PROGRAMMING FUNDAMENTALS

A Modular Structured Approach, 2nd Edition

Kenneth Leroy Busbee and Dave Braunschweig

Programming Fundamentals

Programming Fundamentals

A Modular Structured Approach, 2nd Edition

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Contents

Part II. Data and Operators

Part VI. Arrays

Contents

Chapters

About this Book

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

A Note to Readers

Welcome to Programming Fundamentals – A Modular Structured Approach, 2nd Edition!

The original content for this book was created by Kenneth Leroy Busbee and written specifically for his course based on C++. The goal for this second edition is to make it programming-language neutral, so that it may serve as an introductory programming textbook for students using any of a variety of programming languages, including C++, C#, Java, JavaScript, Python, and Swift. Other languages will be considered upon request.

Programming concepts are introduced generically, with logic demonstrated in pseudocode and flowchart form, followed by examples for different programming languages. Emphasis is placed on a modular, structured approach that supports reuse, maintenance, and self-documenting code.

As you begin to review this edition, please keep the audience in mind. If something is missing, think about whether that concept applies to programming in general or only to certain programming languages, and whether it is a fundamental, first-semester programming concept or something better addressed in a more advanced textbook.

You are encouraged to make use of the Comments page at the end of the book whenever you have suggestions or concerns regarding content or approach. All suggestions will be reviewed and considered.

Dave Braunschweig

About this Textbook

Programming Fundamentals – A Modular Structured Approach, 2nd Edition is an adaptation of "*Programming Fundamentals – A Modular Structured Approach using C++*", written by Kenneth Leroy Busbee, a faculty member at Houston Community College in Houston, Texas. The materials used in the first edition were originally developed by Busbee and others as independent modules for publication within the Connexions environment. The original source is available at https://cnx.org/contents/MDgA8wfz@22.2:YzfkjC2r@17/.

This second edition, adapted by Dave Braunschweig, expands on the original vision by supporting multiple programming languages with pseudocode and flowcharts, and includes example code in C++, C#, Java, JavaScript, Python, and Swift.

Programming fundamentals are often divided into three college courses: Modular/Structured, Object Oriented and Data Structures. This textbook/collection covers the first of those three courses.

Learning Modules

The learning modules of this textbook were written as **standalone** modules. Students using a collection of modules as a textbook will usually view its contents by reading the modules sequentially as presented by the author of the collection.

However, many readers of these modules may find them as a result of an Internet search. The textbook design allows the author of a module to create web links to other modules and Internet locations and designate any necessary prerequisites.

Conceptual Approach

The learning modules of this textbook were, for the most part, written without consideration of a specific programming language. Concepts are presented generically, with program logic demonstrated first in pseudocode and flowchart format. Language-specific examples follow the general overview.

Re-use and Customization

PDF Conversion Problems

There are several known PDF printing problems. A description of the known problems are:

1. When it converts an “Example” the PDF displays the first line of an example properly but indents the remaining lines of the example. This problem occurs for the printing of a book (because it prints a PDF) and downloading either a module or a textbook/collection as a PDF.
2. Within C++ there are three operators that do not convert properly into PDF format.

decrement	—	which is two minus signs
insertion	<<	which is two less than signs
extraction	>>	which is two greater than signs

References

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

Author Acknowledgements

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

1st Edition Acknowledgements

I wish to acknowledge the many people who have helped me and have encouraged me in this project.

1. Mr. Abass Alamnehe, who is a fellow faculty member at Houston Community College. He has encouraged the use of Connexions as an "open source" publishing concept. His comments on several modules have led directly to the improvement of the materials in this textbook/collection.
2. The hundreds (most likely a thousand plus) students that I have taken programming courses that I have taught since 1984. The languages include: COBOL, mainframe IBM assembly, Intel assembly, Pascal, "C" and "C++". They have often suggested that I write my own book because they thought that I was explaining the subject matter better than the author of the textbook that we were using. Little did my students understand that directly or indirectly they aided in the improvement of the materials from which I taught as well as improving me as a teacher.
3. To my future students and all those that will use this textbook/collection. They will provide suggestions for improvement as well as being the thousand eyes identifying the hard to find typos, etc.
4. My wife, Carol, who supports me in all that I do. She has tolerated the many hours that I have spent in concentration on developing the modules that comprise this work. Without her support, this work would not have happened.

Kenneth Leroy Busbee

2nd Edition Acknowledgements

I wish to acknowledge the many people who have helped make this edition possible, including:

- Kenneth Leroy Busbee for his initial vision and willingness to share *Programming Fundamentals – A Modular Structured Approach using C++* as CC-BY, making it possible to build on his success.
- University of Cape Town for likewise sharing *Object-Oriented Programming in Python* as CC-BY-SA and making it possible to build on their efforts.
- Jay Singelmann and Jean Longhurst, who first taught me structured programming.
- Joyce Farrell, whose *Programming Logic and Design* book I have used for several years and has no doubt influenced my approach.
- Devin Cook for developing *Flowgorithm*, releasing it as free software, and graciously allowing its use to generate most of the pseudocode and flowcharts used in this edition of the book.
- Zoe Wake Hyde and the staff and volunteers at Rebus Community for providing a community and platform to create and collaborate on open content.
- April Browne, Carol Potaczek, and Maisie Sparks for providing subject matter expertise and recommendations for content improvement.
- My wife and family for accepting my dedication to open educational resources and loving me

anyway.

Dave Braunschweig

References

- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++
- Cover Art: Puzzle pieces – CC0 by MsReadIt, downloaded from https://openclipart.org/detail/231093/puzzle-pieces

PART I

INTRODUCTION TO PROGRAMMING

Overview

This chapter introduces programming, the software development process, tools and methods used to develop and test programs. These include integrated development environments (IDEs), version control, input and output, and a Hello World program in pseudocode and flowchart format. The programming languages C++, C#, Java, JavaScript, Python, and Swift are introduced with example code.

Chapter Outline

Learning Objectives

1. Understand key terms and definitions.
2. Create pseudocode for a programming problem.
3. Create a flowchart for a programming problem.
4. Perform software testing for a programming problem.
5. List the four categories and give examples of errors that may be encountered when using an Integrated Development Environment (IDE).
6. Test an Integrated Development Environment using a Hello World program.
7. Modify an existing program to meet given requirements.

Systems Development Life Cycle

KENNETH LEROY BUSBEE

Overview

The **Systems Development Life Cycle** (SDLC) describes a process for planning, creating, testing, and deploying an information system. A number of SDLC models or methodologies have been implemented to address different system needs, including waterfall, spiral, Agile software development, rapid prototyping, and incremental.[1]

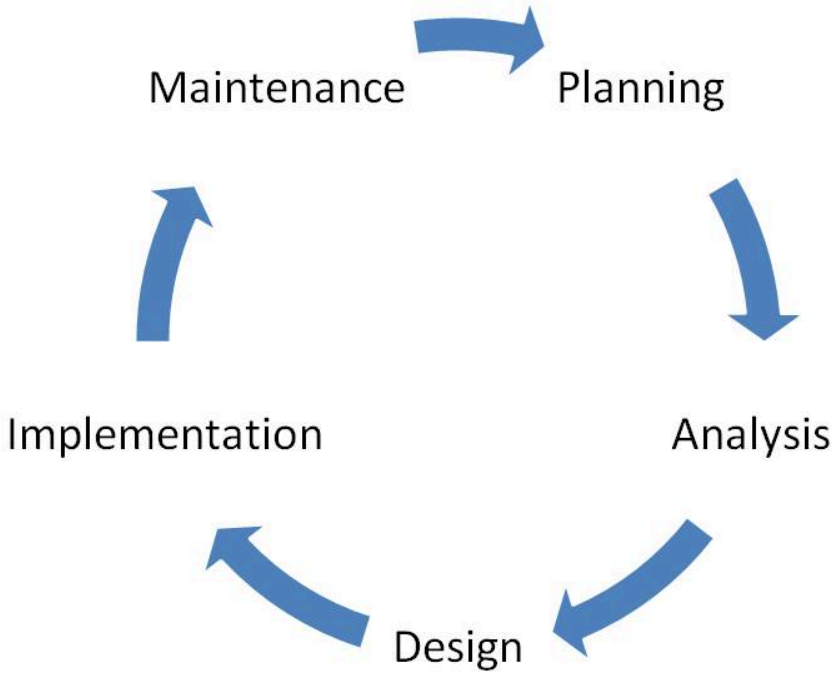
Discussion

The Systems Development Life Cycle is the big picture of creating an information system that handles a major task (referred to as an application). The **applications** usually consist of many programs. An example would be the Department of Defense supply system, the customer system used at your local bank, the repair parts inventory system used by car dealerships. There are thousands of applications that use an information system created just to help solve a business problem.

Another example of an information system would be the "101 Computer Games" software you might buy at any of several retail stores. This is an entertainment application, that is we are applying the computer to do a task (entertain you). The software actually consists of many different programs (checkers, chess, tic tac toe, etc.) that were most likely written by several different programmers.

Computer professionals that are in charge of creating applications often have the job title of **System Analyst**. The major steps in creating an application include the following and start at **Planning** step.

1. Wikipedia: Systems development life cycle



Systems Development Life Cycle

During the **Design** phase, the System Analyst will document the inputs, processing, and outputs of each program within the application. During the **Implementation** phase, programmers would be assigned to write the specific programs using a programming language decided by the System Analyst. Once the system of programs is tested the new application is installed for people to use. As time goes by, things change and a specific part or program might need repair. During the **Maintenance** phase, it goes through a mini planning, analysis, design, and implementation. The programs that need modification are identified and programmers change or repair those programs. After several years of use, the system usually becomes obsolete. At this point, a major revision of the application is done. Thus the cycle repeats itself.

Key Terms

applications
: An information system or collection of programs that handles a major task.

implementation
: The phase of a Systems Development Life Cycle where the programmers would be assigned to write specific programs.

life cycle
: Systems Development Life Cycle: Planning – Analysis – Design – Implementation – Maintenance

system analyst
: Computer professional in charge of creating applications.

References

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

Program Design

KENNETH LEROY BUSBEE

Overview

Program design consists of the steps a programmer should do before they start coding the program in a specific language. These steps when properly documented will make the completed program easier for other programmers to maintain in the future. There are three broad areas of activity:

- Understanding the Program
- Using Design Tools to Create a Model
- Develop Test Data

Understanding the Program

If you are working on a project as one of many programmers, the system analyst may have created a variety of documentation items that will help you understand what the program is to do. These could include screen layouts, narrative descriptions, documentation showing the processing steps, etc. If you are not on a project and you are creating a simple program you might be given only a simple description of the purpose of the program. Understanding the purpose of a program usually involves understanding its:

- Inputs
- Processing
- Outputs

This **IPO** approach works very well for beginning programmers. Sometimes, it might help to visualize the program running on the computer. You can imagine what the monitor will look like, what the user must enter on the keyboard and what processing or manipulations will be done.

Using Design Tools to Create a Model

At first, you will not need a hierarchy chart because your first programs will not be complex. But as they grow and become more complex, you will divide your program into several modules (or functions).

The first modeling tool you will usually learn is **pseudocode**. You will document the logic or algorithm of each function in your program. At first, you will have only one function, and thus your pseudocode will follow closely the IPO approach above.

There are several methods or tools for planning the logic of a program. They include: flowcharting, hierarchy or structure charts, pseudocode, HIPO, Nassi-Schneiderman charts, Warnier-Orr diagrams, etc. Programmers are expected to be able to understand and do flowcharting and pseudocode. These methods of developing the model of a program are usually taught in most computer courses. Several standards exist for flowcharting and pseudocode and most are very similar to each other. However, most companies have their own documentation standards and styles. Programmers are expected to be able to quickly adapt to any flowcharting or pseudocode

standards for the company at which they work. The other methods that are less universal require some training which is generally provided by the employer that chooses to use them.

Later in your programming career, you will learn about using application software that helps create an information system and/or programs. This type of software is called Computer-Aided Software Engineering (CASE).

Understanding the logic and planning the algorithm on paper before you start to code is a very important concept. Many students develop poor habits and skipping this step is one of them.

Develop Test Data

Test data consists of the programmer providing some input values and predicting the outputs. This can be quite easy for a simple program and the test data can be used to check the model to see if it produces the correct results.

Key Terms

IPO
: Inputs – Processing – Outputs

pseudocode
: English-like statements used to convey the steps of an algorithm or function.

test data
: Providing input values and predicting the outputs.

References

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

Program Quality

DAVE BRAUNSCHWEIG

Overview

Program quality describes fundamental properties of the program's source code and executable code, including reliability, robustness, usability, portability, maintainability, efficiency, and readability.

Discussion

Whatever the approach to development may be, the final program must satisfy some fundamental properties. The following properties are among the most important:

- **Reliability**: how often the results of a program are correct. This depends on the conceptual correctness of algorithms, and minimization of programming mistakes, such as mistakes in resource management (e.g., buffer overflows and race conditions) and logic errors (such as division by zero or off-by-one errors).
- **Robustness**: how well a program anticipates problems due to errors (not bugs). This includes situations such as incorrect, inappropriate or corrupt data, unavailability of needed resources such as memory, operating system services and network connections, user error, and unexpected power outages.
- **Usability**: the ergonomics of a program: the ease with which a person can use the program for its intended purpose or in some cases even unanticipated purposes. Such issues can make or break its success even regardless of other issues. This involves a wide range of textual, graphical and sometimes hardware elements that improve the clarity, intuitiveness, cohesiveness, and completeness of a program's user interface.
- **Portability**: the range of computer hardware and operating system platforms on which the source code of a program can be compiled/interpreted and run. This depends on differences in the programming facilities provided by the different platforms, including hardware and operating system resources, expected behavior of the hardware and operating system, and availability of platform specific compilers (and sometimes libraries) for the language of the source code.
- **Maintainability**: the ease with which a program can be modified by its present or future developers in order to make improvements or customizations, fix bugs and security holes, or adapt it to new environments. Good practices during initial development make the difference in this regard. This quality may not be directly apparent to the end user but it can significantly affect the fate of a program over the long term.
- **Efficiency**/performance: the measure of system resources a program consumes (processor time, memory space, slow devices such as disks, network bandwidth and to some extent even user interaction): the less, the better. This also includes careful management of resources, for example cleaning up temporary files and eliminating memory leaks.
- **Readability**: the ease with which a human reader can comprehend the purpose, control flow, and operation of source code. It affects the aspects of quality above, including portability, usability and most importantly maintainability. Readability is important because programmers spend the majority of their time reading, trying to understand and modifying existing source

code, rather than writing new source code. Unreadable code often leads to bugs, inefficiencies, and duplicated code.

Key Terms

efficiency
: The measure of system resources a program consumes.

maintainability
: The ease with which a program can be modified by its present or future developers.

portability
: The range of computer hardware and operating system platforms on which the source code of a program can be compiled/interpreted and run.

readability
: The ease with which a human reader can comprehend the purpose, control flow, and operation of source code.

reliability
: How often the results of a program are correct.

robustness
: How well a program anticipates problems due to errors.

usability
: The ease with which a person can use the program.

References

- Wikipedia: Computer programming

Pseudocode

KENNETH LEROY BUSBEE

Overview

Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm.[1]

Discussion

Pseudocode is one method of designing or planning a program. **Pseudo** means false, thus pseudocode means false code. A better translation would be the word fake or imitation. Pseudocode is fake (not the real thing). It looks like (imitates) real code but it is NOT real code. It uses English statements to describe what a program is to accomplish. It is fake because no compiler exists that will translate the pseudocode to any machine language. Pseudocode is used for documenting the program or module design (also known as the algorithm).

The following outline of a simple program illustrates pseudocode. We want to be able to enter the ages of two people and have the computer calculate their average age and display the answer.

Outline using Pseudocode

```
Input
    display a message asking the user to enter the first age
    get the first age from the keyboard
    display a message asking the user to enter the second age
    get the second age from the keyboard

Processing
    calculate the answer by adding the two ages together and dividing by tw

Output
    display the answer on the screen
    pause so the user can see the answer
```

After developing the program design, we use the pseudocode to write code in a language (like C++, Java, Python, etc.) where you must follow the rules of the language (syntax) in order to code the logic or algorithm presented in the pseudocode. Pseudocode usually does not include other items produced during programming design such as identifier lists for variables or test data.

There are other methods for planning and documenting the logic for a program. One method is HIPO. It stands for Hierarchy plus Input Process Output and was developed by IBM in the 1960s. It involved using a hierarchy (or structure) chart to show the relationship of the sub-routines (or

1. Wikipedia: Pseudocode

functions) in a program. Each sub-routine had an IPO piece. Since the above problem/task was simple, we did not need to use multiple sub-routines, thus we did not produce a hierarchy chart. We did incorporate the IPO part of the concept for the pseudocode outline.

Key Terms

pseudo
Means false and includes the concepts of fake or imitation.

References

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

Flowcharts

KENNETH LEROY BUSBEE

Overview

A **flowchart** is a type of diagram that represents an algorithm, workflow or process. The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.[1]

Discussion

Common flowcharting symbols and examples follow. When first reading this section, focus on the simple symbols and examples. Return to this section in later chapters to review the advanced symbols and examples.

Simple Flowcharting Symbols

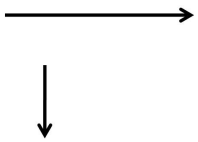
Terminal

The rounded rectangles, or terminal points, indicate the flowchart's starting and ending points.

Flow Lines

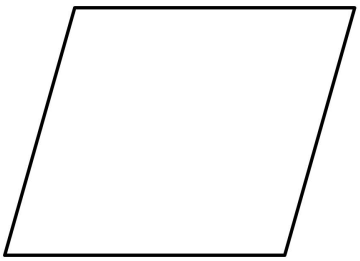
Note: The default flow is left to right and top to bottom (the same way you read English). To save time arrowheads are often only drawn when the flow lines go contrary the normal.

1. Wikipedia: Flowchart



Input/Output

The parallelograms designate input or output operations.

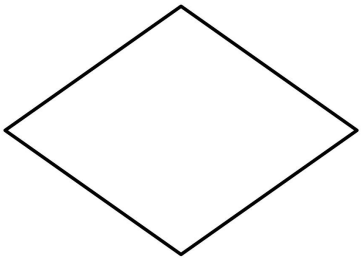


Process

The rectangle depicts a process such as a mathematical computation, or a variable assignment.

Decision

The diamond is used to represent the true/false statement being tested in a decision symbol.

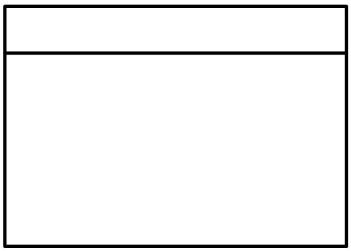


Advanced Flowcharting Symbols

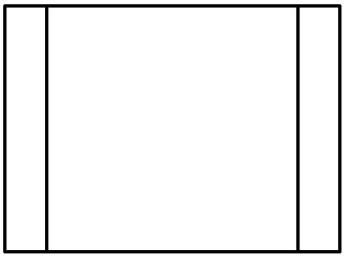
Module Call

A program module is represented in a flowchart by rectangle with some lines to distinguish it from process symbol. Often programmers will make a distinction between program control and specific task modules as shown below.

Local module: usually a program control function.



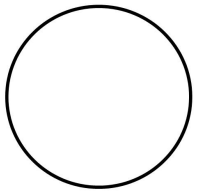
Library module: usually a specific task function.



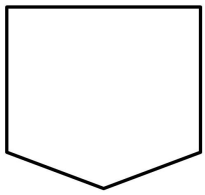
Connectors

Sometimes a flowchart is broken into two or more smaller flowcharts. This is usually done when a flowchart does not fit on a single page, or must be divided into sections. A connector symbol, which is a small circle with a letter or number inside it, allows you to connect two flowcharts on the same page. A connector symbol that looks like a pocket on a shirt, allows you to connect to a flowchart on a different page.

On-Page Connector



Off-Page Connector



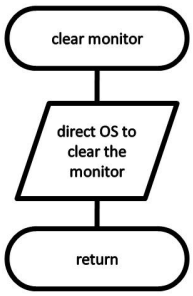
Simple Examples

We will demonstrate various flowcharting items by showing the flowchart for some pseudocode.

Functions

pseudocode: Function with no parameter passing

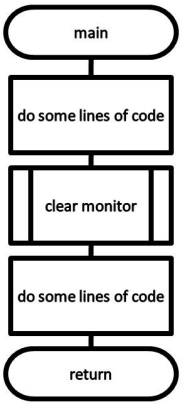
```
Function clear monitor
    Pass In: nothing
    Direct the operating system to clear the monitor
    Pass Out: nothing
End function
```



Function clear monitor

pseudocode: Function main calling the clear monitor function

```
Function main
    Pass In: nothing
    Doing some lines of code
    Call: clear monitor
    Doing some lines of code
    Pass Out: value zero to the operating system
End function
```



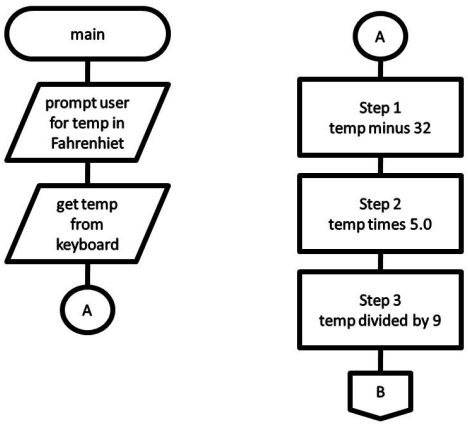
Function main

Sequence Control Structures

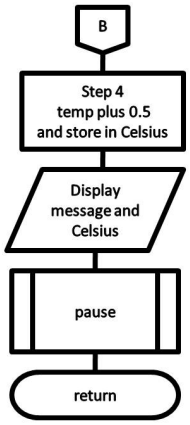
The next item is pseudocode for a simple temperature conversion program. This demonstrates the use of both the on-page and off-page connectors. It also illustrates the sequence control structure where nothing unusual happens. Just do one instruction after another in the sequence listed.

pseudocode: Sequence control structure

```
Filename: Solution_Lab_04_Pseudocode.txt
Purpose:  Convert Temperature from Fahrenheit to Celsius
Author:   Ken Busbee; © 2008 Kenneth Leroy Busbee
Date:     Dec 24, 2008

Pseudocode = IPO Outline

input
    display a message asking user for the temperature in Fahrenheit
    get the temperature from the keyboard

 processing
    calculate the Celsius by subtracting 32 from the Fahrenheit
    temperature then multiply the result by 5 then
    divide the result by 9. Round up or down to the whole numbe
    HINT: Use 32.0 when subtracting to ensure floating-point ac

 output
    display the celsius with an appropriate message
    pause so the user can see the answer
```



Sequence control structure



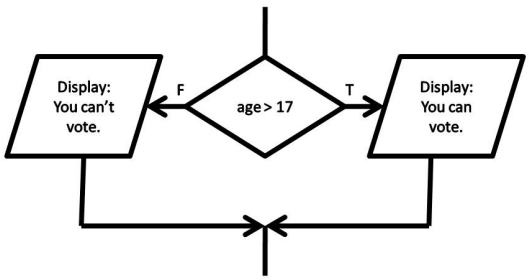
Sequence control structured continued

Advanced Examples

Selection Control Structures

pseudocode: If then Else

```
If age > 17
    Display a message indicating you can vote.
Else
    Display a message indicating you can't vote.
Endif
```

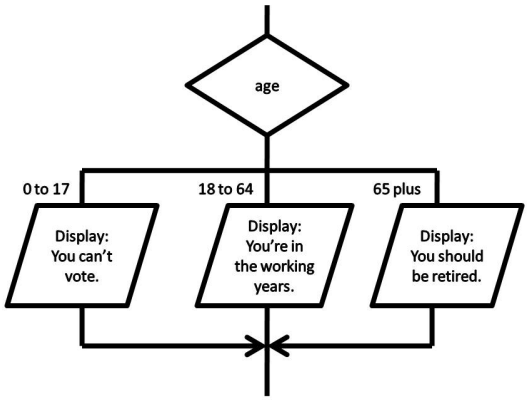


If then Else control structure

pseudocode: Case

```
Case of age
```

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

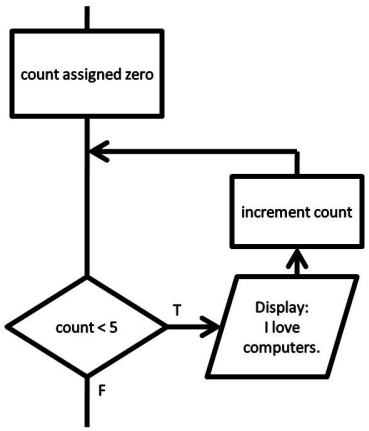


Case control structure

Iteration (Repetition) Control Structures

pseudocode: While

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

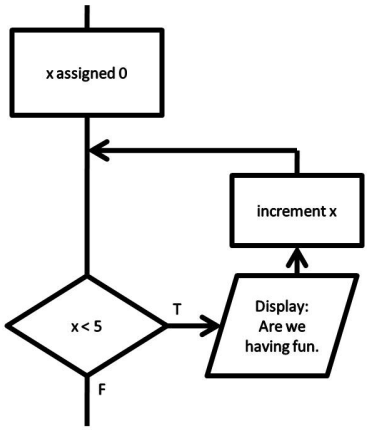


While control structure

pseudocode: For

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

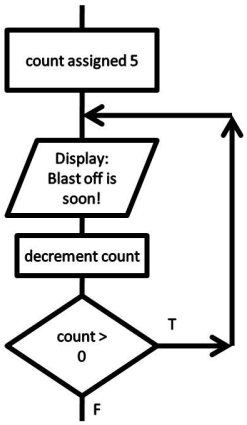
The for loop does not have a standard flowcharting method and you will find it done in different ways. The for loop as a counting loop can be flowcharted similar to the while loop as a counting loop.



For control structure

pseudocode: Do While

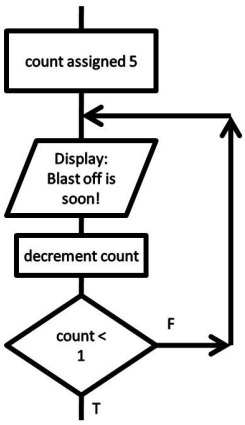
```
count assigned five
Do
    Display "Blast off is soon!"
    Decrement count
While count > zero
```



Do While control structure

pseudocode: Repeat Until

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



Repeat Until control structure

Key Terms

decision symbol
A diamond used in flowcharting for asking a question and making a decision.
flow lines
Lines (sometimes with arrows) that connect the various flowcharting symbols.
flowcharting
A programming design tool that uses graphical elements to visually depict the flow of logic within a function.
input/output symbol
A parallelogram used in flowcharting for input/output interactions.
process symbol
A rectangle used in flowcharting for normal processes such as assignment.

References

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

Software Testing

KENNETH LEROY BUSBEE

Overview

Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:[1]

- meets the requirements that guided its design and development
- responds correctly to all kinds of inputs
- performs its functions within an acceptable time
- is sufficiently usable
- can be installed and run in its intended environments
- achieves the general result its stakeholders desire

Discussion

Test data consists of the user providing some input values and predicting the outputs. This can be quite easy for a simple program and the test data can be used twice.

1. to check the model to see if it produces the correct results (**model checking**)
2. to check the coded program to see if it produces the correct results (**code checking**)

Test data is developed by using the algorithm of the program. This algorithm is usually documented during the program design with either flowcharting or pseudocode. Here is the pseudocode in outline form describing the inputs, processing, and outputs for a program used to calculate gross pay for hourly work.

Pseudocode using an IPO Outline for Calculating Gross Pay

```
Input
    display a message asking user for their hours worked
    get the hours from the keyboard
    display a message asking user for their pay rate
    get the rate from the keyboard

Processing
    calculate the gross pay by:
        multiplying the hours worked by the hourly rate
```

1. Wikipedia: Software testing

```
Output
    display the gross pay on the monitor
    pause so the user can see the answer
```

Creating Test Data and Model Checking

Test data is used to verify that the inputs, processing, and outputs are working correctly. As test data is initially developed it can verify that the documented algorithm (pseudocode in the example we are doing) is correct. It helps us understand and even visualize the inputs, processing, and outputs of the program.

Inputs: I worked 37.5 hours this week and my hourly rate is $15.50 per hour. We should verify that the pseudocode is prompting the user for this data.

Processing: Using my solar powered handheld calculator, I can calculate the gross pay would be: 37.5 * 15.50 or $581.25. We should verify that the pseudocode is performing the correct calculations.

Output: Only the significant information (total gross pay) is displayed for the user to see. We should verify that the appropriate information is being displayed.

Testing the Coded Program – Code Checking

The test data can be developed and used to test the algorithm that is documented (in our case our pseudocode) during the program design phase. Once the program is code with compiler and linker errors resolved, the programmer gets to play user and should test the program using the test data developed. When you run your program, how will you know that it is working properly? Did you properly plan your logic to accomplish your purpose? Even if your plan was correct, did it get converted correctly (coded) into the chosen programming language? The answer (or solution) to all of these questions is our test data.

By developing test data we are predicting what the results should be, thus we can verify that our program is working properly. When we run the program we would enter the input values used in our test data. Hopefully, the program will output the predicted values. If not then our problem could be any of the following:

1. The plan (IPO outline or another item) could be wrong
2. The conversion of the plan to code might be wrong
3. The test data results were calculated wrong

Resolving problems of this nature can be the most difficult problems a programmer encounters. You must review each of the above to determine where the error is lies. Fix the error and re-test your program.

Key Terms

code checking
Using test data to check the coded program in a specific language (like C++).

model checking
Using test data to check the design model (usually done in pseudocode).

References

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

Integrated Development Environment

KENNETH LEROY BUSBEE

Overview

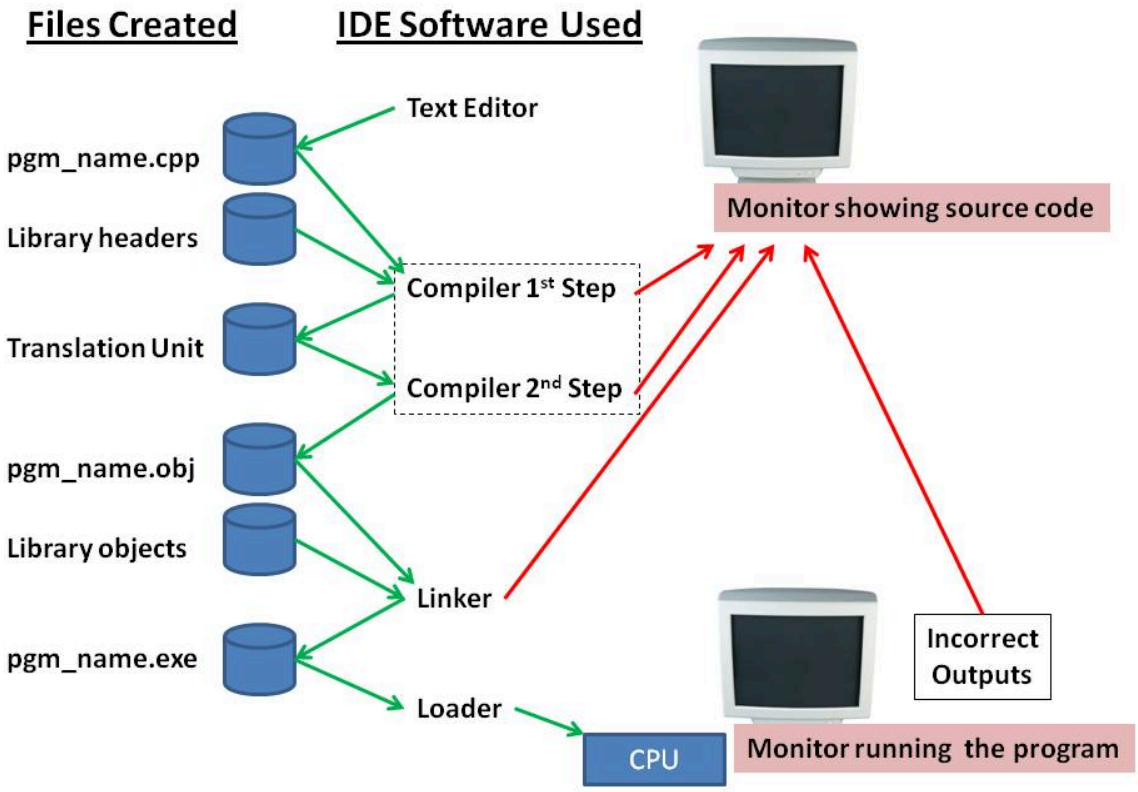
An **integrated development environment (IDE)** is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools, and a debugger. Most modern IDEs have intelligent code completion. Some IDEs contain a compiler, interpreter, or both. The boundary between an integrated development environment and other parts of the broader software development environment is not well-defined. Sometimes a version control system, or various tools to simplify the construction of a graphical user interface (GUI), are integrated. Many modern IDEs also have a class browser, an object browser, and a class hierarchy diagram, for use in object-oriented software development.[1]

Discussion

High-level language programs are usually written (coded) as ASCII text into a source code file. A unique file extension (Examples: .asm .c .cpp .java .js .py) is used to identify it as a source code file. As you might guess for our examples – Assembly, "C", "C++", Java, JavaScript, and Python, however, they are just ASCII text files (other text files usually use the extension of .txt). The source code produced by the programmer must be converted to an executable machine code file specifically for the computer's CPU (usually an Intel or Intel-compatible CPU within today's world of computers). There are several steps in getting a program from its source code stage to running the program on your computer. Historically, we had to use several software programs (a text editor, a compiler, a linker, and operating system commands) to make the conversion and run our program. However, today all those software programs with their associated tasks have been **integrated** into one program. However, this one program is really many software items that create an **environment** used by programmers to **develop** software. Thus the name: Integrated Development Environment or IDE.

Programs written in a high-level language are either directly executed by some kind of interpreter or converted into machine code by a compiler (and assembler and linker) for the CPU to execute. JavaScript, Perl, Python, and Ruby are examples of interpreted programming languages. C, C++, C#, Java, and Swift are examples of compiled programming languages.[2] The following figure shows the progression of activity in an IDE as a programmer enters the source code and then directs the IDE to compile and run the program.

1. Wikipedia: Integrated development environment
2. Wikipedia: Interpreter (computing)



Integrated Development Environment or IDE

Upon starting the IDE software the programmer usually indicates the file he or she wants to open for editing as source code. As they make changes they might either do a “save as” or “save”. When they have finished entering the source code, they usually direct the IDE to “compile & run” the program. The IDE does the following steps:

1. If there are any unsaved changes to the source code file it has the **test editor** save the changes.
2. The **compiler** opens the source code file and does its **first step** which is executing the **pre-processor** compiler directives and other steps needed to get the file ready for the second step. The #include will insert header files into the code at this point. If it encounters an error, it stops the process and returns the user to the source code file within the text editor with an error message. If no problems encountered it saves the source code to a temporary file called a translation unit.
3. The **compiler** opens the translation unit file and does its **second step** which is **converting** the programming language code to machine instructions for the CPU, a data area, and a list of items to be resolved by the linker. Any problems encountered (usually a syntax or violation of the programming language rules) stops the process and returns the user to the source code file within the text editor with an error message. If no problems encountered it saves the machine instructions, data area, and linker resolution list as an object file.
4. The **linker** opens the program object file and links it with the library object files as needed. Unless all linker items are resolved, the process stops and returns the user to the source code file within the text editor with an error message. If no problems encountered it saves the linked objects as an executable file.

5. The IDE directs the operating system's program called the **loader** to load the executable file into the computer's memory and have the Central Processing Unit (CPU) start processing the instructions. As the user interacts with the program, entering test data, he or she might discover that the outputs are not correct. These types of errors are called logic errors and would require the user to return to the source code to change the algorithm.

Resolving Errors

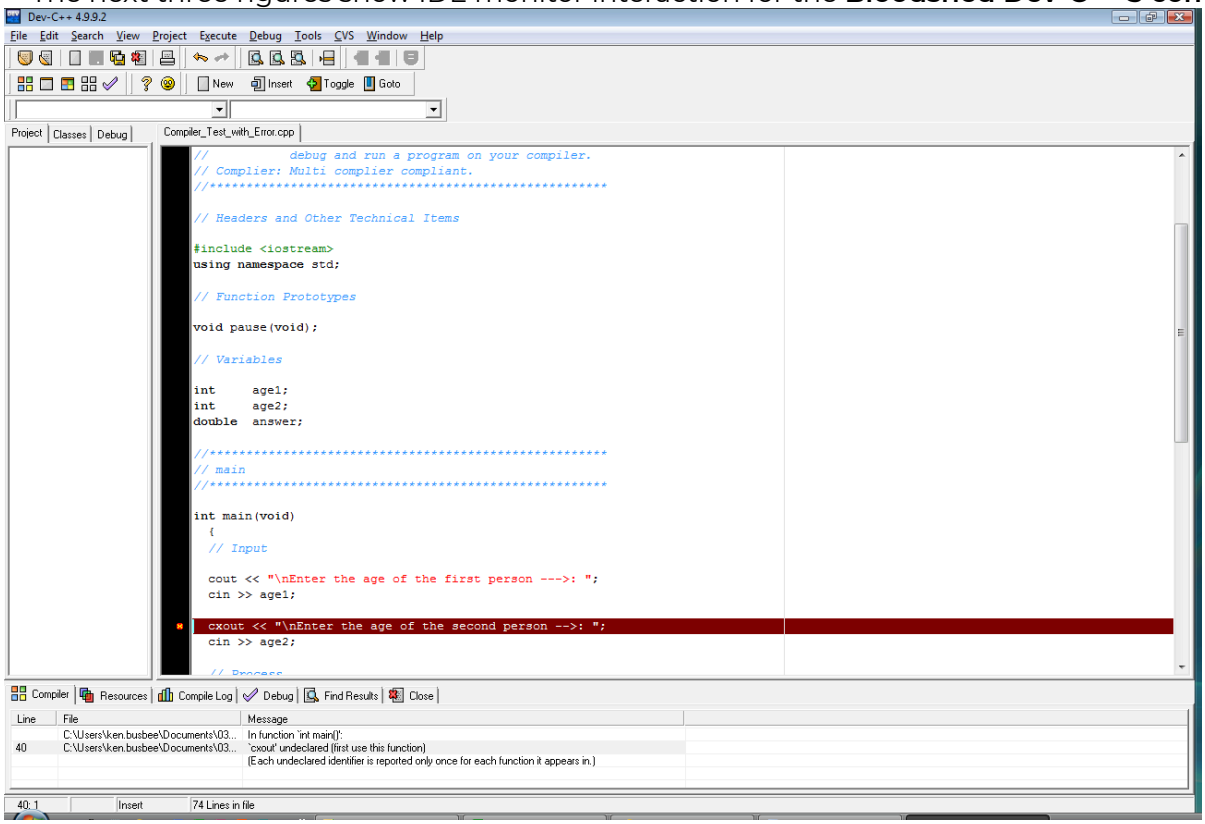
Despite our best efforts at becoming perfect programmers, we will create errors. Solving these errors is known as **debugging** your program. The three types of errors in the order that they occur are:

1. Compiler
2. Linker
3. Logic

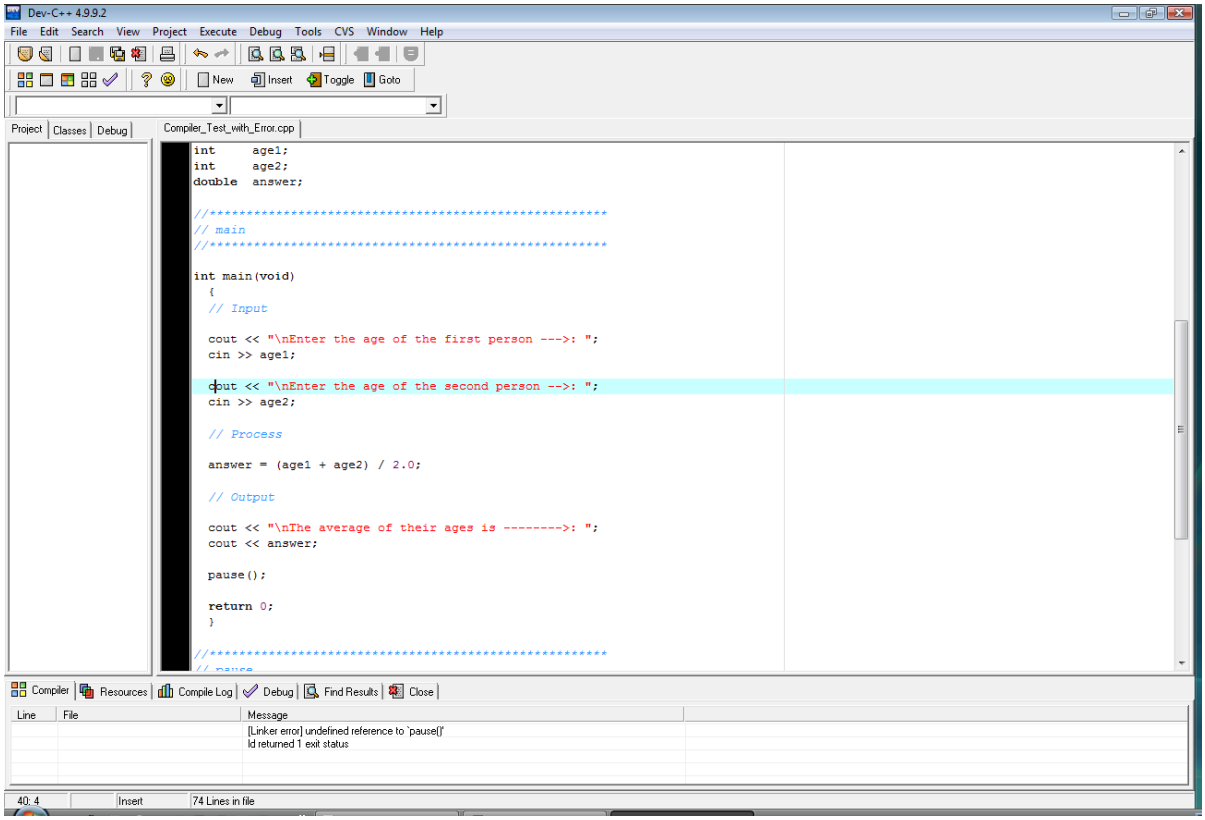
There are two types of compiler errors; pre-processor (1st step) and conversion (2nd step). A review of Figure 1 above shows the four arrows returning to the source code so that the programmer can correct the mistake.

During the conversion (2nd step) the compiler might give a **warning** message which in some cases may not be a problem to worry about. For example: Data type demotion may be exactly what you want your program to do, but most compilers give a warning message. Warnings don't stop the compiling process but as their name implies, they should be reviewed.

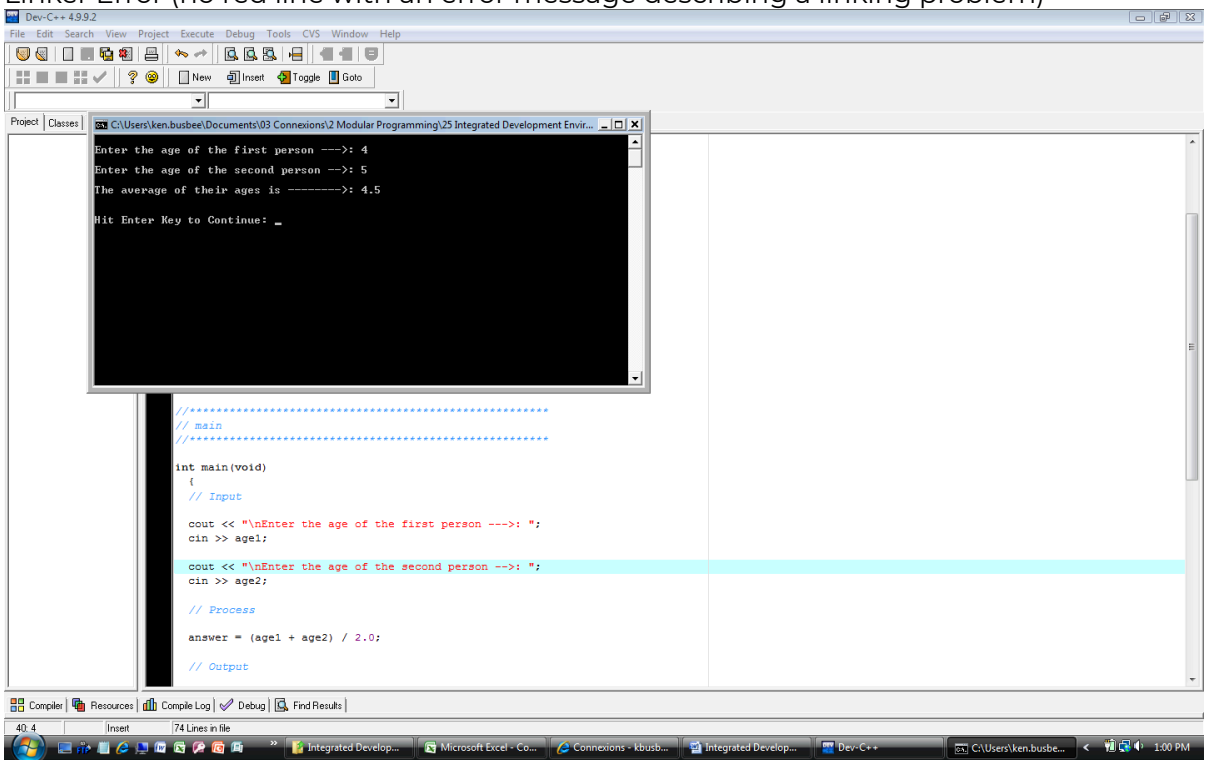
The next three figures show IDE monitor interaction for the **Bloodshed Dev-C++ 5 compiler/IDE**.



Compiler Error (the red line is where the compiler stopped)



Linker Error (no red line with an error message describing a linking problem)



Logic Error (from the output within the “Black Box” area)

Key Terms

compiler
Converts source code to object code.
debugging
The process of removing errors from a program. 1) compiler 2) linker 3) logic
linker
Connects or links object files into an executable file.
loader
Part of the operating system that loads executable files into memory and directs the CPU to start running the program.
pre-processor
The first step the compiler does in converting source code to object code.
text editor
A software program for creating and editing ASCII text files.
warning
A compiler alert that there might be a problem.

References

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

Version Control

DAVE BRAUNSCHWEIG

Overview

Version control, also known as revision control or source control, is the management of changes to documents, computer programs, large websites, and other collections of information. Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged.[1]

Version control systems (VCS) most commonly run as stand-alone applications, but may also be embedded in various types of software, including integrated development environments (IDEs).

Discussion

Version control implements a systematic approach to recording and managing changes in files. At its simplest, version control involves taking 'snapshots' of your file at different stages. This snapshot records information about when the snapshot was made, and also about what changes occurred between different snapshots. This allows you to 'rewind' your file to an older version. From this basic aim of version control, a range of other possibilities is made available.[2]

Version control allows you to:[3]

- Track developments and changes in your files
- Record the changes you made to your file in a way that you will be able to understand later
- Experiment with different versions of a file while maintaining the original version
- 'Merge' two versions of a file and manage conflicts between versions
- Revert changes, moving 'backward' through your history to previous versions of your file

Version control is particularly useful for facilitating collaboration. One of the original motivations behind version control systems was to allow different people to work on large projects together. Using version control to collaborate allows for a greater deal of flexibility and control than many other solutions. As an example, it would be possible for two people to work on a file at the same time and then merge these together. If there were 'conflicts' between the two versions, the version control system would allow you to see these conflicts and make an active decision about how to 'merge' these different versions into a new 'third' document. With this approach you would also retain a 'history' of the previous version should you wish to revert back to one of these later on.[4]

Popular version control systems include:[5]

- Git

1. Wikipedia: Version control
2. Programming Historian: An Introduction to Version Control Using GitHub Desktop
3. Programming Historian: An Introduction to Version Control Using GitHub Desktop
4. Programming Historian: An Introduction to Version Control Using GitHub Desktop
5. G2Crowd: Best Version Control Systems

- Helix VCS
- Microsoft Team Foundation Server
- Subversion

The following focuses on using the Git version control system.

Git

Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development, but it can be used to keep track of changes in any set of files. Git was created by Linus Torvalds in 2005 for development of the Linux kernel and is free and open source software.[6]

Free public and private git repositories are available from:

- Bitbucket
- GitHub

Cloning an existing repository requires only a URL to the repository and the following git command:

- `git clone <url>`

Once cloned, repositories are synchronized by pushing and pulling changes. If the original source repository has been modified, the following git command is used to pull those changes to the local repository:

- `git pull`

Local changes must be added and committed, and then pushed to the remote repository. *Note the period (dot) at the end of the first command.*

- `git add .`
- `git commit -m "reason for commit"`
- `git push`

If there are conflicts between the local and remote repositories, the changes should be merged and then pushed. If necessary, local changes may be forced upon the remote server using:

- `git push --force`

Key Terms

branch

A separate working copy of files under version control which may be developed independently

6. Wikipedia: Git

from the origin.

clone
: Create a new repository containing the revisions from another repository.

commit
: To write or merge the changes made in the working copy back to the repository.

merge
: An operation in which two sets of changes are applied to a file or set of files.

push
: Copy revisions from the current repository to a remote repository.

pull
: Copy revisions from a remote repository to the current repository.

version control
: The management of changes to documents, computer programs, large websites, and other collections of information.

References

- Programming Historian: An Introduction to Version Control Using GitHub Desktop

Input and Output

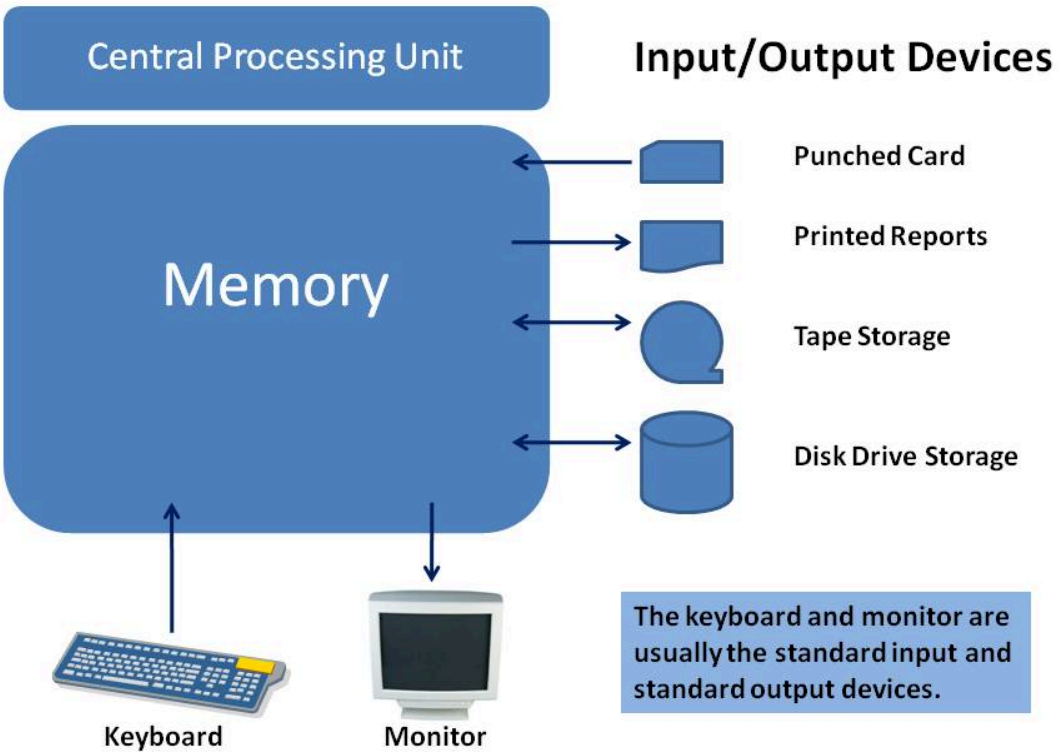
KENNETH LEROY BUSBEE

Overview

Input and output, or I/O is the communication between an information processing system, such as a computer, and the outside world, possibly a human or another information processing system. Inputs are the signals or data received by the system and outputs are the signals or data sent from it.[1]

Discussion

Every task we have the computer do happens inside the central processing unit (CPU) and the associated memory. Once our program is loaded into memory and the operating system directs the CPU to start executing our programming statements the computer looks like this:



1. Wikipedia: Input/output

CPU – Memory – Input/Output Devices

Our program now loaded into memory has basically two areas:

- Machine instructions – our instructions for what we want done
- Data storage – our variables that we using in our program

Often our program contains instructions to interact with the input/output devices. We need to move data into (read) and/or out of (write) the memory data area. A **device** is a piece of equipment that is electronically connected to the memory so that data can be transferred between the memory and the device. Historically this was done with punched cards and printouts. Tape drives were used for electronic storage. With time we migrated to using disk drives for storage with keyboards and monitors (with monitor output called soft copy) replacing punch cards and printouts (called hard copy).

Most computer operating systems and by extension programming languages have identified the keyboard as the **standard input device** and the monitor as the **standard output device**. Often the keyboard and monitor are treated as the default device when no other specific device is indicated.

Key Terms

device
: A piece of equipment that is electronically connected to the memory so that data can be transferred between the memory and the device.

escape code
: A code directing an output device to do something.

extraction
: Aka reading or getting data from an input device.

insertion
: Aka writing or sending data to an output device.

standard input
: The keyboard.

standard output
: The monitor.

References

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

Hello World

DAVE BRAUNSCHWEIG

`Hello world!` Overview

A “**Hello, world**!” program is a computer program that outputs or displays “Hello, world!” to a user. Being a very simple program in most programming languages, it is often used to illustrate the basic syntax of a programming language for a working program, and as such is often the very first program people write.[1]

Discussion

A “Hello, world!” program is traditionally used to introduce novice programmers to a programming language. “Hello, world!” is also traditionally used in a sanity test to make sure that a computer language is correctly installed, and that the operator understands how to use it.[2]

The tradition of using the phrase “Hello, world!” as a test message was influenced by an example program in the seminal book *The C Programming Language*. The example program from that book prints “hello, world” (without capital letters or exclamation mark), and was inherited from a 1974 Bell Laboratories internal memorandum by Brian Kernighan.[3]

In addition to displaying “Hello, world!”, a “Hello, world!” program might include comments. A **comment** is a programmer-readable explanation or annotation in the source code of a computer program. They are added with the purpose of making the source code easier for humans to understand, and are generally ignored by compilers and interpreters. The syntax of comments in various programming languages varies considerably.[4]

Pseudocode

```
Function Main
    ... This program displays "Hello world!"
    Output "Hello world!"
End
```

1. Wikipedia: "Hello, World!" program
2. Wikipedia: "Hello, World!" program
3. Wikipedia: "Hello, World!" program
4. Wikipedia: Comment (computer programming)

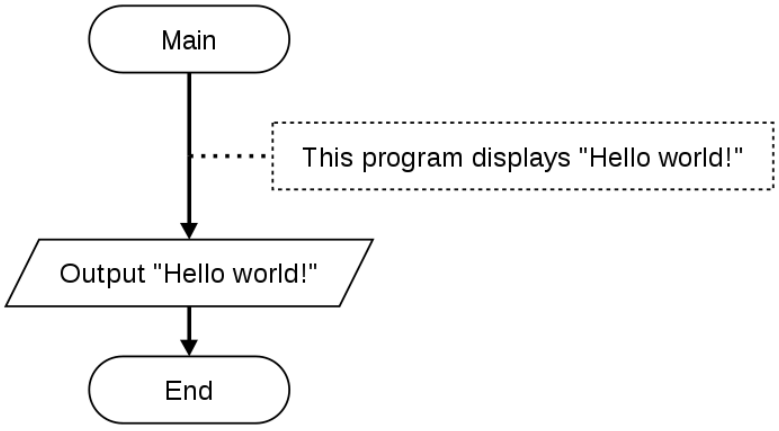
Output

```
Hello world!
```

Each code element represents:[5]

- `Function Main` begins the main function
- `...` begins a comment
- `Output` indicates the following value(s) will be displayed or printed
- `"Hello world!"` is the literal string to be displayed
- `End` ends a block of code

Flowchart



Examples

The following pages provide examples of “Hello, world!” programs in different programming languages. Each page includes an explanation of the code elements that comprise the program and links to IDEs you may use to test the program.

Key Terms

comment
A programmer-readable explanation or annotation in the source code of a computer program.

5. Wikibooks: Programming Fundamentals/Hello World

References

- Wikiversity: Computer Programming
- Flowgorithm – Flowchart Programming Language

C++ Examples

DAVE BRAUNSCHWEIG

Overview

C++ is a general-purpose programming language. It has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation. C++ was developed by Bjarne Stroustrup at Bell Labs starting in 1979 as an extension of the C language. The C++ programming language was initially standardized in 1998.[1]

C++ is one of the most popular current programming languages[2] and is often used in computer science courses.

Example

Hello World

```
// This program displays "Hello world!"
//
// References:
// http://www.cplusplus.com/doc/tutorial/program_structure/

#include <iostream>

int main()
{
    std::cout << "Hello world!";
}
```

1. Wikipedia: C++
2. TIOBE: Index

Output

```
Hello world!
```

Discussion

Each code element represents:[3]

- `//` begins a comment
- `#include <iostream>` includes standard input and output streams
- `int main()` begins the main function, which returns an integer value
- `{` begins a block of code
- `std::cout` is standard output
- `<<` directs the next element to standard output
- `"Hello world!"` is the literal string to be displayed
- `;` ends each line of C++ code
- `}` ends a block of code

C++ IDEs

There are many free cloud-based and local IDEs available to begin coding in C++. Check with your instructor or do your own research for recommendations.

Cloud-Based IDEs

- CodeChef
- GDB Online
- Ideone
- paiza.IO
- PythonTutor
- repl.it
- TutorialsPoint

Local IDEs

- Code::Blocks
- Dev-C++
- Microsoft Visual Studio

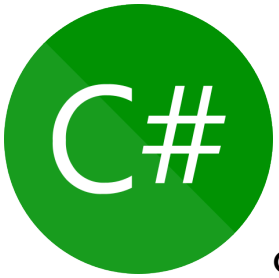
3. Wikibooks: Programming Fundamentals/Hello World

References

- Wikiversity: Computer Programming

C# Examples

DAVE BRAUNSCHWEIG



Overview

C# is a general-purpose, object-oriented programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed around 2000 by Microsoft within its .NET initiative and later approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure.[1]

C# is one of the most popular current programming languages[2], is the primary language for Windows application development and is often used in computer science and gaming courses.

Example

Hello World

```
// This program displays "Hello world!"
//
// References:
// https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-

public class Hello
{
    public static void Main()
    {
        System.Console.WriteLine("Hello world!");
    }
}
```

1. Wikipedia: C Sharp (programming language)
2. TIOBE: Index

Output

```
Hello world!
```

Discussion

Each code element represents:[3]

- `//` begins a comment
- `public class Hello` begins the Hello World program
- `{` begins a block of code
- `public static void Main()` begins the main function
- `System.Console.WriteLine()` calls the standard output write line function
- `"Hello world!"` is the literal string to be displayed
- `;` ends each line of C# code
- `}` ends a block of code

C# IDEs

There are many free cloud-based and local IDEs available to begin coding in C#. Check with your instructor or do your own research for recommendations.

Cloud-Based IDEs

- CodeChef
- C# Pad
- .NET Fiddle
- Ideone
- paiza.IO
- Rextester
- repl.it
- TutorialsPoint

Local IDEs

- Microsoft Visual Studio
- Visual Studio Code

3. Wikibooks: Programming Fundamentals/Hello World

References

- Wikiversity: Computer Programming

Java Examples

DAVE BRAUNSCHWEIG

Overview

Java is a general-purpose computer-programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java was originally developed by James Gosling at Sun Microsystems and released in 1995.[1]

Java is one of the most popular current programming languages[2] and is often used in computer science courses.

Example

Hello World

```
// This program displays "Hello world!"
//
// References:
// https://introcs.cs.princeton.edu/java/11hello/HelloWorld.java.html

class Main {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

1. Wikipedia: Java (programming language)
2. TIOBE: Index

Output

```
Hello world!
```

Discussion

Each code element represents:[3]

- `//` begins a comment
- `class hello` begins the Hello World program
- `{` begins a block of code
- `public static void main(String[] args)` begins the main function
- `System.out.println()` calls the standard output print line function
- `"Hello world!"` is the literal string to be displayed
- `;` ends each line of Java code
- `}` ends a block of code

Java IDEs

There are many free cloud-based and local IDEs available to begin coding in Java. Check with your instructor or do your own research for recommendations.

Cloud-Based IDEs

- CodeChef
- GDB Online
- Ideone
- paiza.IO
- PythonTutor
- repl.it
- TutorialsPoint

Local IDEs

- BlueJ
- jEdit
- jGRASP

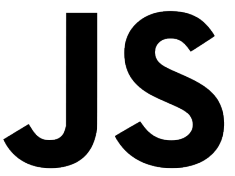
3. Wikibooks: Programming Fundamentals/Hello World

References

- Wikiversity: Computer Programming

JavaScript Examples

DAVE BRAUNSCHWEIG



Overview

JavaScript, often abbreviated as JS, is a high-level, interpreted programming language. Alongside HTML and CSS, JavaScript is one of the three core technologies of the World Wide Web. JavaScript enables interactive web pages and therefore is an essential part of web applications. The vast majority of websites use it, and all major web browsers have a dedicated JavaScript engine to execute it.[1]

JavaScript is one of the most popular current programming languages[2], and is the primary programming language for front-end web development. JavaScript has been implemented in multiple platforms with different I/O commands. Several examples follow.

Example

Hello World – Console Log

```
// This script displays "Hello world!".
//
// References:
// https://www.digitalocean.com/community/tutorials/how-to-write-your-first

console.log("Hello world!")
```

Output

```
Hello world!
```

1. Wikipedia: JavaScript
2. TIOBE: Index

Discussion

Each code element represents:

- `//` begins a comment
- `console.log()` writes to the JavaScript console output log
- `"Hello world!"` is the literal string to be displayed

Hello World – Window Alert

```
// This script displays "Hello world!".
//
// References:
// https://www.digitalocean.com/community/tutorials/how-to-write-your-first

alert("Hello world!")
```

Output

```
Hello world!
```

Discussion

Each code element represents:

- `//` begins a comment
- `alert()` calls the window alert function to display a message
- `"Hello world!"` is the literal string to be displayed

Hello World – Document Write

```
// This script displays "Hello world!".
//
// References:
// https://www.w3schools.com/jsref/met_doc_write.asp
document.write("Hello world!")
```

Output

```
Hello world!
```

Discussion

Each code element represents:

- `//` begins a comment
- `document.write()` writes output to the current document
- `"Hello world!"` is the literal string to be displayed

JavaScript IDEs

There are many free cloud-based and local IDEs available to begin coding in JavaScript. Check with your instructor or do your own research for recommendations.

Cloud-Based IDEs

- Chapman.edu: Online JavaScript Interpreter
- CodeChef
- GDB Online
- Ideone
- paiza.IO
- PythonTutor
- repl.it

Local IDEs

- Brackets
- Visual Studio Code

References

- Wikiversity: Computer Programming

Python Examples

DAVE BRAUNSCHWEIG

Overview

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.[1]

Python is one of the most popular current programming languages[2], is frequently recommended as a first programming language, and often used in information systems and data science courses.

Example

Hello World

```
# This program displays "Hello world!"
#
# References:
# https://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python_3/He

print("Hello world!")
```

Output

```
Hello world!
```

Discussion

Each code element represents:[3]

1. Wikipedia: Python (programming language)
2. TIOBE: Index
3. Wikibooks: Programming Fundamentals/Hello World

- `#` begins a comment
- `print()` calls the print function
- `"Hello world!"` is the literal string to be displayed

Python IDEs

There are many free cloud-based and local IDEs available to begin coding in Python. Check with your instructor or do your own research for recommendations.

Cloud-Based IDEs

- CodeChef
- GDB Online
- Ideone
- paiza.IO
- Python Fiddle
- PythonTutor
- repl.it
- TutorialsPoint

Local IDEs

- IDLE
- Thonny

References

- Wikiversity: Computer Programming

Swift Examples

DAVE BRAUNSCHWEIG

Swift is a general-purpose, multi-paradigm, compiled programming language developed by Apple Inc. for iOS, macOS, watchOS, tvOS, and Linux. Apple intended Swift to support many core concepts associated with Objective-C, but in a “safer” way, making it easier to catch software bugs. Swift was introduced in 2014.[1]

Swift is a popular programming language for the Apple platforms it supports, but it lacks support for Microsoft Windows environments.[2]

Example

Hello World

```
// This program displays "Hello world!"
//
// References:
//     https://developer.apple.com/library/content/documentation/Swift/Conc

print("Hello world!")
```

Output

```
Hello world!
```

Discussion

Each code element represents:[3]

1. Swift (programming language)
2. TIOBE: Index
3. Wikibooks: Programming Fundamentals/Hello World

- `//` begins a comment
- `print()` calls the print function
- `"Hello world!"` is the literal string to be displayed

Swift IDEs

There are several free cloud-based and local IDEs available to begin coding in Swift. Check with your instructor or do your own research for recommendations.

Cloud-Based IDEs

- GDB Online
- IBM Swift Sandbox
- Ideone
- iSwift
- paiza.IO
- repl.it

Local IDEs

- AppCode
- Atom
- Xcode

References

- Wikiversity: Computer Programming

Practice: Introduction to Programming

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Review Questions

True / False:

1. Beginning programmers participate in all phases of the Systems Development Life Cycle.
2. Coding the program in a language like C++ is the first task of planning. You plan as you code.
3. Pseudocode is the only commonly used planning tool.
4. Pseudocode has a strict set of rules and is the same everywhere in the computer programming industry.
5. Test data is developed for testing the program once it is code into a language like C++.
6. The word pseudo means false and includes the concepts of fake or imitation.
7. Many programmers pick up the bad habit of not completing the planning step before starting to code the program.
8. IDE means Integer Division Expression.
9. Most modern compilers are really an IDE type of software, not just a compiler.
10. Programming errors are extremely easy to understand and fix.

Answers:

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

Short Answer:

1. List the steps of the Systems Development Life Cycle and indicate which step you are likely to work in as a new computer professional.
2. List and describe what might cause the four (4) types of errors encountered in a program using a compiler and an Integrated Development Environment software product.

Activities

Pseudocode and Flowcharts

The following activities focus on software planning and testing using pseudocode and / or flowcharts.

1. Search the Internet for pseudocode for making a peanut butter and jelly sandwich. Based on the examples you find, create pseudocode to make your own favorite sandwich or other non-prepackaged meal. Note: Because peanut butter and jelly sandwich examples are already available, you must select something else for your pseudocode. Test your pseudocode by reading the instructions out loud as someone else follows your directions.
2. Search the Internet for a flowchart for making a peanut butter and jelly sandwich. Use a free online or downloadable flowchart tool to create a flowchart that describes how to make your favorite sandwich or other non-prepackaged meal. Note: Because peanut butter and jelly sandwich examples are already available, you must select something else for your flowchart. Test your flowchart by reading the instructions out loud while someone else follows your directions.
3. Create pseudocode or a flowchart for a program that would interact with bank customers and help them determine the value of a bag or jar of coins brought in for deposit. Include counts for pennies, nickels, dimes and quarters and calculate the total value of all of the coins deposited. Test your program by having someone else follow the instructions and guide them as they use your program.
4. Create pseudocode or a flowchart for a program that allows the user to enter gallons of gas and converts it to liters (metric system). NOTE: One US gallon equals 3.7854 liters. Test your program by having someone else follow the instructions and guide them as they use your program.
5. A major restaurant sends a chef to purchase fruits and vegetables every day. Upon returning to the store the chef must enter two pieces of data for each item purchased: the quantity (Example: 2 cases) and the price paid (Example: $4.67). The program has a list of 20 items and after the chef enters the information, the program provides a total for the purchases for that day. Prepare test data for five (5) items: apples, oranges, bananas, lettuce, and tomatoes.

Programming Languages and Integrated Development Environments

The following activities focus on selecting a programming language and testing integrated development environments.

1. Research different programming languages and select a programming language to use with this textbook. Copy the Hello World example code for your selected programming language and use one of the free cloud-based IDEs to try running the Hello World program.
2. Modify the example Hello World program to instead display `Hello <name>!`, where `<name>` is your name. Include comments at the top of the program and test the program to verify that it works correctly.
3. Research free downloadable tools for your selected programming language (interpreter/compiler, IDE, etc.). Consider downloading and installing a development environment on your system. If you set up your own development environment, test the environment using your Hello Name program written above.

References

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

PART II

DATA AND OPERATORS

Overview

This chapter introduces constants and variables, data types, and operators.

Chapter Outline

- Constants and Variables
- Identifier Names
- Data Types
  - Integer Data Type
  - Floating-Point Data Type
  - String Data Type
  - Boolean Data Type
  - Nothing Data Type
- Order of Operations
- Assignment
- Arithmetic Operators
- Integer Division and Modulus
- Unary Operations
- Lvalue and Rvalue
- Data Type Conversions
- Input-Process-Output Model
- Code Examples
  - C++
  - C#
  - Java
  - JavaScript
  - Python
  - Swift
- Practice

Learning Objectives

1. Understand key terms and definitions.
2. Understand basic data types and how operators manipulate data.
3. Given example pseudocode, flowcharts, and source code, create a program that uses appropriate data types and operators to solve a given problem.

Constants and Variables

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

A **constant** is a value that cannot be altered by the program during normal execution, i.e., the value is constant. When associated with an identifier, a constant is said to be "named," although the terms "constant" and "named constant" are often used interchangeably. This is contrasted with a **variable**, which is an identifier with a value that can be changed during normal execution, i.e., the value is variable.[1]

Discussion

Understanding Constants

A **constant** is a data item whose value cannot change during the program's execution. Thus, as its name implies – the value is constant.

A **variable** is a data item whose value can change during the program's execution. Thus, as its name implies – the value can vary.

Constants are used in two ways. They are:

1. literal constant
2. defined constant

A literal constant is a **value** you type into your program wherever it is needed. Examples include the constants used for initializing a variable and constants used in lines of code:

```
21
12.34
'A'
"Hello world!"
false
null
```

In addition to literal constants, most textbooks refer to symbolic constants or named constants as a constant represented by a name. Many programming languages use ALL CAPS to define named constants.

1. Wikipedia: Constant (computer programming)

Language	Example
C++	`#define PI 3.14159` or `const double PI = 3.14159;`
C#	`const double PI = 3.14159;`
Java	`const double PI = 3.14159;`
JavaScript	`const PI = 3.14159;`
Python	`PI = 3.14159`
Swift	`let pi = 3.14159`

Technically, Python does not support named constants, meaning that it is possible (but never good practice) to change the value of a constant later. There are workarounds for creating constants in Python, but they are beyond the scope of a first-semester textbook.

Defining Constants and Variables

Named constants must be assigned a value when they are defined. Variables do not have to be assigned initial values. Variables once defined may be assigned a value within the instructions of the program.

Language	Example
C++	`double value = 3;`
C#	`double value = 3;`
Java	`double value = 3;`
JavaScript	`var value = 3;` `let value = 3;`
Python	`value = 3`
Swift	`var value:Int = 3`

Key Terms

constant
A data item whose value cannot change during the program's execution.

variable
A data item whose value can change during the program's execution.

References

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

Identifier Names

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

Within programming a variety of items are given descriptive names to make the code more meaningful to us as humans. These names are called “Identifier Names”. Constants, variables, type definitions, functions, etc. when declared or defined are identified by a name. These names follow a set of rules that are imposed by:

1. the language’s technical limitations
2. good programming practices
3. common industry standards for the language

Discussion

Technical to Language

- Use only allowable characters (in many languages the first character must be alphabetic or underscore, can continue with alphanumeric or underscore)
- Can’t use reserved words
- Length limit

These attributes vary from one programming language to another. The allowable characters and reserved words will be different. The length limit refers to how many characters are allowed in an identifier name and often is compiler dependent and may vary from compiler to compiler for the same language. However, all programming languages have some form of the technical rules listed here.

Good Programming Techniques

- Meaningful
- Be case consistent

Meaningful identifier names make your code easier for another to understand. After all what does “p” mean? Is it pi, price, pennies, etc. Thus do not use cryptic (look it up in the dictionary) identifier names.

Some programming languages treat upper and lower case letters used in identifier names as the same. Thus: pig and Pig are treated as the same identifier name. Unknown to you the programmer, the compiler usually forces all identifier names to upper case. Thus: pig and Pig both get changed to PIG. However, not all programming languages act this way. Some will treat upper and lower case letters as being different things. Thus: pig and Pig are two different identifier names. If you declare it as pig and then reference it in your code later as Pig – you get a different variable or perhaps a compiler error. To avoid the problem altogether, we teach students to **be case consistent**. Use an

identifier name only one way and spell it (upper and lower case) the same way every time within your program.

Industry Rules

Almost all programming languages and most coding shops have a standard code formatting style guide programmers are expected to follow. Among these are three common identifier casing standards:

- camelCase – each word is capitalized except the first word, with no intervening spaces
- PascalCase – each word is capitalized including the first word, with no intervening spaces
- snake_case – each word is lowercase with underscores separating words

C++, Java, and JavaScript typically use camelCase, with PascalCase reserved for libraries and classes. C# uses primarily PascalCase with camelCase parameters. Python uses snake_case for most identifiers. In addition, the following rules apply:

- Do not start with an underscore (used for technical programming)
- CONSTANTS IN ALL UPPER CASE (often UPPER_SNAKE_CASE).

These rules are decided by the industry (those who are using the programming language).

Key Terms

camel case
: The practice of writing compound words or phrases such that each word or abbreviation in the middle of the phrase begins with a capital letter, with no intervening spaces or punctuation.

Pascal case
: The practice of writing compound words or phrases such that each word or abbreviation in the phrase begins with a capital letter, including the first letter, with no intervening spaces or punctuation.

reserved word
: Words that cannot be used by the programmer as identifier names because they already have a specific meaning within the programming language.

snake case
: The practice of writing compound words or phrases in which the elements are separated with one underscore character (_) and no spaces, with each element's initial letter usually lowercased within the compound and the first letter either upper or lower case.

References

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

Data Types

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

A **data type** is a classification of data which tells the compiler or interpreter how the programmer intends to use the data. Most programming languages support various types of data, including integer, real, character or string, and Boolean.[1]

Discussion

Our interactions (inputs and outputs) with a program are treated in many languages as a stream of bytes. These bytes represent data that can be interpreted as representing values that we understand. Additionally, within a program, we process this data in various ways such as adding them up or sorting them. This data comes in different forms. Examples include:

- your name – a string of characters
- your age – usually an integer
- the amount of money in your pocket – usually a value measured in dollars and cents (something with a fractional part)

A major part of understanding how to design and code programs is centered in understanding the types of data that we want to manipulate and how to manipulate that data.

Common data types include:

Data Type	Represents	Examples
integer	whole numbers	`-5` , `0` , `123`
floating point (real)	fractional numbers	`-87.5` , `0.0` , `3.14159`
string	A sequence of characters	`"Hello world!"`
Boolean	logical true or false	`true` , `false`
nothing	no data	`null`

The common data types usually exist in most programming languages and act or behave similarly from language to language. Additional complex and/or composite data types may exist and vary from language to language.

1. Wikipedia: Data type

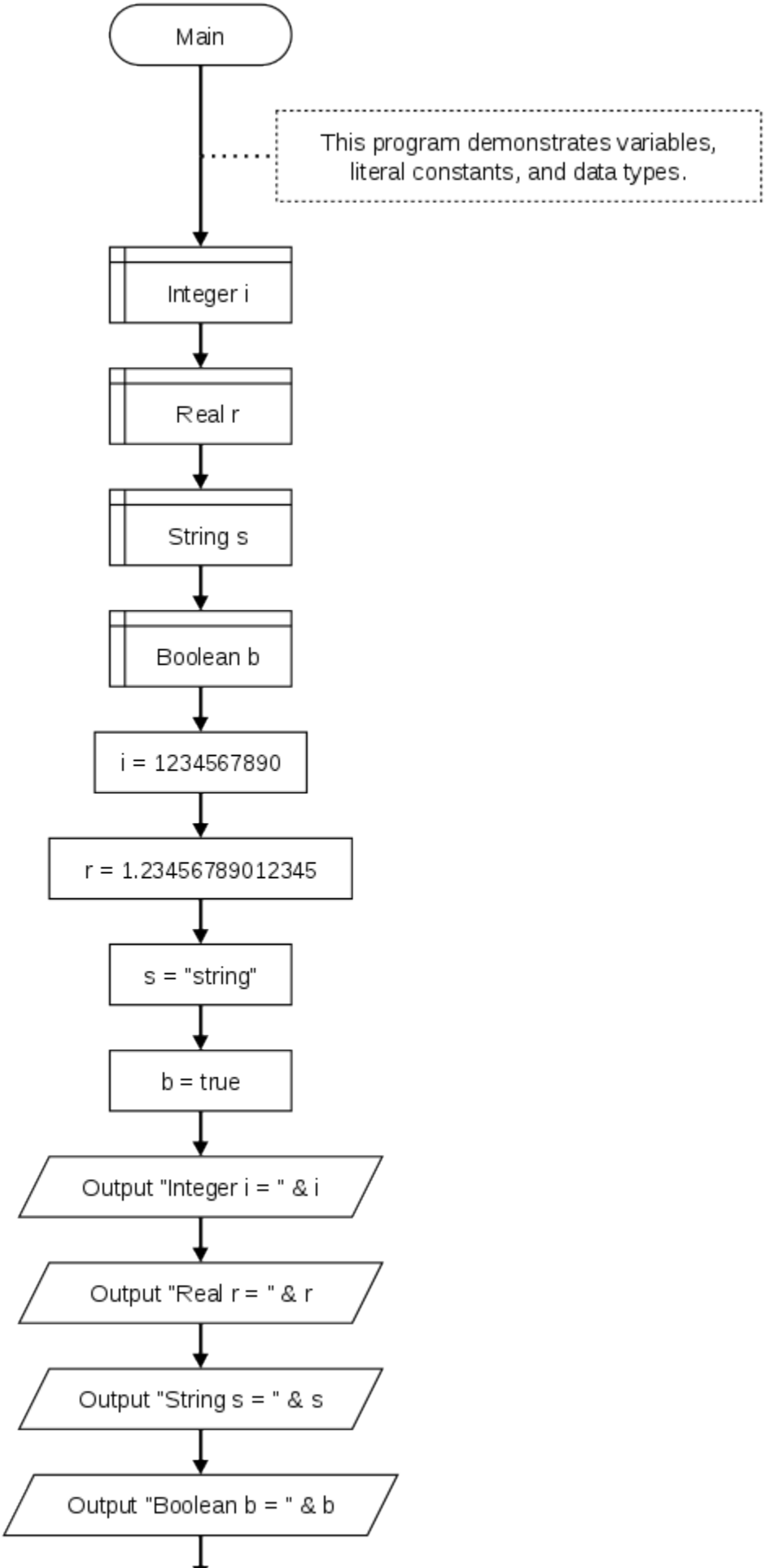
Pseudocode

```
Function Main
    ... This program demonstrates variables, literal constants, and data ty

    Declare Integer i
    Declare Real r
    Declare String s
    Declare Boolean b

    Assign i = 1234567890
    Assign r = 1.23456789012345
    Assign s = "string"
    Assign b = true

    Output "Integer i = " & i
    Output "Real r = " & r
    Output "String s = " & s
    Output "Boolean b = " & b
End
```

Output

```
Integer i = 1234567890
Real r = 1.23456789012345
String s = string
Boolean b = true
```

Flowchart


Main
This program demonstrates variables,
literal constants, and data types.
Integer i
Real r
String s
Boolean b
i = 1234567890
r = 1.23456789012345
s = "string"
b = true
Output "Integer i = " & i
Output "Real r = " & r
Output "String s = " & s
Output "Boolean b = " & b

Key Terms

Boolean
: A data type representing logical true or false.

data type
: Defines a set of values and a set of operations that can be applied on those values.

floating point
: A data type representing numbers with fractional parts.

integer
: A data type representing whole numbers.

string
: A data type representing a sequence of characters.

References

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

Integer Data Type

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

An **integer data type** represents some range of mathematical integers. Integral data types may be of different sizes and may or may not be allowed to contain negative values. Integers are commonly represented in a computer as a group of binary digits (bits). The size of the grouping varies so the set of integer sizes available varies between different types of computers and different programming languages.[1]

Discussion

The integer data type basically represents whole numbers (no fractional parts). The integer values jump from one value to another. There is nothing between 6 and 7. It could be asked why not make all your numbers floating point which allow for fractional parts. The reason is threefold. First, some things in the real world are not fractional. A dog, even with only 3 legs, is still one (1) dog not ¾ of a dog. Second, the integer data type is often used to control program flow by counting, thus the need for a data type that jumps from one value to another. Third, integer processing is significantly faster within the CPU than is floating point processing.

The integer data type has similar attributes and acts or behaves similarly in all programming languages that support it.

1. Wikipedia: Integer (computer science)

Language	Reserved Word	Size	Range
C++	`short`	16 bits / 2 bytes	-32,768 to32,767
C++	`int`	varies	depends on compiler
C++	`long`	32 bits / 4 bytes	-2,147,483,648 to 2, 147,483,647
C++	`long long`	64 bits / 8 bytes	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
C#	`short`	16 bits / 2 bytes	-32,768 to32,767
C#	`int`	32 bits / 4 bytes	-2,147,483,648 to 2, 147,483,647
C#	`long`	64 bits / 8 bytes	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Java	`short`	16 bits / 2 bytes	-32,768 to32,767
Java	`int`	32 bits / 4 bytes	-2,147,483,648 to 2, 147,483,647
Java	`long`	64 bits / 8 bytes	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
JavaScript	N/A		
Python	`int()`		no limit
Swift	`Int`	varies	depends on platform
Swift	`Int32`	32 bits / 4 bytes	-2,147,483,648 to 2, 147,483,647
Swift	`Int64`	64 bits / 8 bytes	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

For C++ and Swift the size of a default integer varies with the compiler being used and the computer. This effect is known as being **machine dependent**. These variations of the integer data type are an annoyance for a beginning programmer. For a beginning programmer, it is more important to understand the general attributes of the integer data type that apply to most programming languages.

JavaScript does not support an integer data type, but the `Math.round()` function may be used to return the value of a number rounded to the nearest integer.[2]

Python 3 integers are not limited in size, however, `sys.maxsize` may be used to determine the maximum practical size of a list or string index.[3]

Key Terms

machine dependent
An attribute of a programming language that changes depending on the computer's CPU.

2. Mozilla: Math.round()
3. Python.org: Integers

References

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

Floating-Point Data Type

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

A **floating-point data type** uses a formulaic representation of real numbers as an approximation so as to support a trade-off between range and precision. For this reason, floating-point computation is often found in systems which include very small and very large real numbers, which require fast processing times. A number is, in general, represented approximately to a fixed number of significant digits and scaled using an exponent in some fixed base.[1]

Discussion

The floating-point data type is a family of data types that act alike and differ only in the size of their domains (the allowable values). The floating-point family of data types represents number values with fractional parts. They are technically stored as two integer values: a **mantissa** and an **exponent**. The floating-point family has the same attributes and acts or behaves similarly in all programming languages. They can always store negative or positive values thus they always are signed; unlike the integer data type that could be unsigned. The **domain** for floating-point data types varies because they could represent very large numbers or very small numbers. Rather than talk about the actual values, we mention the **precision**. The more bytes of storage the larger the mantissa and exponent, thus more precision.

Language	Reserved Word	Size	Precision	Range
C++	`float`	32 bits / 4 bytes	7 decimal digits	±3.40282347E+38
C++	`double`	64 bits / 8 bytes	15 decimal digits	±1.79769313486231570E+308
C#	`float`	32 bits / 4 bytes	7 decimal digits	±3.40282347E+38
C#	`double`	32 bits / 4 bytes	15 decimal digits	±1.79769313486231570E+308
Java	`float`	32 bits / 4 bytes	7 decimal digits	±3.40282347E+38
Java	`double`	32 bits / 4 bytes	15 decimal digits	±1.79769313486231570E+308
JavaScript	`Number`	64 bits / 8 bytes	15 decimal digits	±1.79769313486231570E+308
Python	`float()`	64 bits / 8 bytes	15 decimal digits	±1.79769313486231570E+308
Swift	`Float`	32 bits / 4 bytes	7 decimal digits	±3.40282347E+38
Swift	`Double`	64 bits / 8 bytes	15 decimal digits	±1.79769313486231570E+308

1. Wikipedia: Floating-point arithmetic

Key Terms

double
The most often used floating-point family data type used.
mantissa exponent
The two integer parts of a floating-point value.
precision
The effect on the domain of floating-point values given a larger or smaller storage area in bytes.

References

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

String Data Type

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

A **string data type** 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

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]

The **character** data type represents individual or single characters. Characters comprise a variety of symbols such as the alphabet (both upper and lower case) the numeral digits (0 to 9), punctuation, etc. All computers store character data in a one-byte field as an integer value. Because a byte consists of 8 bits, this one-byte field has 2^8 or 256 possibilities using the positive values of 0 to 255.

C++, C#, and Java differentiate between single characters and strings using single quotes and double quotes, respectively. JavaScript, Python, and Swift do not differentiate between characters and strings and use either single quotes or double quotes to define string literals.

1. Wikipedia: String (computer science)
2. Wikipedia: String (computer science)

Language	Reserved Word	Example
C++	`char`	`'A'`
C++	`string`	`"Hello world!"`
C#	`char`	`'A'`
C#	`String`	`"Hello world!"`
Java	`char`	`'A'`
Java	`String`	`"Hello world!"`
JavaScript	`String`	`'Hello world!'` , `"Hello world!"`
Python	`str()`	`'Hello world!'` , `"Hello world!"`
Swift	`Character`	`"A"`
Swift	`String`	`"Hello world!"`

Most computing devices use the **ASCII** (stands for American Standard Code for Information Interchange and is pronounced "ask-key") Character Set which has established values for 0 to 127. For the values of 128 to 255 they usually use the Extended ASCII Character Set. When we hit the capital A on the keyboard, the keyboard sends a byte with the bit pattern equal to an integer 65. When the byte is sent from the memory to the monitor, the monitor converts the integer value of 65 to into the symbol of the capital A to display on the monitor.

For now, we will address only the use of strings and characters as constants. Most modern compilers that are part of an Integrated Development Environment (IDE) will color the source code to help the programmer see different features more readily. Beginning programmers will use string constants to send messages to standard output.

Key Terms

ASCII
American Standard Code for Information Interchange

character
A data type representing single text characters like the alphabet, numeral digits, punctuation, etc.

double quote marks
Used to create string type data within most programming languages.

single quote marks
Used to create character type data within languages that differentiate between string and character data types.

string
A series or array of characters as a single piece of data.

References

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

Boolean Data Type

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

A **Boolean data type** has one of two possible values (usually denoted true and false), intended to represent the two truth values of logic and Boolean algebra. It is named after George Boole, who first defined an algebraic system of logic in the mid 19th century. The Boolean data type is primarily associated with conditional statements, which allow different actions by changing control flow depending on whether a programmer-specified Boolean condition evaluates to true or false.[1]

Discussion

The Boolean data type is also known as the logical data type and represents the concepts of true and false. The name "Boolean" comes from the mathematician George Boole; who in 1854 published: An Investigation of the Laws of Thought. Boolean algebra is the area of mathematics that deals with the logical representation of true and false using the numbers 0 and 1. The importance of the Boolean data type within programming is that it is used to control programming structures (if then else, while loops, etc.) that allow us to implement "choice" into our algorithms.

The Boolean data type has the same attributes and acts or behaves similarly in all programming languages. However, while all languages recognize false as 0, some languages define true as -1 rather than 1. This is the result of storing the Boolean values as an integer and using a one's complement representation that negates all bits rather than only the rightmost bit. To simplify processing, most programming languages recognize any non-zero value as being true.

Language	Reserved Word	True	False
C++	`bool`	`true`	`false`
C#	`bool` or `Boolean`	`true`	`false`
Java	`bool`	`true`	`false`
JavaScript	`Boolean()`	`true`	`false`
Python	`bool()`	`True`	`False`
Swift	`Bool`	`true`	`false`

1. Wikipedia: Boolean data type

Key Terms

Boolean
A data type representing the concepts of true or false.

one's complement
The value obtained by inverting all the bits in the binary representation of a number (swapping 0s for 1s and vice versa).

References

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

Nothing Data Type

DAVE BRAUNSCHWEIG

Overview

A **nothing data type** is a feature of some programming languages which allow the setting of a special value to indicate a missing or uninitialized value rather than using the value 0 (zero).[1]

Discussion

Most programming languages support the use of a reserved word or words to represent missing, uninitialized, or invalid values.

Language	Reserved Word	Meaning
C++	`null`	no value
C#	`null`	no value
Java	`null`	no value
JavaScript	`null`	no value
JavaScript	`NaN`	Not a Number
Python	`None`	no value
Swift	`nil`	no value

Key Terms

NaN
: Reserved word used to indicate a non-numeric value in a numeric variable.

null
: Reserved word used to represent a missing value or invalid value.

1. Wikipedia: Nullable type

Order of Operations

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

The order of operations (or operator precedence) is a collection of rules that reflect conventions about which procedures to perform first in order to evaluate a given mathematical expression.[1]

Discussion

Single values by themselves are important; however, we need a method of manipulating values (processing data). Scientists wanted an accurate machine for manipulating values. They wanted a machine to process numbers or calculate answers (that is, compute the answer). Prior to 1950, dictionaries listed the definition of computers as " humans that do computations". Thus, all of the terminology for describing data manipulation is math oriented. Additionally, the two fundamental data type families (the integer family and floating-point family) consist entirely of number values.

An Expression Example with Evaluation

Let's look at an example: 2 + 3 * 4 + 5 is our expression but what does it equal?

1. the symbols of + meaning addition and * meaning multiplication are our operators
2. the values 2, 3, 4 and 5 are our operands
3. precedence says that multiplication is higher than addition
4. thus, we evaluate the 3 * 4 to get 12
5. now we have: 2 + 12 + 5
6. the associativity rules say that addition goes left to right, thus we evaluate the 2 +12 to get 14
7. now we have: 14 + 5
8. finally, we evaluate the 14 + 5 to get 19; which is the value of the expression

Parentheses would change the outcome. (2 + 3) * (4 + 5) evaluates to 45.
Parentheses would change the outcome. (2 + 3) * 4 + 5 evaluates to 25.

Operator Precedence Chart

Each computer language has some rules that define precedence and associativity. They often follow rules we may have already learned. Multiplication and division come before addition and subtraction is a rule we learned in grade school. This rule still works.
Order of Operations[2]

1. Wikipedia: Order of operations
2. Wikipedia: Order of operations

- Parentheses
- Exponents
- Multiplication / Division
- Addition / Subtraction

A common mnemonic to remember this rule is *PEMDAS*, or *Please Excuse My Dear Aunt Sally*. Precedence rules may vary from one programming language to another. You should refer to the reference sheet that summarizes the rules for the language that you are using. It is often called an Operator Precedence, Precedence of Operators, or Order of Operations chart. You should review this chart as needed when evaluating expressions.

A valid expression consists of operand(s) and operator(s) that are put together properly. Why the (s)? Some operators are:

1. Unary – only have one operand
2. Binary – have two operands, one on each side of the operator
3. Trinary – have two operator symbols that separate three operands

Most operators are binary, that is they require two operands. Some precedence charts indicate of which operators are unary and trinary and thus all others are binary.

Key Terms

associativity
: Determines the order in which the operators of the same precedence are allowed to manipulate the operands.

evaluation
: The process of applying the operators to the operands and resulting in a single value.

expression
: A valid sequence of operand(s) and operator(s) that reduces (or evaluates) to a single value.

operand
: A value that receives the operator's action.

operator
: A language-specific syntactical token (usually a symbol) that causes an action to be taken on one or more operands.

parentheses
: Change the order of evaluation in an expression. You do what's in the parentheses first.

precedence
: Determines the order in which the operators are allowed to manipulate the operands.

References

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

Assignment

KENNETH LEROY BUSBEE

Overview

An **assignment** statement sets and/or re-sets the value stored in the storage location(s) denoted by a variable name; in other words, it copies a value into the variable.[1]

Discussion

The assignment operator allows us to change the value of a modifiable data object (for beginning programmers this typically means a variable). It is associated with the concept of moving a value into the storage location (again usually a variable). Within most programming languages the symbol used for assignment is the equal symbol. But bite your tongue, when you see the = symbol you need to start thinking: assignment. The assignment operator has two operands. The one to the left of the operator is usually an identifier name for a variable. The one to the right of the operator is a value.

Simple Assignment

```
age = 21
```

The value 21 is moved to the memory location for the variable named: age. Another way to say it: age is assigned the value 21.

Assignment with an Expression

```
total_cousins = 4 + 3 + 5 + 2
```

The item to the right of the assignment operator is an expression. The expression will be evaluated and the answer is 14. The value 14 would be assigned to the variable named: total_cousins.

Assignment with Identifier Names in the Expression

```
students_period_1 = 25
students_period_2 = 19
total_students = students_period_1 + students_period_2
```

The expression to the right of the assignment operator contains some identifier names. The program would fetch the values stored in those variables; add them together and get a value of 44; then assign the 44 to the total_students variable.

1. Wikipedia: Assignment (computer science)

Key Terms

assignment
An operator that changes the value of a modifiable data object.

References

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

Arithmetic Operators

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

The basic **arithmetic operations** are addition, subtraction, multiplication, and division. Arithmetic is performed according to an order of operations.[1]

Discussion

An operator performs an action on one or more operands. The common arithmetic operators are:

Action	Common Symbol
Addition	`+`
Subtraction	`-`
Multiplication	`*`
Division	`/`
Modulus (associated with integers)	`%`

These arithmetic operators are binary that is they have two operands. The operands may be either constants or variables.

```
age + 1
```

This expression consists of one operator (addition) which has two operands. The first is represented by a variable named age and the second is a literal constant. If age had a value of 14 then the expression would evaluate (or be equal to) 15.

These operators work as you have learned them throughout your life with the exception of division and modulus. We normally think of division as resulting in an answer that might have a fractional part (a floating-point data type). However, division, when both operands are of the integer data type, may act differently. Please refer to the next section on “Integer Division and Modulus”.

Arithmetic Assignment Operators

Many programming languages support a combination of the assignment (`=`) and arithmetic operators (`+` , `-` , `*` , `/` , `%`). Various textbooks call them “compound assignment operators” or “combined assignment operators”. Their usage can be explained in terms of the assignment operator and the arithmetic operators. In the table, we will use the variable age and you can assume that it is of integer data type.

1. Wikipedia: Arithmetic operators

Arithmetic assignment examples:	Equivalent code:
`age += 14;`	`age = age + 14;`
`age -= 14;`	`age = age - 14;`
`age *= 14;`	`age = age * 14;`
`age /= 14;`	`age = age / 14;`
`age %= 14;`	`age = age % 14;`

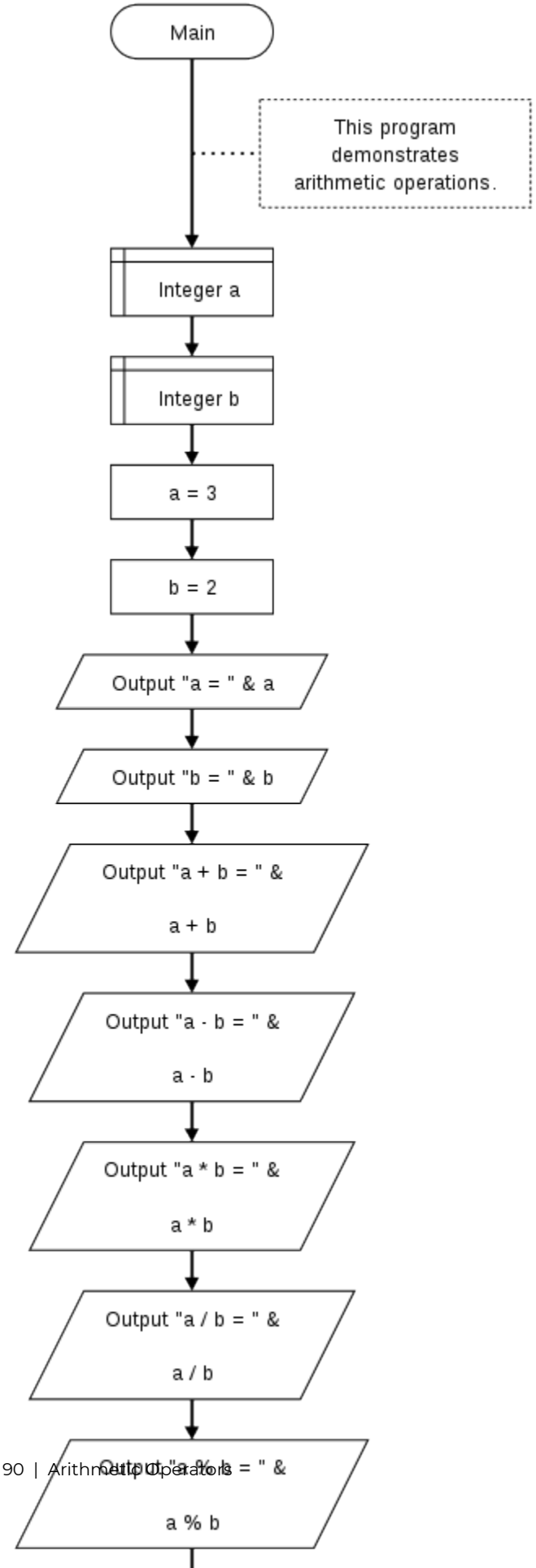
Pseudocode

```
Function Main
    ... This program demonstrates arithmetic operations.
    Declare Integer a
    Declare Integer b

    Assign a = 3
    Assign b = 2
    Output "a = " & a
    Output "b = " & b
    Output "a + b = " & a + b
    Output "a - b = " & a - b
    Output "a * b = " & a * b
    Output "a / b = " & a / b
    Output "a % b = " & a % b
End
```

Output

```
a = 3
b = 2
a + b = 5
a - b = 1
a * b = 6
a / b = 1.5
a % b = 1
```

Flowchart


Main
This program demonstrates arithmetic operations.
Integer a
Integer b
a = 3
b = 2
Output "a = " & a
Output "b = " & b
Output "a + b = " & a + b
Output "a - b = " & a - b
Output "a * b = " & a * b
Output "a / b = " & a / b
Output "a % b = " & a % b

References

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

Integer Division and Modulus

KENNETH LEROY BUSBEE

Overview

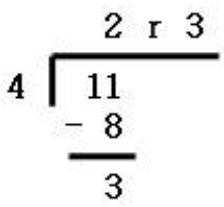
In **integer division** and **modulus**, the dividend is divided by the divisor into an integer quotient and a remainder. The integer quotient operation is referred to as integer division, and the integer remainder operation is the modulus.[1][2]

Discussion

By the time we reach adulthood, we normally think of division as resulting in an answer that might have a fractional part (a floating-point data type). This type of division is known as floating-point division. However, division, when both operands are of the integer data type, may act differently, depending on the programming language, and is called: **integer division**. Consider:

```
11 / 4
```

Because both operands are of the integer data type the evaluation of the expression (or answer) would be 2 with no fractional part (it gets thrown away). Again, this type of division is called **integer division** and it is what you learned in grade school the first time you learned about division.



Integer division as learned in grade school.

In the real world of data manipulation there are some things that are always handled in whole units or numbers (integer data type). **Fractions just don't exist.** To illustrate our example: I have 11 dollar coins to distribute equally to my 4 children. How many do they each get? The answer is 2, with me still having 3 left over (or with 3 still remaining in my hand). The answer is not 2 ¾ each or 2.75 for each child. The dollar coins are not divisible into fractional pieces. Don't try thinking out of the box and pretend you're a pirate. Using an axe and chopping the 3 remaining coins into pieces of eight. Then, giving each child 2 coins and 6 pieces of eight or 2 6/8 or 2 ¾ or 2.75. If you do think this way, I will change my example to cans of tomato soup. I dare you to try and chop up three cans of soup

1. Wikipedia: Division (mathematics)
2. Wikipedia: Modulo operation

and give each kid ¾ of a can. Better yet, living things like puppy dogs. After you divide them up with an axe, most children will not want the ¾ of a dog.

What is **modulus**? It's the other part of the answer for integer division. It's the remainder. Remember in grade school you would say, "Eleven divided by four is two remainder three." In many programming languages, the symbol for the modulus operator is the percent sign (%).

```
11 % 4
```

Thus, the answer or value of this expression is 3 or the remainder part of integer division.

Many compilers require that you have integer operands on both sides of the modulus operator or you will get a compiler error. In other words, it does not make sense to use the modulus operator with floating-point operands.

Don't let the following items confuse you.

```
6 / 24 which is different from 6 % 24
```

How many times can you divide 24 into 6? Six divided by 24 is zero. This is different from: What is the remainder of 6 divided by 24? Six, the remainder part is given by modulus.

Evaluate the following division expressions:

1. 14 / 4
2. 5 / 13
3. 7 / 2.0

Evaluate the following modulus expressions:

1. 14 % 4
2. 5 % 13
3. 7 % 2.0

Key Terms

integer division
Division with no fractional parts.

modulus
The remainder part of integer division.

References

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

Unary Operations

KENNETH LEROY BUSBEE

Overview

A **unary operation** is an operation with only one operand. As unary operations have only one operand, they are evaluated before other operations containing them.[1] Common unary operators include Positive (`+`) and Negative (`-`).

Discussion

Unary positive also known as plus and unary negative also known as minus are unique operators. The plus and minus when used with a constant value represent the concept that the values are either positive or negative. Let's consider:

```
+5 + -2
```

We have three operators in this order: unary positive, addition, and unary negative. The answer to this expression is a positive 3. As you can see, one must differentiate between when the plus sign means unary positive and when it means addition. Unary negative and subtraction have the same problem. Let's consider:

```
-2 - +5
```

The expression evaluates to negative 7. Let's consider:

```
7 - -2
```

First constants that do not have a unary minus in front of them are assumed (the default) to be positive. When you subtract a negative number it is like adding, thus the expression evaluates to positive 9.

Negation – Unary Negative

The concept of negation is to take a value and change its sign, that is: flip it. If it is positive make it negative and if it is negative make it positive. Mathematically, it is the following C++ code example, given that money is an integer variable with a value of 6:

```
-money
money * -1
```

The above two expressions evaluate to the same value. In the first line, the value in the variable money is fetched and then it's negated to a negative 6. In the second line, the value in the variable money is fetched and then it's multiplied by negative 1 making the answer a negative 6.

Unary Positive – Worthless

Simply to satisfy symmetry, the unary positive was added to the C++ programming language as on

1. Wikipedia: Unary operation

operator. However, it is a totally worthless or useless operator and is rarely used. However, don't be confused the following expression is completely valid:

```
6 + +5
```

The second + sign is interpreted as unary positive. The first + sign is interpreted as addition.

```
money
+money
money * +1
```

For all three lines, if the value stored in money is 6 the value of the expression is 6. Even if the value in money was negative 77 the value of the expression would be negative 77. The operator does nothing because multiplying anything by 1 does not change its value.

Possible Confusion

Do not confuse the unary negative operator with decrement. Decrement changes the value in the variable and thus is an Lvalue concept. Unary negative does not change the value of the variable but uses it in an Rvalue context. It fetches the value and then negates that value. The original value in the variable does not change.

Because there is no changing of the value associated with the identifier name, the identifier name could represent a variable or named constant.

Exercises

Evaluate the following items involving unary positive and unary negative:

1. +10 – -2
2. -18 + 24
3. 4 – +3
4. +8 + – +5
5. +8 + / +5

Key Terms

minus
Aka unary negative.
plus
Aka unary positive.
unary negative
An operator that causes negation.
unary positive
A worthless operator almost never used.

References

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

Lvalue and Rvalue

KENNETH LEROY BUSBEE

Overview

Some programming languages use the idea of **l-values** and **r-values**, deriving from the typical mode of evaluation on the left and right hand side of an assignment statement. An lvalue refers to an object that persists beyond a single expression. An rvalue is a temporary value that does not persist beyond the expression that uses it.[1]

Discussion

Lvalue and Rvalue refer to the left and right side of the assignment operator. The **Lvalue** (pronounced: L value) concept refers to the requirement that the operand on the left side of the assignment operator is modifiable, usually a variable. **Rvalue** concept pulls or fetches the value of the expression or operand on the right side of the assignment operator. Some examples:

```
age = 39
```

The value 39 is pulled or fetched (Rvalue) and stored into the variable named age (Lvalue); destroying the value previously stored in that variable.

```
voting_age = 18
age = voting_age
```

If the expression has a variable or named constant on the right side of the assignment operator, it would pull or fetch the value stored in the variable or constant. The value 18 is pulled or fetched from the variable named voting_age and stored into the variable named age.

```
age < 17
```

If the expression is a test expression or Boolean expression, the concept is still an Rvalue one. The value in the identifier named age is pulled or fetched and used in the relational comparison of less than.

```
JACK_BENNYS_AGE = 39
JACK_BENNYS_AGE = 65;
```

1. Wikipedia: Value (computer science)

This is illegal because the identifier JACK_BENNYS_AGE does not have Lvalue properties. It is not a modifiable data object, because it is a constant.

Some uses of the Lvalue and Rvalue can be confusing in languages that support increment and decrement operators. Consider:

```
oldest = 55
age = oldest++
```

Postfix increment says to use my existing value then when you are done with the other operators; increment me. Thus, the first use of the oldest variable is an Rvalue context where the existing value of 55 is pulled or fetched and then assigned to the variable age; an Lvalue context. The second use of the oldest variable is an Lvalue context wherein the value of the oldest is incremented from 55 to 56.

Key Terms

Lvalue
: The requirement that the operand on the left side of the assignment operator is modifiable, usually a variable.

Rvalue
: Pulls or fetches the value stored in a variable or constant.

References

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

Data Type Conversions

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

Changing a data type of a value is referred to as “type conversion”. There are two ways to do this:

1. **Implicit** – the change is implied
2. **Explicit** – the change is explicitly done with an operator or function

The value being changed may be:

1. **Promotion** – going from a smaller domain to a larger domain
2. **Demotion** – going from a larger domain to a smaller domain

Discussion

Implicit Type Conversion

Automatic conversion of a value from one data type to another by a programming language, without the programmer specifically doing so, is called implicit type conversion. It happens whenever a binary operator has two operands of different data types. Depending on the operator, one of the operands is going to be converted to the data type of the other. It could be promoted or demoted depending on the operator.

Implicit Promotion

```
55 + 1.75
```

In this example, the integer value 55 is converted to a floating-point value (most likely double) of 55.0. It was promoted.

Implicit Demotion

In programming languages that have explicit integer data types (C++, C#, Java), care must be taken to avoid implicit demotion. For example:

```
int money;
money = 23.16;
```

In this example, the variable money is an integer. We are trying to move a floating-point value 23.16 into an integer storage location. This is demotion and the floating-point value usually gets truncated to 23.

Promotion

Promotion is never a problem because the lower data type (smaller range of allowable values) is a subset of the higher data type (larger range of allowable values). Promotion often occurs with

three of the standard data types: character, integer, and floating-point. The allowable values (or domains) progress from one type to another. That is, the character data type values are a subset of integer values and integer values are a subset of floating-point values; and within the floating-point values, float values are a subset of double. Even though character data represent the alphabetic letters, numeral digits (0 to 9) and other symbols (a period, $, comma, etc.) their bit pattern also represent integer values from 0 to 255. This progression allows us to promote them up the chain from character to integer to float to double.

Demotion

Demotion represents a potential problem with truncation or unpredictable results often occurring. How do you fit an integer value of 456 into a character value? How do you fit the floating-point value of 45656.453 into an integer value? Most compilers give a warning if it detects demotion happening. A compiler warning does not stop the compilation process. It does warn the programmer to check to see if the demotion is reasonable.

If I calculate the number of cans of soup to buy based on the number of people I am serving (say 8) and the servings per can (say 2.3), I would need 18.4 cans. I might want to demote the 18.4 into an integer. It would truncate the 18.4 into 18 and because the value 18 is within the domain of an integer data type, it should demote with the **truncation** side effect.

If I tried demoting a double that contained the number of stars in the Milky Way galaxy into an integer, I might have a get an **unpredictable result** (assuming the number of stars is larger than allowable values within the integer domain).

Explicit Type Conversion

Most languages have a method for the programmer to change or cast a value from one data type to another; called **explicit type conversion**. Some languages support a cast operator. The cast operator is a unary operator; it only has one operand and the operand is to the right of the operator. The operator is a set of parentheses surrounding the new data type. Other languages have functions that perform explicit type conversion. In each of the following examples, the expression value would be 3.

Language	Floating-Point to Integer Type Conversion Example
C++	`(int) 3.14`
C#	`Convert.ToInt32(3.14)`
Java	`Math.floor(3.14)`
JavaScript	`Math.floor(3.14)`
Python	`int(3.14)`
Swift	`Int(3.14)`

In each of the following examples, the expression value would be 3.14.

Language	String to Floating-Point Type Conversion Example
C++	`#include <string.h>` `std::stod("3.14")`
C#	`Convert.ToDouble("3.14")`
Java	`Double.parseDouble("3.14")`
JavaScript	`parseFloat("3.14")`
Python	`float("3.14")`
Swift	`Double("3.14")`

Key Terms

demotion
: Going from a larger domain to a smaller domain.

explicit
: Changing a value's data type with the cast operator.

implicit
: A value that has its data type changed automatically.

promotion
: Going from a smaller domain to a larger domain.

truncation
: The fractional part of a floating-point data type that is dropped when converted to an integer.

References

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

Input-Process-Output Model

DAVE BRAUNSCHWEIG

Overview

The **input–process–output (IPO) model** is a widely used approach in systems analysis and software engineering for describing the structure of an information processing program or another process. Many introductory programming and systems analysis texts introduce this as the most basic structure for describing a process.[1]

Discussion

A computer program or any other sort of process using the input-process-output model receives inputs from a user or other source, does some computations on the inputs, and returns the results of the computations. The system divides the work into three categories:[2]

- A requirement from the environment (input)
- A computation based on the requirement (process)
- A provision for the environment (output)

For example, a program might be written to convert Fahrenheit temperatures into Celsius temperatures. Following the IPO model, the program must:

- Ask the user for the Fahrenheit temperature (input)
- Perform a calculation to convert the Fahrenheit temperature into the corresponding Celsius temperature (process)
- Display the Celsius temperature (output)

Pseudocode

```
Function Main
    ... This program converts an input Fahrenheit temperature to Celsius.

    Declare Real fahrenheit
    Declare Real celsius

    Output "Enter Fahrenheit temperature:"
    Input fahrenheit
```

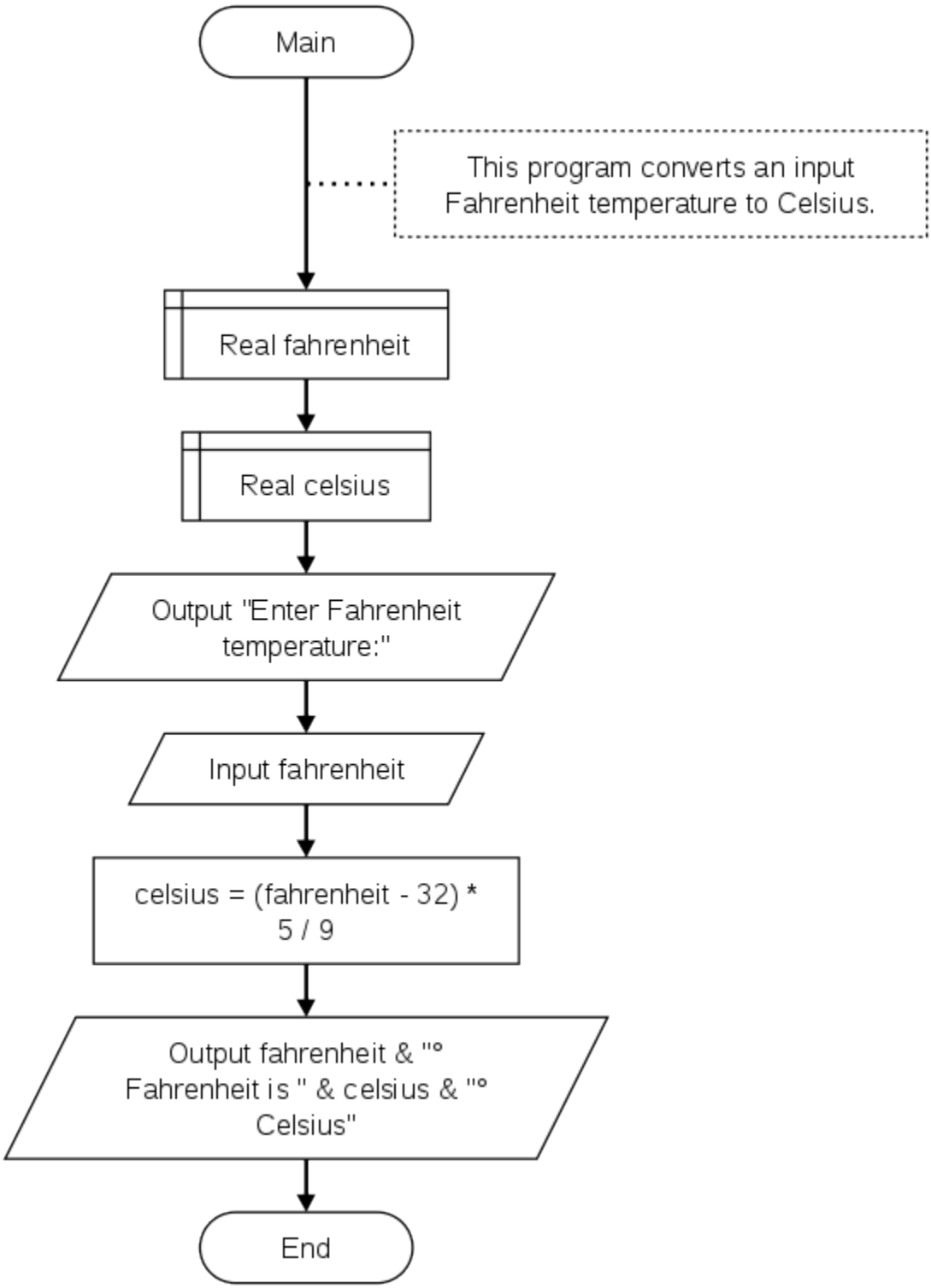
1. Wikipedia: IPO model
2. Wikipedia: IPO model

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

    Output fahrenheit & "° Fahrenheit is " & celsius & "° Celsius"
End
```

Output

```
Enter Fahrenheit temperature:
100
100° Fahrenheit is 37.7777777777778° Celsius
```

Flowchart


Main
This program converts an input Fahrenheit temperature to Celsius.
Real fahrenheit
Real celsius
Output "Enter Fahrenheit temperature:"
Input fahrenheit
celsius = (fahrenheit - 32) * 5 / 9
Output fahrenheit & "° Fahrenheit is " & celsius & "° Celsius"
End

References

- Wikiversity: Computer Programming
- Flowgorithm – Flowchart Programming Language

C++ Examples

DAVE BRAUNSCHWEIG

Overview

The following examples demonstrate data types, arithmetic operations, and input in C++.

Data Types

```
// This program demonstrates variables, literal constants, and data types.

#include <iostream>
#include <sstream>

using namespace std;

int main() {
    int i;
    double d;
    string s;
    bool b;

    i = 1234567890;
    d = 1.23456789012345;
    s = "string";
    b = true;
    cout << "Integer i = " << i << endl;
    cout << "Double d = " << d << endl;
    cout << "String s = " << s << endl;
    cout << "Boolean b = " << b << endl;
    return 0;
}
```

Output

```
Integer i = 1234567890
Real r = 1.23457
String s = string
```

```
Boolean b = 1
```

Discussion

Each code element represents:

- `//` begins a comment
- `#include <iostream>` includes standard input and output streams
- `#include <sstream>` includes standard string streams
- `using namespace std` allows reference to `string`, `cout`, and `endl` without writing `std::string`, `std::cout`, and `std::endl`.
- `int main()` begins the main function, which returns an integer value
- `{` begins a block of code
- `int i` defines an integer variable named i
- `;` ends each line of C++ code
- `double d` defines a double floating-point variable named d
- `string s` defines a string variable named s
- `bool b` defines a Boolean variable named b
- `i = , d = , s =, b =` assign literal values to the corresponding variables
- `cout` is standard output
- `<<` directs the next element to standard output
- `endl` ends the current line
- `return 0` returns the value 0 from main, indicating the main function completed successfully
- `}` ends a block of code

Arithmetic

```
// This program demonstrates arithmetic operations.

#include <iostream>
#include <sstream>

using namespace std;

int main() {
    int a;
    int b;

    a = 3;
    b = 2;
```

```
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "a + b = " << a + b << endl;
    cout << "a - b = " << a - b << endl;
    cout << "a * b = " << a * b << endl;
    cout << "a / b = " << a / b << endl;
    cout << "a % b = " << a + b << endl;
    return 0;
}
```

Output

```
a = 3
b = 2
a + b = 5
a - b = 1
a * b = 6
a / b = 1
a % b = 5
```

Discussion

Each new code element represents:

- `+, -, *, /, and %` represent addition, subtraction, multiplication, division, and modulus, respectively.

Temperature

```
// This program converts an input Fahrenheit temperature to Celsius.
//
// References:
// https://www.mathsisfun.com/temperature-conversion.html
// https://en.wikibooks.org/wiki/C%2B%2B_Programming
#include <iostream>

using namespace std;
```

```
int main() {
    double fahrenheit;
    double celsius;

    cout << "Enter Fahrenheit temperature:" << endl;
    cin >> fahrenheit;

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

    cout << fahrenheit << "° Fahrenheit is " << celsius << "° Celsius" << e

    return 0;
}
```

Output

```
Enter Fahrenheit temperature:
 100
100° Fahrenheit is 37.7778° Celsius
```

Discussion

Each new code element represents:

- `cin >> fahrenheit` reads the next integer from standard input and assigns the value to the fahrenheit variable

References

- Wikiversity: Computer Programming

C# Examples

DAVE BRAUNSCHWEIG

Overview

The following examples demonstrate data types, arithmetic operations, and input in C#.

Data Types

```
// This program demonstrates variables, literal constants, and data types.

using System;

public class DataTypes
{
    public static void Main(string[] args)
    {
        int i;
        double d;
        string s;
        Boolean b;

        i = 1234567890;
        d = 1.23456789012345;
        s = "string";
        b = true;

        Console.WriteLine("Integer i = " + i);
        Console.WriteLine("Double d = " + d);
        Console.WriteLine("String s = " + s);
        Console.WriteLine("Boolean b = " + b);
    }
}
```

Output

```
Integer i = 1234567890
Double d = 1.23456789012345
```

```
String s = string
Boolean b = True
```

Discussion

Each code element represents:

- `//` begins a comment
- `using System` allows references to `Boolean` and `Console` without writing `System.Boolean` and `System.Console`
- `public class DataTypes` begins the Data Types program
- `{` begins a block of code
- `public static void Main()` begins the main function
- `int i` defines an integer variable named i
- `;` ends each line of C# code
- `double d` defines a double floating-point variable named d
- `string s` defines a string variable named s
- `Boolean b` defines a Boolean variable named b
- `i = , d = , s =, b =` assign literal values to the corresponding variables
- `Console.WriteLine()` calls the standard output write line function
- `}` ends a block of code

Arithmetic

```
// This program demonstrates arithmetic operations.

using System;

public class Arithmetic
{
    public static void Main(string[] args)
    {
        int a;
        int b;

        a = 3;
        b = 2;

        Console.WriteLine("a = " + a);
        Console.WriteLine("b = " + b);
        Console.WriteLine("a + b = " + (a + b));
```

```
        Console.WriteLine("a - b = " + (a - b));
        Console.WriteLine("a * b = " + a * b);
        Console.WriteLine("a / b = " + a / b);
        Console.WriteLine("a % b = " + (a + b));
    }
}
```

Output

```
a = 3
b = 2
a + b = 5
a - b = 1
a * b = 6
a / b = 1
a % b = 5
```

Discussion

Each new code element represents:

- `+, -, *, /, and %` represent addition, subtraction, multiplication, division, and modulus, respectively.

Temperature

```
// This program converts an input Fahrenheit temperature to Celsius.

using System;

public class Temperature
{
    public static void Main(string[] args)
    {
        double fahrenheit;
        double celsius;

        Console.WriteLine("Enter Fahrenheit temperature:");
        fahrenheit = Convert.ToDouble(Console.ReadLine());
```

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

        Console.WriteLine(
            fahrenheit.ToString() + "° Fahrenheit is " +
            celsius.ToString() + "° Celsius" + "\n");
    }
}
```

Output

```
Enter Fahrenheit temperature:
 100
100° Fahrenheit is 37.7777777777778° Celsius
```

Discussion

Each new code element represents:

- `Console.ReadLine()` reads the next line from standard input
- `Convert.ToDouble` converts the input to a double floating-point value

References

- Wikiversity: Computer Programming

Java Examples

DAVE BRAUNSCHWEIG

Overview

The following examples demonstrate data types, arithmetic operations, and input in Java.

Data Types

```
// This program demonstrates variables, literal constants, and data types.

public class Main {
    public static void main(String[] args) {
        int i;
        double d;
        String s;
        boolean b;

        i = 1234567890;
        d = 1.23456789012345;
        s = "string";
        b = true;

        System.out.println("Integer i = " + i);
        System.out.println("Double d = " + d);
        System.out.println("String s = " + s);
        System.out.println("Boolean b = " + b);
    }
}
```

Output

```
Integer i = 1234567890
Double d = 1.23456789012345
String s = string
Boolean b = true
```

Discussion

Each code element represents:

- `//` begins a comment
- `public class DataTypes` begins the Data Types program
- `{` begins a block of code
- `public static void main(String[] args)` begins the main function
- `int i` defines an integer variable named i
- `;` ends each line of Java code
- `double d` defines a double floating-point variable named d
- `string s` defines a string variable named s
- `boolean b` defines a Boolean variable named b
- `i = , d = , s =, b =` assign literal values to the corresponding variables
- `System.out.println` calls the standard output print line function
- `}` ends a block of code

Arithmetic

```
// This program demonstrates arithmetic operations.

public class Main {
    public static void main(String[] args) {
        int a;
        int b;

        a = 3;
        b = 2;

        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("a + b = " + (a + b));
        System.out.println("a - b = " + (a - b));
        System.out.println("a * b = " + a * b);
        System.out.println("a / b = " + a / b);
        System.out.println("a % b = " + (a % b));
    }
}
```

Output

```
a = 3
b = 2
a + b = 5
a - b = 1
a * b = 6
a / b = 1
a % b = 1
```

Discussion

Each new code element represents:

- `+, -, *, /, and %` represent addition, subtraction, multiplication, division, and modulus, respectively.

Temperature

```
// This program converts an input Fahrenheit temperature to Celsius.

import java.util.*;

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

    public static void main(String[] args) {
        double fahrenheit;
        double celsius;

        System.out.println("Enter Fahrenheit temperature:");
        fahrenheit = input.nextDouble();

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

        System.out.println(Double.toString(fahrenheit) + "° Fahrenheit is "
    }
}
```

Output

```
Enter Fahrenheit temperature:
 100
100° Fahrenheit is 37.7777777777778° Celsius
```

Discussion

Each new code element represents:

- `private static Scanner input ...` defines an object to read from standard input
- `input.nextDouble()` reads input as a double floating-point value

References

- Wikiversity: Computer Programming

JavaScript Examples

DAVE BRAUNSCHWEIG

Overview

The following examples demonstrate data types, arithmetic operations, and input in JavaScript.

Data Types

```
// This program demonstrates variables, literal constants, and data types.

var n;
var s;
var b;

n = 1.23456789012345;
s = "string";
b = true;

output("Number n = " + n);
output("String s = " + s);
output("Boolean b = " + b);

// Display output to the current environment
function output(text) {
  if (typeof document === 'object') {
    document.write(text);
  }
  else if (typeof console === 'object') {
    console.log(text);
  }
  else {
    print(text);
  }
}
```

Output

```
Number n = 1.23456789012345
String s = string
Boolean b = true
```

Discussion

Each code element represents:

- `//` begins a comment
- `var n, s, and b` define variables
- `;` ends each line of JavaScript code
- `i = , d = , s =, b =` assign literal values to the corresponding variables
- `output()` calls the output function
- `function output(text)` defines a output function that checks the JavaScript environment and writes to the current document, the console, or standard output as appropriate.

Arithmetic

```
// This program demonstrates arithmetic operations.

var a;
var b;

a = 3;
b = 2;
output("a = " + a);
output("b = " + b);
output("a + b = " + (a + b));
output("a - b = " + (a - b));
output("a * b = " + a * b);
output("a / b = " + a / b);
output("a % b = " + (a % b));

// Display output to the current environment
function output(text) {
  if (typeof document === 'object') {
    document.write(text);
  }
```

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

Output

```
a = 3
b = 2
a + b = 5
a - b = 1
a * b = 6
a / b = 1.5
a % b = 1
```

Discussion

Each new code element represents:

- `+, -, *, /, and %` represent addition, subtraction, multiplication, division, and modulus, respectively.

Temperature

```
// This program converts an input Fahrenheit temperature to Celsius.

var fahrenheit;
var celsius;

output("Enter Fahrenheit temperature:");
fahrenheit = input();

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

output(fahrenheit.toString() + "° Fahrenheit is " + celsius + "° Celsius");
```

```
// Get input from the current environment
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();
  }
}

// Display output to the current environment
function output(text) {
  if (typeof document === 'object') {
    document.write(text);
  }
  else if (typeof console === 'object') {
    console.log(text);
  }
  else {
    print(text);
  }
}
```

Output

```
Enter Fahrenheit temperature:
 100
100° Fahrenheit is 37.7777777777778° Celsius
```

Discussion

Each new code element represents:

- `function input(text)` defines a function that checks the JavaScript environment and reads from the window, the console, or standard input as appropriate.

References

- Wikiversity: Computer Programming

Python Examples

DAVE BRAUNSCHWEIG

Overview

The following examples demonstrate data types, arithmetic operations, and input in Python.

Data Types

```
# This program demonstrates variables, literal constants, and data types.

i = 1234567890
f = 1.23456789012345
s = "string"
b = True

print("Integer i =", i)
print("Float f =", f)
print("String s =", s)
print("Boolean b =", b)
```

Output

```
Integer i = 1234567890
Float f = 1.23456789012345
String s = string
Boolean b = true
```

Discussion

Each code element represents:

- `#` begins a comment
- `i = , d = , s =, b =` assign literal values to the corresponding variables
- `print()` calls the print function

Arithmetic

```
# This program demonstrates arithmetic operations.

a = 3
b = 2

print("a =", a)
print("b =", b)
print("a + b =", (a + b))
print("a - b =", (a - b))
print("a * b =", a * b)
print("a / b =", a / b)
print("a % b =", (a % b))
```

Output

```
a = 3
b = 2
a + b = 5
a - b = 1
a * b = 6
a / b = 1.5
a % b = 1
```

Discussion

Each new code element represents:

- `+, -, *, /, and %` represent addition, subtraction, multiplication, division, and modulus, respectively.

Temperature

```
# This program converts an input Fahrenheit temperature to Celsius.

print("Enter Fahrenheit temperature:")
fahrenheit = float(input())
```

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

print(str(fahrenheit) + "° Fahrenheit is " + str(celsius) + "° Celsius")
```

Output

```
Enter Fahrenheit temperature:
 100
100.0° Fahrenheit is 37.77777777777778° Celsius
```

Discussion

Each new code element represents:

- `input()` reads the next line from standard input
- `float()` converts the input to a floating-point value

References

- Wikiversity: Computer Programming

Swift Examples

DAVE BRAUNSCHWEIG

Overview

The following examples demonstrate data types, arithmetic operations, and input in Swift.

Data Types

```
// This program demonstrates variables, literal constants, and data types.

var i: Int
var d: Double
var s: String
var b: Bool

i = 1234567890
d = 1.23456789012345
s = "string"
b = true

print("Integer i =", i)
print("Double d =", d)
print("String s =", s)
print("Boolean b =", b)
```

Output

```
Integer i = 1234567890
Double d = 1.23456789012345
String s = string
Boolean b = true
```

Discussion

Each code element represents:

- `//` begins a comment
- `var i: Int` defines an integer variable named i
- `var d: Double` defines a double floating-point variable named d
- `var s: String` defines a string variable named s
- `var b: Bool` defines a Boolean variable named b
- `i = , d = , s =, b =` assign literal values to the corresponding variables
- `print()` calls the print function

Arithmetic

```
// This program demonstrates arithmetic operations.

var a: Int
var b: Int

a = 3
b = 2

print("a =", a)
print("b =", b)
print("a + b =", (a + b))
print("a - b =", (a - b))
print("a * b =", a * b)
print("a / b =", a / b)
print("a % b =", (a % b))
```

Output

```
a = 3
b = 2
a + b = 5
a - b = 1
a * b = 6
a / b = 1
a % b = 1
```

Discussion

Each new code element represents:

- `+, -, *, /, and %` represent addition, subtraction, multiplication, division, and modulus, respectively.

Temperature

```
// This program converts a Fahrenheit temperature to Celsius.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://developer.apple.com/library/content/documentation/Swift/Conc

var fahrenheit: Double
var celsius: Double

print("Enter Fahrenheit temperature:")
fahrenheit = Double(readLine()!)!

celsius = (fahrenheit - 32) * 5 / 9

print(String(fahrenheit) + "° Fahrenheit is " + String(celsius) + "° Celsiu
```

Output

```
Enter Fahrenheit temperature:
 100
100.0° Fahrenheit is 37.7777777777778° Celsius
```

Discussion

Each new code element represents:

- `readline()!` reads the next line from standard input
- `Double()!` converts the input to a double floating-point value
- `String()` converts the output numeric value to a string

References

- Wikiversity: Computer Programming

Practice: Data and Operators

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Review Questions

True or false:

1. A data type defines a set of values and the set of operations that can be applied to those values.
2. Reserved or key words can be used as identifier names.
3. The concept of precedence says that some operators (like multiplication and division) are to be executed before other operators (like addition and subtraction).
4. An operator that needs two operands, will promote one of the operands as needed to make both operands be of the same data type.
5. Parentheses change the precedence of operators.
6. Integer data types are stored with a mantissa and an exponent.
7. Strings are identified by single quote marks in most programming languages.
8. An operand is a value that receives the operator's action.
9. Arithmetic assignment is a shorter way to write some expressions.
10. Integer division is rarely used in computer programming.

Answers:

1. true
2. false
3. true
4. true
5. false – Parentheses change the order of evaluation in an expression.
6. false
7. false
8. true
9. true
10. false

Short Answer:

1. A men's clothing store that caters to the very rich wants to create a database for its customers that records clothing measurements. They need to record information for shoes, socks, pants, dress shirts and casual shirts. HINT: You may need more than 5 data items.
2. The sequence operator can be used when declaring multiple identifier names for variables or constants of the same data type. Is this a good or bad programming habit and why?

Activities

Complete the following activities using pseudocode, a flowcharting tool, or your selected programming language. Use appropriate data types for each variable, and include separate

statements for input, processing, and output. 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 calculate and display their weekly, monthly, and annual gross pay (hours * rate). Base monthly and annual calculations on 12 months per year and 52 weeks per year.[1]
2. Create a program that asks the user how old they are in years, and then calculate and display their approximate age in months, days, hours, and seconds. For example, a person 1 year old is 12 months old, 365 days old, etc.
3. Review MathsIsFun: US Standard Lengths. Create a program that asks the user for a distance in miles, and then calculate and display the distance in yards, feet, and inches, or ask the user for a distance in miles, and then calculate and display the distance in kilometers, meters, and centimeters.
4. Review MathsIsFun: Area of Plane Shapes. Create a program that asks the user for the dimensions of different shapes and then calculate and display the area of the shapes. Do not include shape choices. That will come later. For now, just include multiple shape calculations in sequence.
5. Create a program that calculates the area of a room to determine the amount of floor covering required. The room is rectangular with the dimensions measured in feet with decimal fractions. The output needs to be in square yards. There are 3 linear feet (9 square feet) to a yard.
6. Create a program that helps the user determine how much paint is required to paint a room and how much it will cost. Ask the user for the length, width, and height of a room, the price of a gallon of paint, and the number of square feet that a gallon of paint will cover. Calculate the total area of the four walls as `2 * length * height + 2 * width * height` Calculate the number of gallons as: `total area / square feet per gallon` Note: You must round up to the next full gallon. To round up, add `0.9999` and then convert the resulting value to an integer. Calculate the total cost of the paint as: `gallons * price per gallon`.
7. 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, based on the popular myth that one human year equals seven dog years. Be sure to include the dog's name in the output, such as:
   `Spike is 14 years old in dog years.`

References

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

1. PythonLearn: Variables, expressions, and statements

PART III

FUNCTIONS

Overview

This chapter introduces modular programming, functions, parameters, return values, and scope.

Chapter Outline

- Modular Programming
- Hierarchy or Structure Chart
- Function Examples
- Parameters and Arguments
- Call by Value vs. Call by Reference
- Return Statement
- Void Data Type
- Scope
- Programming Style
- Standard Libraries
- Code Examples
  - C++
  - C#
  - Java
  - JavaScript
  - Python
  - Swift
- Practice

Learning Objectives

1. Understand key terms and definitions.
2. Given example pseudocode, flowcharts, and source code, create a program that uses functions, parameters, and return values to solve a given problem.

Modular Programming

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

Modular programming is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.[1]

Concept of Modularization

One of the most important concepts of programming is the ability to group some lines of code into a unit that can be included in our program. The original wording for this was a sub-program. Other names include: macro, sub-routine, procedure, module and function. We are going to use the term **function** for that is what they are called in most of the predominant programming languages of today. Functions are important because they allow us to take large complicated programs and to divide them into smaller manageable pieces. Because the function is a smaller piece of the overall program, we can concentrate on what we want it to do and test it to make sure it works properly. Generally, functions fall into two categories:

1. **Program Control** – Functions used to simply sub-divide and control the program. These functions are unique to the program being written. Other programs may use similar functions, maybe even functions with the same name, but the content of the functions are almost always very different.
2. **Specific Task** – Functions designed to be used with several programs. These functions perform a specific task and thus are usable in many different programs because the other programs also need to do the specific task. Specific task functions are sometimes referred to as building blocks. Because they are already coded and tested, we can use them with confidence to more efficiently write a large program.

The main program must establish the existence of functions used in that program. Depending on the programming language, there is a formal way to:

1. define a function (its **definition** or the code it will execute)
2. **call** a function
3. declare a function (a **prototype** is a declaration to a compiler)

> Note: Defining and calling functions are common activities across programming languages.

1. Wikipedia: Modular programming

Declaring functions with prototypes is specific to certain programming languages, including C and C++.

Program Control functions normally do not communicate information to each other but use a common area for variable storage. Specific Task functions are constructed so that data can be communicated between the calling program piece (which is usually another function) and the function being called. This ability to communicate data is what allows us to build a specific task function that may be used in many programs. The rules for how the data is communicated in and out of a function vary greatly by programming language, but the concept is the same. The data items passed (or communicated) are called parameters. Thus the wording: **parameter passing**. The four data communication options include:

1. no communication in with no communication out
2. no communication in with some communication out
3. some communication in with some communication out
4. some communication in with no communication out

Program Control Function

The main program piece in many programming languages is a special function with the **identifier name** of main. The special or uniqueness of main as a function is that this is where the program starts executing code and this is where it usually stops executing code. It is often the first function defined in a program and appears after the area used for includes, other technical items, declaration of prototypes, the listing of global constants and variables and any other items generally needed by the program. The code to define the function main is provided; however, it is not prototyped or usually called like other functions within a program.

Specific Task Function

We often have the need to perform a specific task that might be used in many programs.
General layout of a function in a statically-typed language such as C++, C#, and Java:

```
<return value data type> function identifier name(<data type> <identifier n
    //lines of code;
    return <value>;
}
```

General layout of a function in a dynamically typed language such as JavaScript and Python:

```
function identifier name(<identifier name for input value>) {
    //lines of code;
    return <value>;
}
```

```
def function identifier name(<identifier name for input value>):
    //lines of code
    return <value>
```

In some programming languages, functions have a set of **braces** {} used for identifying a group or block of statements or lines of code. Other languages use indenting or some type of begin and end statements to identify a code block. There are normally several lines of code within a function.

Programming languages will either have specific task functions defined before or after the main function, depending on coding conventions for the given language.

When you call a function you use its identifier name and a set of parentheses. You place any data items you are passing inside the parentheses. After our program is compiled and running, the lines of code in the main function are executed, and when it gets to the calling of a specific task function, the control of the program moves to the function and starts executing the lines of code in the function. When it's done with the lines of code, it will return to the place in the program that called it (in our example the function main) and continue with the code in that function.

Program Layout

Most programs have several items before the functions, including:

1. Documentation – Most programs have a comment area at the start of the program with a variety of comments pertinent to the program.
2. Include or import statements used to access standard library functions.
3. Language-specific code such as namespace references or function prototypes.
4. Global or module-level constants and variables, when required.

Key Terms

braces
Used to identify a block of code in languages such as C++, C#, Java, and JavaScript.

function
What modules are called in many predominant programming languages of today.

function call
A function's using or invoking of another function.

function definition
The code that defines what a function does.

function prototype
A function's communications declaration to a compiler.

identifier name
The name given by the programmer to identify a function or other program items such as variables.

modularization
The ability to group some lines of code into a unit that can be included in our program.

parameter passing
How the data is communicated in to and out of a function.

program control
: Functions used to simply subdivide and control the program.

specific task
: Functions designed to be used with several programs.

References

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

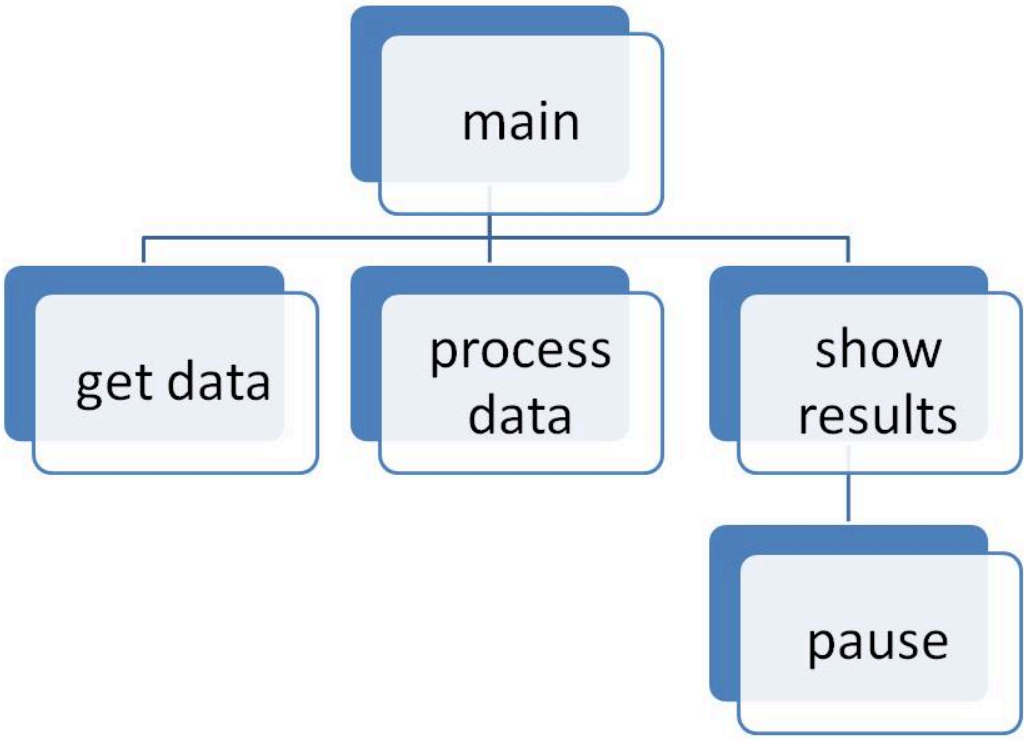
Hierarchy or Structure Chart

KENNETH LEROY BUSBEE

Overview

The **hierarchy chart** (also known as a **structure chart**) shows the relationship between various modules. Its name comes from its general use in showing the organization (or structure) of a business. The President at the top, then vice presidents on the next level, etc. Within the context of a computer program, it shows the relationship between modules (or functions). Detail logic of the program is not presented. It does represent the organization of the functions used within the program showing which functions are calling on a subordinate function. Those above are calling those on the next level down.

Hierarchy charts are created by the programmer to help document a program. They convey the big picture of the modules (or functions) used in a program.



Hierarchy or Structure chart for a program that has five functions.

Key Terms

hierarchy chart
Convey the relationship or big picture of the various functions in a program.
structure chart
Another name for a hierarchy chart.

References

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

Function Examples

DAVE BRAUNSCHWEIG

Overview

The following pseudocode and flowchart examples take the Temperature program from the previous chapter and separate the functionality into independent functions for input, processing, and output, as GetFahrenheit, CalculateCelsius, and DisplayResult, respectively.

Discussion

As independent functions, each function acts as a miniature program, with its own input, processing, and output. As you review the following code, note which functions have parameters (input) and which functions have return values (output). Parameters and return values will be discussed in the next few pages.

Function	Purpose	Parameters (input)	Return Value (output)
Main	main program	none	none
GetFahrenheit	input	none	fahrenheit
CalculateCelsius	processing	fahrenheit	celsius
DisplayResult	output	fahrenheit, celsius	none

Pseudocode

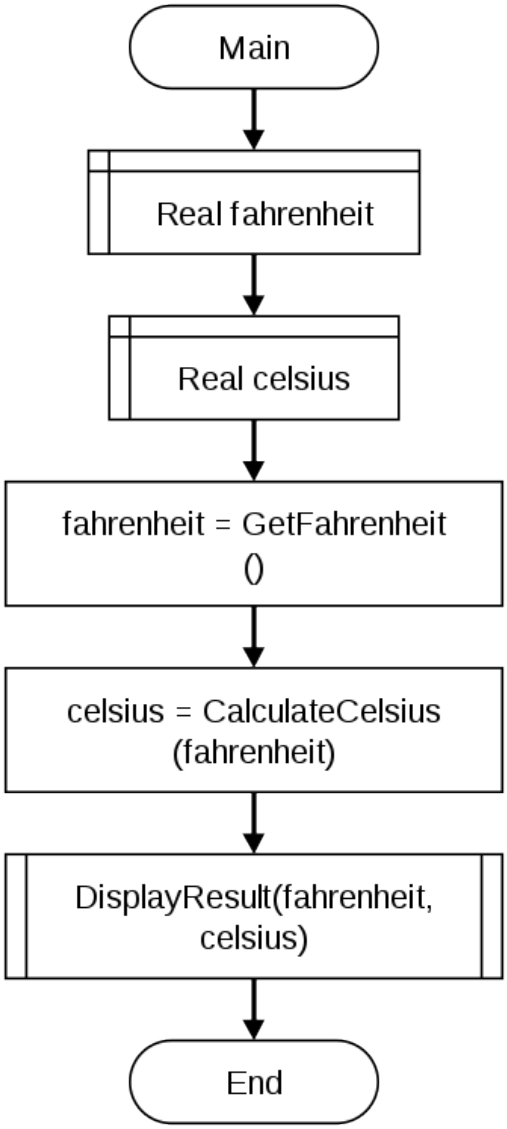
```
Function Main
    ... This program asks the user for a Fahrenheit temperature,
    ... converts the given temperature to Celsius,
    ... and displays the results.

    Declare Real fahrenheit
    Declare Real celsius

    Assign fahrenheit = GetFahrenheit()
    Assign celsius = CalculateCelsius(fahrenheit)
    Call DisplayResult(fahrenheit, celsius)
End

Function GetFahrenheit
    Declare Real fahrenheit
```

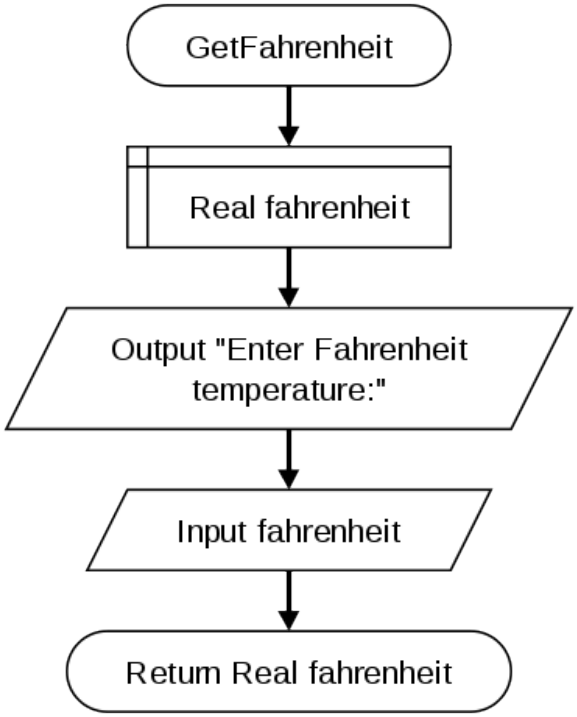
```
    Output "Enter Fahrenheit temperature:"
    Input fahrenheit
Return Real fahrenheit

Function CalculateCelsius (Real fahrenheit)
    Declare Real celsius

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

Function DisplayResult (Real fahrenheit, Real celsius)
    Output fahrenheit & "° Fahrenheit is " & celsius & "° Celsius"
End
```

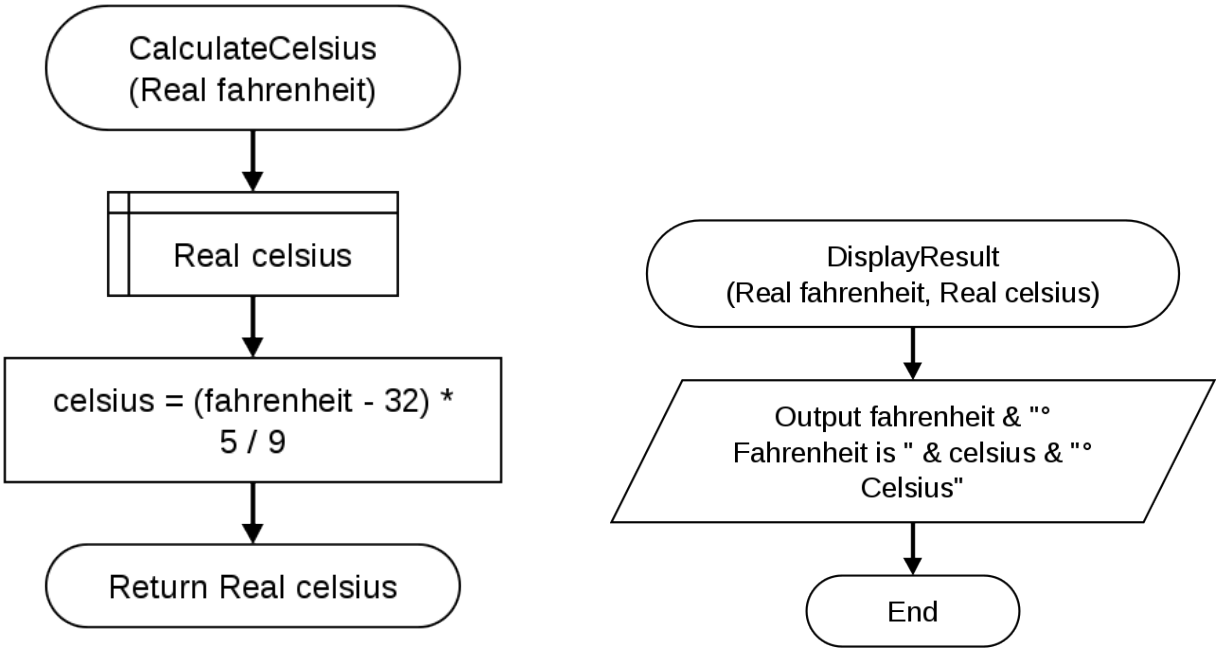
Output

```
Enter Fahrenheit temperature:
 100
100° Fahrenheit is 37.7777777777778° Celsius
```

Flowchart


Main
Real fahrenheit
Real celsius
fahrenheit = GetFahrenheit ()
celsius = CalculateCelsius (fahrenheit)
DisplayResult(fahrenheit, celsius)
End


GetFahrenheit
Real fahrenheit
Output "Enter Fahrenheit temperature:"
Input fahrenheit
Return Real fahrenheit



References

- Wikiversity: Computer Programming
- Flowgorithm – Flowchart Programming Language

Parameters and Arguments

DAVE BRAUNSCHWEIG

Overview

A **parameter** is a special kind of variable used in a function to refer to one of the pieces of data provided as input to the function. These pieces of data are the values of the **arguments** with which the function is going to be called/invoked. An ordered list of parameters is usually included in the definition of a function, so that, each time the function is called, its arguments for that call are evaluated, and the resulting values can be assigned to the corresponding parameters.[1]

Discussion

Recall that the modular programming approach separates the functionality of a program into *independent* modules. To separate the functionality of one function from another, each function is given its own unique input variables, called parameters. The parameter values, called arguments, are passed to the function when the function is called. Consider the following function pseudocode:

```
Function CalculateCelsius (Real fahrenheit)
    Declare Real celsius

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

If the CalculateCelsius function is called passing in the value 100, as in `CalculateCelsius(100)`, the parameter is `fahrenheit` and the argument is `100`. The terms parameter and argument are often used interchangeably. However, parameter refers to the variable identifier (fahrenheit) while argument refers to the variable value (100).

Functions may have no parameters or multiple parameters. Consider the following function pseudocode:

```
Function DisplayResult (Real fahrenheit, Real celsius)
    Output fahrenheit & "° Fahrenheit is " & celsius & "° Celsius"
End
```

If the DisplayResult function is called passing in the values 98.6 and 37.0, as in `DisplayResults(98.6, 37.0)`, the argument or value for the fahrenheit parameter is 98.6 and the argument or value for the celsius parameter is 37.0. Note that the arguments are passed

1. Wikipedia: Parameter (computer programming)

positionally. Calling `DisplayResults(37.0, 98.6)` would result in incorrect output, as the value of fahrenheit would be 37.0 and the value of celsius would be 98.6.

Some programming languages, such as Python, support named parameters. When calling functions using named parameters, parameter names and values are used, and positions are ignored. When names are not used, arguments are identified by position. For example, any of the following function calls would be valid:

```
CalculateCelsius(98.6, 37.0)
CalculateCelsius(fahrenheit=98.6, celsius=37.0)
CalculateCelsius(celsius=37.0, fahrenheit=98.6)
```

Key Terms

argument
A value provided as input to a function.

parameter
A variable identifier provided as input to a function.

References

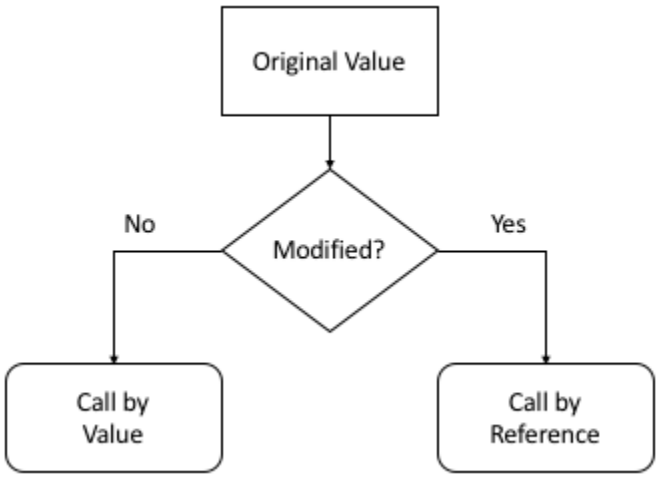
- Wikiversity: Computer Programming

Call by Value vs. Call by Reference

DAVE BRAUNSCHWEIG

Overview

In **call by value**, a parameter acts within the function as a new local variable initialized to the value of the argument (a local (isolated) copy of the argument). In **call by reference**, the argument variable supplied by the caller can be affected by actions within the called function.[1]



Call-by-value vs. call-by-reference

Discussion

Call by Value

Within most current programming languages, parameters are passed by value by default, with the argument as a copy of the calling value. Arguments are isolated, and functions are free to make changes to parameter values without any risk of impact to the calling function. Consider the following pseudocode:

```
Function Main
    Declare Real fahrenheit

    Assign fahrenheit = 100
    Output "Main fahrenheit = " & fahrenheit
```

1. Wikipedia: Parameter (computer programming)

```
    Call ChangeFahrenheit(fahrenheit)
    Output "Main fahrenheit = " & fahrenheit
End

Function ChangeFahrenheit (Real fahrenheit)
    Output "ChangeFahrenheit fahrenheit = " & fahrenheit
    Assign fahrenheit = 0
    Output "ChangeFahrenheit fahrenheit = " & fahrenheit
End
```

Output

```
Main fahrenheit = 100
ChangeFahrenheit fahrenheit = 100
ChangeFahrenheit fahrenheit = 0
Main fahrenheit = 100
```

In English, the Main function assigns the value 100 to the variable fahrenheit, displays that value, and then calls ChangeFahrenheit passing a copy of that value. The called function displays the argument, changes it, and displays it again. Execution returns to the calling function, and Main displays the value of the original variable. With call by value, the variable fahrenheit in the calling function and the parameter fahrenheit in the called function refer to different memory addresses, and the called function cannot change the value of the variable in the calling function.

Call by Reference

If a programming language uses or supports call by reference, the variable in the calling function and the parameter in the called function refer to the same memory address, and the called function may change the value of the variable in the calling function. Using the same code example as above, call by reference output would change to:

```
Main fahrenheit = 100
ChangeFahrenheit fahrenheit = 100
ChangeFahrenheit fahrenheit = 0
Main fahrenheit = 0
```

Programming languages that support both call by value and call by reference use some type of key word or symbol to indicate which parameter passing method is being used.

Language	Call By Value	Call by Reference
C++	default	use `&parameter` in called function
C#	default	use `ref parameter` in calling and called functions
Java	default	applies to arrays and objects
JavaScript	default	applies to arrays and objects
Python	default	applies to arrays (lists) and mutable objects

Arrays and objects are covered in later chapters.

Key Terms

call by reference
: Parameters passed by calling functions may be modified by called functions.

call by value
: Parameters passed by calling functions cannot be modified by called functions.

References

- Wikiversity: Computer Programming

Return Statement

DAVE BRAUNSCHWEIG AND KENNETH LEROY BUSBEE

Overview

A **return statement** causes execution to leave the current function and resume at the point in the code immediately after where the function was called. Return statements in many languages allow a function to specify a return value to be passed back to the code that called the function.[1]

Discussion

The return statement exits a function and returns to the statement where the function was called. Most programming languages support optionally returning a single value to the calling function. Consider the following pseudocode:

```
Function Main
    ...
    Assign fahrenheit = GetFahrenheit()
    ...
End

Function GetFahrenheit
    Declare Real fahrenheit

    Output "Enter Fahrenheit temperature:"
    Input fahrenheit
Return Real fahrenheit
```

In English, the Main function calls the GetFahrenheit function, passing in no parameters. The GetFahrenheit function retrieves input from the user and returns that input back to the main function, where it is assigned to the variable fahrenheit. In this example, the Main function has no return value.

Note that functions are independent, and each function must declare its own variables. While both functions have a variable named fahrenheit, they are not the same variable. Each variable refers to a different location in memory. Just as parameters by default are passed by position rather than by name, return values are also passed by position rather than by name. The following code would generate the same results.

1. Wikipedia: Return statement

```
Function Main
    ...
    Assign fahrenheit = GetTemperature()
    ...
End

Function GetTemperature
    Declare Real temperature

    Output "Enter Fahrenheit temperature:"
    Input temperature
Return Real temperature
```

Most programming languages support either zero or one return value from a function. There are some older programming languages where return values are not supported. In those languages, the modules are often referred to as subroutines rather than functions. There are also programming languages that support multiple return values in a single return statement, however, only single return values or no return value will be used in this book.

Key Terms

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

References

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

Void Data Type

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

The void data type, similar to the Nothing data type described earlier, is the data type for the result of a function that returns normally, but does not provide a result value to its caller.[1]

Discussion

The **void data type** has no values and no operations. It's a data type that represents the lack of a data type.

Language	Reserved Word
C++	`void`
C#	`void`
Java	`void`
JavaScript	`void`
Python	N/A
Swift	`Void`

Many programming languages need a data type to define the lack of return value to indicate that nothing is being returned. The void data type is typically used in the definition and prototyping of functions to indicate that either nothing is being passed in and/or nothing is being returned.

Key Terms

void data type
A data type that has no values or operators and is used to represent nothing.

References

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

1. Wikipedia: Void type

Scope

KENNETH LEROY BUSBEE

Overview

The **scope** of an identifier name binding – an association of a name to an entity, such as a variable – is the region of a computer program where the binding is valid: where the name can be used to refer to the entity. Such a region is referred to as a scope block. In other parts of the program, the name may refer to a different entity (it may have a different binding), or to nothing at all (it may be unbound).[1]

Discussion

Scope is the area of the program where an item (be it variable, constant, function, etc.) that has an identifier name is recognized. In our discussion, we will use a variable and the place within a program where the variable is defined determines its scope.

Global scope (and by extension global data storage) occurs when a variable is defined "outside of a function". When compiling the program it creates the storage area for the variable within the program's **data area as part of the object code**. The object code has a machine code piece, a data area, and linker resolution instructions. Because the variable has global scope it is available to all of the functions within your source code. It can even be made available to functions in other object modules that will be linked to your code; however, we will forgo that explanation now. A key wording change should be learned at this point. Although the variable has global scope, technically it is available only from **the point of definition to the end of the program source code**. That is why most variables with global scope are placed near the top of the source code before any functions. This way they are available to all of the functions.

Local scope (and by extension local data storage) occurs when a variable is defined "inside of a function". When compiling, the compiler creates machine instructions that will direct the creation of storage locations on an area known as the **stack which is part of the computer's memory**. These memory locations exist until the function completes its task and returns to its calling function. In assembly language, we talk about items being pushed onto the stack and popped off the stack when the function terminates. Thus, the stack is a reusable area of memory being used by all functions and released as functions terminate. Although the variable has local scope, technically it is available only from **the point of definition to the end of the function**. The parameter passing of data items into a function establishes them as local variables. Additionally, any other variables or constants needed by the function usually occur near the top of the function definition so that they are available during the entire execution of the function's code.

Scope is an important concept for modularization. Program control functions may use global scope for variables and constants placing them near the top of the program before any functions. Specific task functions use only local scope variables by passing data as needed into the function with parameter passing and creating local variables and constants as needed. Any information that needs to be communicated back to the calling function is again done via parameter passing. This **closed communications model** that passes all data into and out of a

1. Wikipedia: Scope (computer science)

function creates an important predecessor concept for **encapsulation** which is used in object-oriented programming.

Key Terms

data area
A part of an object code file used for storage of data.
global scope
Data storage defined outside of a function.
local scope
Data storage defined inside of a function.
scope
The area of a source code file where an identifier name is recognized.
stack
A part of the computer's memory used for storage of data.

References

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

Programming Style

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

Programming style is a set of rules or guidelines used when writing the source code for a computer program. Following a particular programming style will help programmers read and understand source code conforming to the style, and help to avoid introducing errors.[1]

Discussion

Within the programming industry there is a desire to make software programs easy to maintain. The desire centers on money. Simply put, it costs less money to maintain a well written program. One important aspect of program maintenance is making source code listings clear and as easy to read as possible. To that end we will consider the following:

1. Documentation
2. Vertical Alignment
3. Comments
4. Indentation
5. Meaningful Identifier Names Consistently Typed
6. Appropriate use of Typedef

The above items are not needed in order for the source code to compile. Technically the compiler does not read the source code the way humans read the source code. But that is exactly the point; the desire is to make the source code easier for humans to read. You should not be confused between what is possible (technically will run) and what is okay (acceptable good programming practice that leads to readable code).

For each of these items, check style guides for your selected programming language to determine standards and best practices. The following are general guidelines to consider.

Documentation

Documentation is usually placed at the top of the program using several comment lines. The amount of information would vary based on the requirements or standards of the company who is paying its employees or independent contractors to write the code.

Vertical Alignment

You see this within the documentation area. All of the items are aligned up within the same

1. Wikipedia: Programming style

column. This vertical alignment occurs again when variables are defined. When declaring variables or constants many textbooks put several items on one line; like this:

```
float  length, width, height;
```

However common this is in textbooks, it would generally not be acceptable to standards used in most companies. You should declare each item on its own line; like this:

```
float  length;
float  width;
float  height;
```

This method of using one item per line is more readable by humans. It is quicker to find an identifier name because you can read the list vertically faster than searching horizontally. Some programmers list them in alphabetic order.

The lines of code inside functions are also aligned vertically and typically indented two or four spaces from the left. The indentation helps set the block off visually.

Comments

Experts have varying viewpoints on whether, and when, comments are appropriate in source code. Some assert that source code should be written with few comments, on the basis that the source code should be self-explanatory or self-documenting. Others suggest code should be extensively commented, with over 50% of the non-whitespace characters in source code being contained within comments).[2]

In between these views is the assertion that comments are neither beneficial nor harmful by themselves, and what matters is that they are correct and kept in sync with the source code, and omitted if they are superfluous, excessive, difficult to maintain or otherwise unhelpful.[3]

Indentation

For languages that use curly braces, there are two common indentation styles:

```
function(parameters) {
    // code
}
```

```
function(parameters)
{
    // code
```

2. Wikipedia: Comment (computer programming)
3. Wikipedia: Comment (computer programming)

```
}
```

In either case, it is important to maintain vertical alignment between the start of the code block and the closing curly brace.

The number of spaces used for indenting blocks of code is typically two or four spaces. Care should be taken to ensure that the IDE or code editor inserts spaces rather than tab characters for indents.

Meaningful Identifier Names Consistently Typed

As the name implies "identifier names" should clearly identify who (or what) you are talking about. Calling your spouse "Snooky" may be meaningful to only you. Others might need to see her full name (Jane Mary Smith) to appropriately identify who you are talking about. The same concept in programming is true. Variables, constants, functions, and other identifiers should use meaningful names. Additionally, those names should be typed consistently in terms of upper and lower case as they are used in the program. Don't define a variable as: Pig and then type it later on in your program as: pig.

A good rule of thumb for identifiers in procedural programs (as opposed to object-oriented programs) is to use verb-noun combinations for function identifiers and use noun or adjective-noun combinations for constant and variable identifiers. If a function name requires two verbs or two nouns to fully describe the function, it should probably be split into separate functions.

Key Terms

braces
Used to identify a block of code in languages such as C++, C#, Java, and JavaScript.

consistent
A rule that says to type identifier names in upper and lower case consistently throughout your source code.

comments
Information inserted into a source code file for documentation of the program.

documentation
A method of preserving information useful to others in understanding an information system or part thereof.

indention
A method used to make sections of source code more visible.

meaningful
A rule that says identifier names must be easily understood by another reading the source code.

vertical alignment
A method of listing items vertically so that they are easier to read quickly.

References

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

Standard Libraries

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

Many common or standard functions, whose definitions have already been written, are ready to be used in any program. They are organized into a group of functions (think of them as several books) and are collectively called a **standard library**. There are many functions organized into several libraries For example, within most programming languages many math functions exist and have been coded (and placed into libraries). These functions were written by programmers and tested to ensure that they work properly. In most cases, the functions were reviewed by several people to double and triple check to ensure that they did what was expected. We have the advantage of using these functions with confidence that they will work properly in our programs, thus saving us time and money.

Discussion

The main program must establish the existence of functions used in that program. Depending on the programming language, there is a formal way to:

1. define a function
2. declare a function (a prototype is a declaration to a compiler)
3. call a function

When we create functions in our program, we usually see them in the following order in our source code listing:

1. declare the function (prototype)
2. call the function
3. define the function

When we use functions created by others that have been organized into a library, we include a header file in our program which contains the prototypes for the functions. Just like functions that we create, we see them in the following order in our source code listing:

1. declaring the function (prototype provided in the include file)
2. call the function (with parameter passing of values)
3. define the function (it is either defined in the header file or the linker program provides the actual object code from a Standard Library object area)

In most cases, the user can look at the prototype and understand exactly how the communications (parameter passing) into and out of the function will occur when the function is called. Let's look at the math example of absolute value.

Language	Example
C++	`#include <cmath>` `std::abs(number);`
C#	`Math.Abs(number);`
Java	`Java.lang.Math.abs(number)`
JavaScript	`Math.abs(number);`
Python	`abs(number)`
Swift	`abs(number)`

Not wanting to have a long function name the designers named it: **abs** instead of "absolute". This might seem to violate the identifier naming rule of using meaningful names, however, when identifier names are established for standard libraries they are often shortened to a name that is easily understood by all who would be using them. If I had two integer variables named apple and banana; and I wanted to store the absolute value of banana into apple; then a line of code to call this function would be:

```
apple = abs(banana);
```

Let's say it in English, pass the function absolute the value stored in variable banana and assign the returning value from the function to the variable apple. Thus, if you know the prototype you can usually properly call the function and use its returning value (if it has one) without ever seeing the definition of the code (i.e. the source code that tells the function how to get the answer; that is written by someone else; and either included in the header file or compiled and placed into an object library; and linked during the linking step of the Integrated Development Environment (IDE).

Key Terms

abs
A function within a standard library which stands for absolute value.

confidence
The reliance that Standard Library functions work properly.

standard library
A set of specific task functions that have been added to the programming language for universal use.

References

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

C++ 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://en.wikibooks.org/wiki/C%2B%2B_Programming

#include <iostream>

using namespace std;

double getFahrenheit();
double calculateCelsius(double);
void displayResult(double, double);

int main() {
    double fahrenheit;
    double celsius;

    fahrenheit = getFahrenheit();
    celsius = calculateCelsius(fahrenheit);
    displayResult(fahrenheit, celsius);

    return 0;
}

double getFahrenheit() {
    double fahrenheit;

    cout << "Enter Fahrenheit temperature:" << endl;
    cin >> fahrenheit;

    return fahrenheit;
}
```

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

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

    return celsius;
}

void displayResult(double fahrenheit, double celsius) {
    cout << fahrenheit << "° Fahrenheit is "
        << celsius << "° Celsius" << endl;
}
```

Output

```
Enter Fahrenheit temperature:
 100
100° Fahrenheit is 37.7778° Celsius
```

References

- Wikiversity: Computer Programming

C# 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://en.wikibooks.org/wiki/C_Sharp_Programming

using System;

class Temperature
{
    public static void Main (string[] args)
    {
        double fahrenheit;
        double celsius;

        fahrenheit = GetFahrenheit();
        celsius = CalculateCelsius(fahrenheit);
        DisplayResult(fahrenheit, celsius);
    }

    private static double GetFahrenheit()
    {
        string input;
        double fahrenheit;

        Console.WriteLine("Enter Fahrenheit temperature:");
        input = Console.ReadLine();
        fahrenheit = Convert.ToDouble(input);

        return fahrenheit;
    }

    private static double CalculateCelsius(double fahrenheit)
```

```
    {
        double celsius;

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

        return celsius;
    }

    private static void DisplayResult(double fahrenheit, double celsius)
    {
        Console.WriteLine(fahrenheit.ToString() + "° Fahrenheit is " +
            celsius.ToString() + "° Celsius");
    }
}
```

Output

```
Enter Fahrenheit temperature:
 100
100° Fahrenheit is 37.7777777777778° Celsius
```

References

- Wikiversity: Computer Programming

Java 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://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) {
        double fahrenheit;
        double celsius;

        fahrenheit = getFahrenheit();
        celsius = calculateCelsius(fahrenheit);
        displayResult(fahrenheit, celsius);
    }

    private static double getFahrenheit() {
        double fahrenheit;

        System.out.println("Enter Fahrenheit temperature:");
        fahrenheit = input.nextDouble();

        return fahrenheit;
    }

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

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

```
        return celsius;
    }

    private static void displayResult(double fahrenheit, double celsius) {
        System.out.println(fahrenheit + "° Fahrenheit is " +
            celsius + "° Celsius");
    }
}
```

Output

```
Enter Fahrenheit temperature:
 100
100° Fahrenheit is 37.7777777777778° Celsius
```

References

- Wikiversity: Computer Programming

JavaScript 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://en.wikibooks.org/wiki/JavaScript

main();

function main() {
  var fahrenheit = getFahrenheit();
  var celisus = calculateCelsius(fahrenheit);
  displayResult(fahrenheit, celisus);
}

function getFahrenheit() {
  var fahrenheit = input("Enter Fahrenheit temperature:");
  return fahrenheit;
}

function calculateCelsius(fahrenheit) {
  var celisus = (fahrenheit - 32) * 5 / 9;
  return celisus;
}

function displayResult(fahrenheit, celisus) {
  output(fahrenheit + "° Fahrenheit is " +
    celisus + "° Celsius");
}

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 Fahrenheit temperature:
 100
100° Fahrenheit is 37.7777777777778° Celsius
```

References

- Wikiversity: Computer Programming

Python 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://en.wikibooks.org/wiki/Python_Programming


def get_fahrenheit():
    print("Enter Fahrenheit temperature:")
    fahrenheit = float(input())
    return fahrenheit


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


def display_result(fahrenheit, celsius):
    print(str(fahrenheit) + "° Fahrenheit is " +
        str(celsius) + "° Celsius")


def main():
    fahrenheit = get_fahrenheit()
    celsius = calculate_celsius(fahrenheit)
    display_result(fahrenheit, celsius)


main()
```

Output

```
Enter Fahrenheit temperature:
 100
100.0° Fahrenheit is 37.77777777777778° Celsius
```

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 getFahrenheit() -> Double {
    var fahrenheit: Double

    print("Enter Fahrenheit temperature:")
    fahrenheit = Double(readLine(strippingNewline: true)!)!

    return fahrenheit
}

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

    celsius = (fahrenheit - 32) * 5 / 9

    return celsius
}

func displayResult(fahrenheit: Double, celsius: Double) {
    print(String(fahrenheit) + "° Fahrenheit is " + String(celsius) + "° Ce
}

func main() {
    var fahrenheit: Double
    var celsius: Double

    fahrenheit = getFahrenheit()
    celsius = calculateCelsius(fahrenheit:fahrenheit)
```

```
    displayResult(fahrenheit:fahrenheit, celsius:celsius)
}

main()
```

Output

```
Enter Fahrenheit temperature:
 100
100.0° Fahrenheit is 37.7777777777778° Celsius
```

References

- Wikiversity: Computer Programming

Practice: Functions

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Review Questions

True / False

1. In addition to the term function as the name of a subprogram, the computer industry also uses macro, procedure and module.
2. Generally, functions fall into two categories: Program Control and Specific Task.
3. Hierarchy Charts and Structure Charts are basically the same thing.
4. Program Control functions are used to simply subdivide and control the program.
5. The void data type is rarely used in C++.
6. Making source code readable is only used by beginning programmers.
7. Scope refers to a brand of mouthwash.
8. User-defined specific task functions are usually placed into a user-defined library.
9. Local and global data storage is associated with the concept of scope.
10. Creating a header file for user-defined specific task functions is a difficult task.
11. The stack is part of the computer's memory used for storage of data.
12. The standard library is a set of specific task functions that have been added to the programming language for universal use.
13. Programmers should not have confidence that standard library functions work properly.
14. It would be easier to write programs without using specific task functions.

Answers:

1. true
2. true
3. true
4. true
5. false
6. false
7. false – Although Scope is a brand of mouthwash; we are looking for the computer-related definition.
8. true
9. true
10. false – It may seem difficult at first, but with a little practice it is really quite easy.
11. true
12. true
13. false
14. false

Short Answer

1. Create a hierarchy chart for the function example program found in this chapter.
2. Review the programs you have already created for this course. Based on coding standards for

your selected programming language, identify some problems that make your code "undocumented", "unreadable" or wrong in some other way.

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 calculate and display their weekly, monthly, and annual gross pay (hours * rate). Base monthly and annual calculations on 12 months per year and 52 weeks per year.[1]
2. Create a program that asks the user how old they are in years, and then calculate and display their approximate age in months, days, hours, and seconds. For example, a person 1 year old is 12 months old, 365 days old, etc.
3. Review MathsIsFun: US Standard Lengths. Create a program that asks the user for a distance in miles, and then calculate and display the distance in yards, feet, and inches, or ask the user for a distance in miles, and then calculate and display the distance in kilometers, meters, and centimeters.
4. Review MathsIsFun: Area of Plane Shapes. Create a program that asks the user for the dimensions of different shapes and then calculate and display the area of the shapes. Do not include shape choices. That will come later. For now, just include multiple shape calculations in sequence.
5. Create a program that calculates the area of a room to determine the amount of floor covering required. The room is rectangular with the dimensions measured in feet with decimal fractions. The output needs to be in square yards. There are 3 linear feet (9 square feet) to a yard.
6. Create a program that helps the user determine how much paint is required to paint a room and how much it will cost. Ask the user for the length, width, and height of a room, the price of a gallon of paint, and the number of square feet that a gallon of paint will cover. Calculate the total area of the four walls as `2 * length * height + 2 * width * height` Calculate the number of gallons as: `total area / square feet per gallon` Note: You must round up to the next full gallon. To round up, add `0.9999` and then convert the resulting value to an integer. Calculate the total cost of the paint as: `gallons * price per gallon` .
7. 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, based on the popular myth that one human year equals seven dog years. Be sure to include the dog's name in the output, such as:
   `Spike is 14 years old in dog years.`

1. PythonLearn: Variables, expressions, and statements

References

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

PART IV

CONDITIONS

Overview

This chapter introduces conditions and selection control structures.

Chapter Outline

- Structured Programming
- Selection Control Structures
- If Then Else
- Code Blocks
- Relational Operators
- Assignment vs. Equality
- Logical Operators
- Nested If Then Else
- Case Control Structure
- Condition Examples
- Code Examples
  - C++
  - C#
  - Java
  - JavaScript
  - Python
  - Swift
- Practice

Learning Objectives

1. Understand key terms and definitions.
2. Given example pseudocode, flowcharts, and source code, create a program that uses conditions and selection control structures to solve a given problem.

Structured Programming

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines in contrast to using simple tests and jumps such as the go to statement, which can lead to "**spaghetti code**" that is potentially difficult to follow and maintain.[1]

Discussion

One of the most important concepts of programming is the ability to control a program so that different lines of code are executed or that some lines of code are executed many times. The mechanisms that allow us to control the flow of execution are called **control structures**. Flowcharting is a method of documenting (charting) the flow (or paths) that a program would execute. There are three main categories of control structures:

- **Sequence** – Very boring. Simply do one instruction then the next and the next. Just do them in a given sequence or in the order listed. Most lines of code are this.

- **Selection** – This is where you select or choose between two or more flows. The choice is decided by asking some sort of question. The answer determines the path (or which lines of code) will be executed.

- **Iteration** – Also known as repetition, it allows some code (one to many lines) to be executed (or repeated) several times. The code might not be executed at all (repeat it zero times), executed a fixed number of times or executed indefinitely until some condition has been met. Also known as looping because the flowcharting shows the flow looping back to repeat the task.

A fourth category describes unstructured code.

- **Branching** – An uncontrolled structure that allows the flow of execution to jump to a different part of the program. This category is rarely used in modular structured programming.

All high-level programming languages have control structures. All languages have the first three categories of control structures (sequence, selection, and iteration). Most have the if then else structure (which belongs to the selection category) and the while structure (which belongs to the iteration category). After these two basic structures, there are usually language variations.

The concept of **structured programming** started in the late 1960's with an article by Edsger Dijkstra. He proposed a "go to less" method of planning programming logic that eliminated the

1. Wikipedia: Structured programming

need for the branching category of control structures. The topic was debated for about 20 years. But ultimately – "By the end of the 20th century nearly all computer scientists were convinced that it is useful to learn and apply the concepts of structured programming."[2]

Key Terms

branching
: An uncontrolled structure that allows the flow of execution to jump to a different part of the program.

control structures
: Mechanisms that allow us to control the flow of execution within a program.

iteration
: A control structure that allows some lines of code to be executed many times.

selection
: A control structure where the program chooses between two or more options.

sequence
: A control structure where the program executes the items in the order listed.

spaghetti code
: A pejorative phrase for unstructured and difficult to maintain source code.[3]

structured programming
: A method of planning programs that avoids the branching category of control structures.

References

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

2. Wikipedia: Structured programming
3. Wikipedia: Spaghetti code

Selection Control Structures

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

In **selection control structures**, conditional statements are features of a programming language which perform different computations or actions depending on whether a programmer-specified Boolean condition evaluates to true or false.[1]

Discussion

The basic attribute of a selection control structure is to be able to select between two or more alternate paths. This is described as either two-way selection or multi-way selection. A question using Boolean concepts usually controls which path is selected. All of the paths from a selection control structure join back up at the end of the control structure, before moving on to the next lines of code in a program.

If Then Else Control Structure

The **if then else** control structure is a two-way selection.

```
If age > 17
    Output "You can vote."
False:
    Output "You can't vote."
End
```

Language	Reserved Words
C++	`if` , `else`
C#	`if` , `else`
Java	`if` , `else`
JavaScript	`if` , `else`
Python	`if` , `elif` , `else`
Swift	`if` , `else`

1. Wikipedia: Conditional (computer programming)

Case Control Structure

The **case** control structure is a multi-way selection. Case control structures compare a given value with specified constants and take action according to the first expression to match.[2]

```
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
```

Language	Reserved Words
C++	`switch`, `case`, `break`, `default`
C#	`switch`, `case`, `break`, `default`
Java	`switch`, `case`, `break`, `default`
JavaScript	`switch`, `case`, `break`, `default`
Python	N/A
Swift	`switch`, `case`, `break` (optional), `default`

Python does not support a case control structure. There are workarounds, but they are beyond the scope of this book.

Key Terms

if then else
: A two-way selection control structure.

case
: A multi-way selection control structure.

References

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

2. Wikipedia: Conditional (computer programming)

If Then Else

KENNETH LEROY BUSBEE

Overview

The **if–then–else** construct, sometimes called if-then, is a two-way selection structure common across many programming languages. Although the syntax varies from language to language, the basic structure looks like:[1]

```
If (boolean condition) Then
    (consequent)
Else
    (alternative)
End If
```

Discussion

We are going to introduce the control structure from the selection category that is available in every high level language. It is called the **if then else** structure. Asking a question that has a true or false answer controls the if then else structure. It looks like this:

```
if the answer to the question is true
    then do this
else because it is false
    do this
```

In most 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:

```
if expression is true
    then do this
else because it is false
    do this
```

Some languages use reserved words of: "if", "then" and "else". Many eliminate the "then". Additionally the "do this" can be tied to true and false. You might see it as:

1. Wikipedia: Conditional (computer programming)

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

And most languages infer the “is true” you might see it as:

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

The above four forms of the control structure are saying the same thing. The else word is often not used in our English speaking today. However, consider the following conversation between a mother and her child.

Child asks, “Mommy, may I go out side and play?”

Mother answers, “If your room is clean then you may go outside and play or else you may go sit on a chair for five minutes as punishment for asking me the question when you knew your room was dirty.”

Let’s note that all of the elements are present to determine the action (or flow) that the child will be doing. Because the question (your room is clean) has only two possible answers (true or false) the actions are **mutually exclusive**. Either the child 1) goes outside and plays or 2) sits on a chair for five minutes. One of the actions is executed; never both of the actions.

One Choice – Implied Two-Way Selection

Often the programmer will want to do something only if the expression is true, that is with no false action. The lack of a false action is also referred to as a “null else” and would be written as:

```
if expression
    action true
else
    do nothing
```

Because the “else do nothing” is implied, it is usually written in short form like:

```
if expression
    action true
```

Key Terms

if then else

A two-way selection control structure.

mutually exclusive
: Items that do not overlap. Example: true or false.

References

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

Code Blocks

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

Overview

A **code block**, sometimes referred to as a compound statement, is a lexical structure of source code which is grouped together. Blocks consist of one or more declarations and statements. A programming language that permits the creation of blocks, including blocks nested within other blocks, is called a block-structured programming language. Blocks are fundamental to structured programming, where control structures are formed from blocks.[1]

Discussion

The Need for a Compound Statement

Within many programming languages, there can be only **one statement listed as the action part of a control structure:**

```
if (expression)
    statement
else
    statement
```

Often, we will want to do more than one statement. This problem is overcome by creating a code block or compound statement. For programming languages that use curly braces {} to designate code blocks, a compound if-then-else statement would be similar to:

```
if(expression)
{
    statement;
    statement;
}
else
{
    statement;
    statement;
}
```

1. Wikipedia: Block (programming)

Because programmers often forget that they can have **only one statement listed as the action part of a control structure**; the programming industry encourages the use of indentation (to see the action parts clearly) and the use of compound statements (braces) **always**, even when there is only one statement. Thus:

```
if(expression)
{
    statement;
}
else
{
    statement;
}
```

By writing code in this manner, if the programmer modifies the code by adding more statements to either the action true or the action false; they will not introduce either compiler or logic errors. Using indentation and braces should become standard practice in any language that requires the use of compound statements with control structures.

Indentation and End Block

Other programming languages require explicit designation of code blocks through either indentation or some type of end block statement. For example, Python uses indentation to indicate the statements in a code block:

```
if expression:
    statement
    statement
else:
    statement
    statement
```

Lua uses an end block reserved word:

```
if expression then
    statement
    statement
else
    statement
    statement
end
```

The general if-then-else structure in each of these programming languages is similar, as is the required or expected indentation. The difference is in the syntax used to designate the code blocks.