 100.0
```

```
temp1.fahrenheit = 212.0

temp2.fahrenheit = 0
temp2.celsius = -17.77777777777778

temp2.fahrenheit = 100.0
temp2.celsius = 37.77777777777778
```

References

- Wikiversity: Computer Programming

Swift Examples

DAVE BRAUNSCHWEIG

Objects

```
// This class converts temperature between Celsius and Fahrenheit.
// It may be used by assigning a value to either Celsius or Fahrenheit
// and then retrieving the other value, or by calling the ToCelsius or
// ToFahrenheit methods directly.

class Temperature {
    var _celsius:Double = 0
    var _fahrenheit:Double = 32

    init(celsius:Double?=nil, fahrenheit:Double?=nil) {
        if celsius != nil {
            self.celsius = celsius!
        }

        if fahrenheit != nil {
            self.fahrenheit = fahrenheit!
        }
    }

    var celsius: Double {
        get {
            return self._celsius
        }
        set {
            self._celsius = newValue
            self._fahrenheit = toFahrenheit(celsius:self._celsius)
        }
    }

    var fahrenheit: Double {
        get {
            return self._fahrenheit
        }
        set {
            self._fahrenheit = newValue
```

```
            self._celsius = toCelsius(fahrenheit:self._fahrenheit)
        }
    }

    func getCelsius() -> Double {
        return self.celsius
    }

    func setCelsius(celsius:Double) {
        self.celsius = celsius
        self.fahrenheit = toFahrenheit(celsius:celsius)
    }

    func getFahrenheit() -> Double {
        return self.fahrenheit
    }

    func setFahrenheit(fahrenheit:Double) {
        self.fahrenheit = fahrenheit
        self.celsius = toCelsius(fahrenheit:fahrenheit)
    }

    func toCelsius(fahrenheit:Double) -> Double {
        return (fahrenheit - 32) * 5 / 9
    }

    func toFahrenheit(celsius:Double) -> Double {
        return celsius * 9 / 5 + 32
    }
}

// This program creates instances of the Temperature class to convert Celsi
// and Fahrenheit temperatures.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://developer.apple.com/library/content/documentation/Swift/Conc

func main() {
    let temp1 =  Temperature(celsius:0);
    print("temp1.celsius = " + String(temp1.celsius));
    print("temp1.fahrenheit = " + String(temp1.fahrenheit));
    print("");
```

```
    temp1.celsius = 100;
    print("temp1.celsius = " + String(temp1.celsius));
    print("temp1.fahrenheit = " + String(temp1.fahrenheit));
    print("");

    let temp2 = Temperature(fahrenheit:0);
    print("temp2.fahrenheit = " + String(temp2.fahrenheit));
    print("temp2.celsius = " + String(temp2.celsius));
    print("");

    temp2.fahrenheit = 100;
    print("temp2.fahrenheit = " + String(temp2.fahrenheit));
    print("temp2.celsius = " + String(temp2.celsius));
}

main()
```

Output

```
temp1.celsius = 0.0
temp1.fahrenheit = 32.0

temp1.celsius = 100.0
temp1.fahrenheit = 212.0

temp2.fahrenheit = 0.0
temp2.celsius = -17.7777777777778

temp2.fahrenheit = 100.0
temp2.celsius = 37.7777777777778
```

References

- Wikiversity: Computer Programming

Practice

KENNETH LEROY BUSBEE

Review Questions

Answer the following statements as either true or false:

1. Procedural programming and object-oriented programming cannot be done with the same compiler/IDE.
2. Object-oriented programming encapsulates data and functions.

Answers:

1. false
2. true

Short Answer

1. Describe the fundamental differences between procedural (modular structured) programming and object-oriented programming.

Activities

Complete the following activities using your selected programming language. Use separate functions for input, each type of processing, and output. Avoid global variables by passing parameters and returning results. Create test data to validate the accuracy of each program. Add comments at the top of the program and include references to any resources used.

1. Review MathsIsFun: Area of Plane Shapes. Create a program that asks the user what shape they would like to calculate the area for. Use if/else conditional statements to determine their selection and then gather the appropriate input and calculate and display the area of the shape. Perform all area calculations using a ShapeArea class that has separate methods to calculate and return the area for different shapes. Include data validation in the class and error handling in the main program.
2. Create a program that asks the user how old they are in years. Then ask the user if they would like to know how old they are in months, days, hours, or seconds. Use if/else conditional statements to display their approximate age in the selected timeframe. Perform all calculations using an AgeConverter class that accepts the age in years during initialization and has separate properties and methods to calculate and return the age in months, days, hours, and seconds. Include data validation in the class and error handling in the main program.
3. Review Wikipedia: Zeller's congruence. Create a program that asks the user for their birthday (year, month, and day) and then calculate and display the day of the week on which they were born. Use if/else conditional statements to convert the numeric day of the week to the correct string representation (Monday, Tuesday, Wednesday, etc.). Perform all calculations using a

DayOfWeek class that accepts the year, month, and day during initialization and has separate properties and methods to calculate and return the day of week as a number, as an abbreviated string (Mon, Tue, etc.), and as a full string (Monday, Tuesday, etc.). Include data validation in the class and error handling in the main program.

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++
- Wikiversity: Computer Programming