Programming Fundamentals

Programming Fundamentals

A Modular Structured Approach, 2nd Edition

DAVE BRAUNSCHWEIG AND KENNETH LEROY BUSBEE

Contents

Chapter II. Data and Operators

Chapter V. Loops

Chapter VI. Arrays

Chapter VII. Strings and Files

Contents

Chapters

About this Book

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:

.

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

maintained and you distribute your contributions under the same license.

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

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

Author Acknowledgements

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

- archive.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

CHAPTER 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

- Systems Development Life Cycle
- Program Design
- Program Quality
- Pseudocode
- Flowcharts
- Software Testing
- Integrated Development Environment
- Version Control
- Input and Output
- Hello World
- Code Examples
  - C++
  - C#
  - Java

    - JavaScript
    - Python
    - Swift
- Practice

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

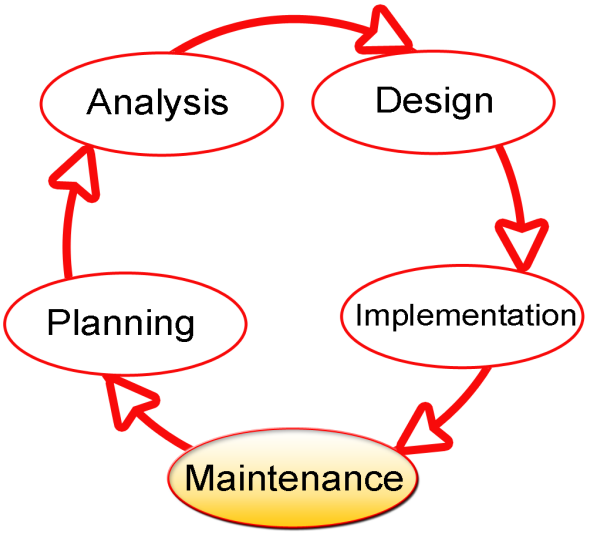
The Systems Development Life Cycle is the big picture of creating an information system that handles a major task (referred to as an application). The **applications** usually consist of many programs. An example would be the Department of Defense supply system, the customer system used at your local 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

1. Wikipedia: Systems development life cycle

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.



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

- archive.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

- archive.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
```

1. Wikipedia: Pseudocode

```
    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 di

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 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

- archive.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.

1. Wikipedia: Flowchart

Simple Flowcharting Symbols

Terminal

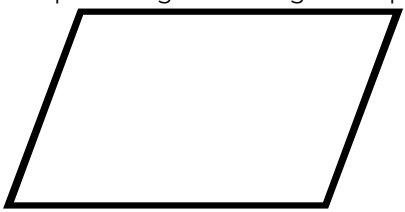
The rounded rectangles, or terminal points, indicate the flowchart's starting and ending points.

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.

Input/Output

The parallelograms designate input or output operations.

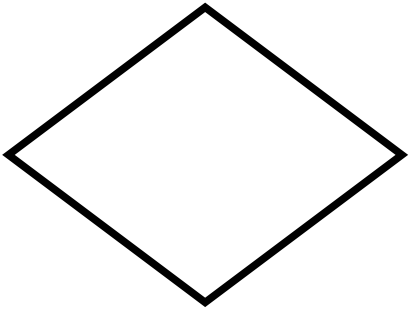


Process

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

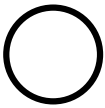
Module Call

A program module is represented in a flowchart by rectangle with some lines to distinguish it from process symbol.

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

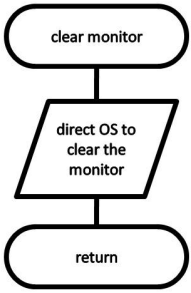
Simple Examples

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
```

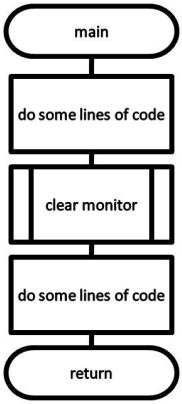


Function clear monitor

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

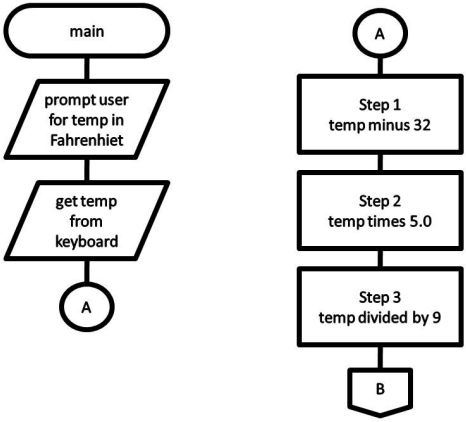
Sequence Control Structures

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.

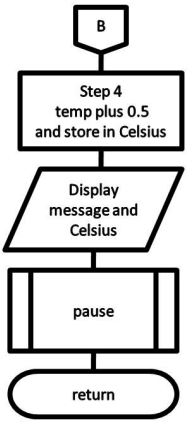
pseudocode: Sequence control structure

```
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 Fahren
    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



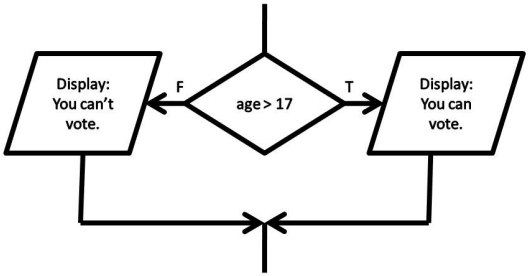
Sequence control structured continued

Advanced Examples

Selection Control Structures

pseudocode: If then Else

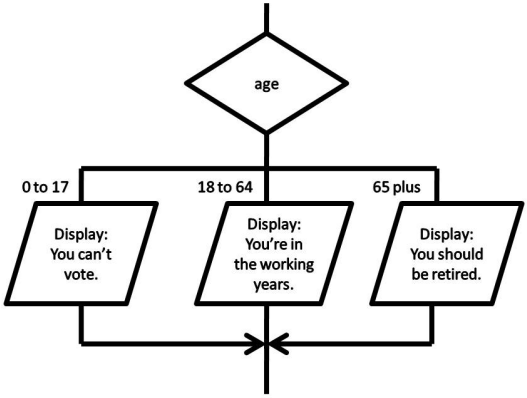
```
If age > 17
    Display a message indicating you can vote.
Else
    Display a message indicating you can't vote.
Endif
```



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
```

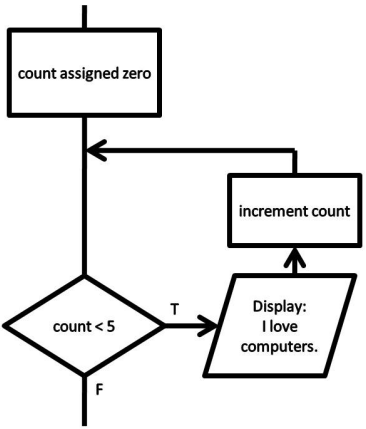


Case control structure

Iteration (Repetition) Control Structures

pseudocode: While

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

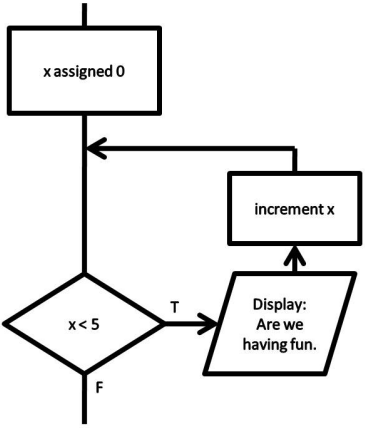


While control structure

pseudocode: For

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

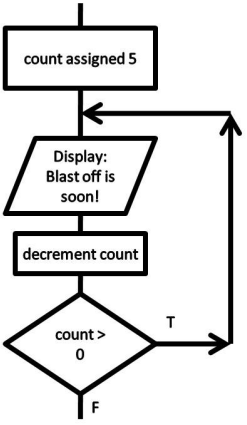
The for loop does not have a standard flowcharting method and you will find it done in different ways. The for loop as a counting loop can be flowcharted similar to the while loop as a counting loop.



For control structure

pseudocode: Do While

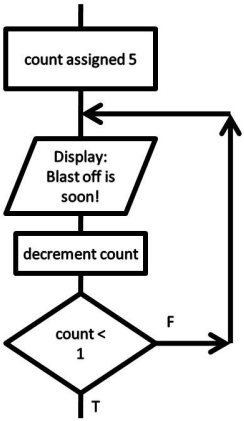
```
count assigned five
Do
    Display "Blast off is soon!"
    Decrement count
While count > zero
```



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

- archive.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

1. Wikipedia: Software testing

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

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

- archive.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

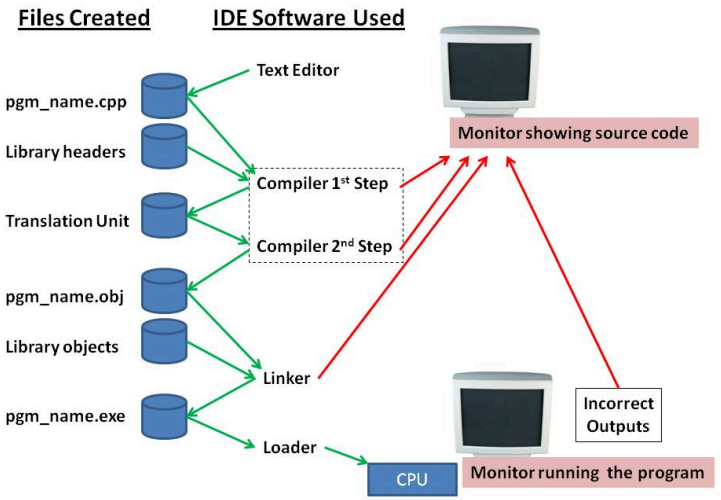
High-level language programs are usually written (coded) as

1. Wikipedia: Integrated development environment

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.

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:

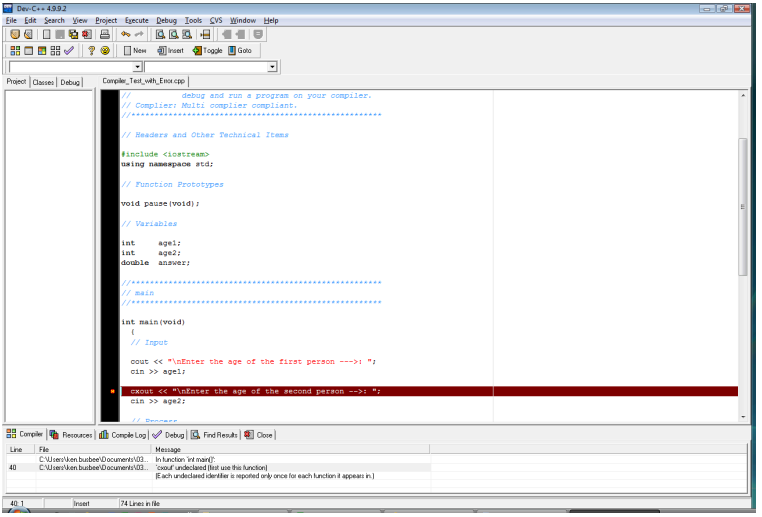
1. Compiler
2. Linker

3. Logic

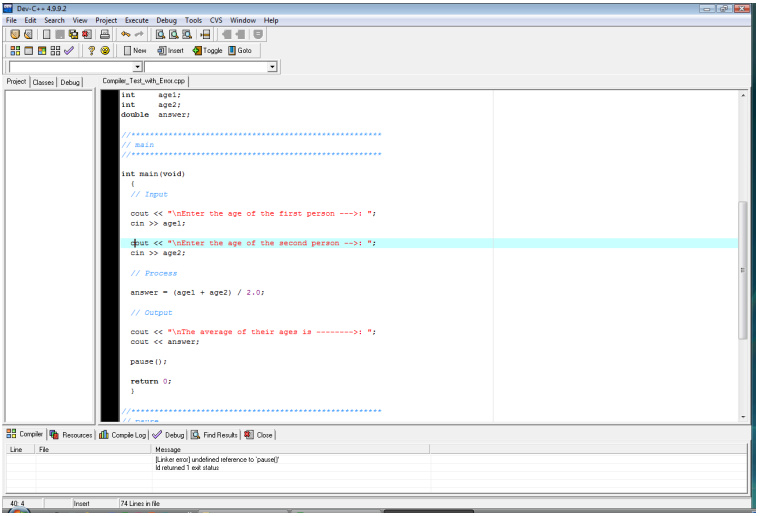
There are two types of compiler errors; pre-processor (1st step) and conversion (2nd step). A review of Figure 1 above shows the four arrows returning to the source code so that the programmer can correct the mistake.

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.

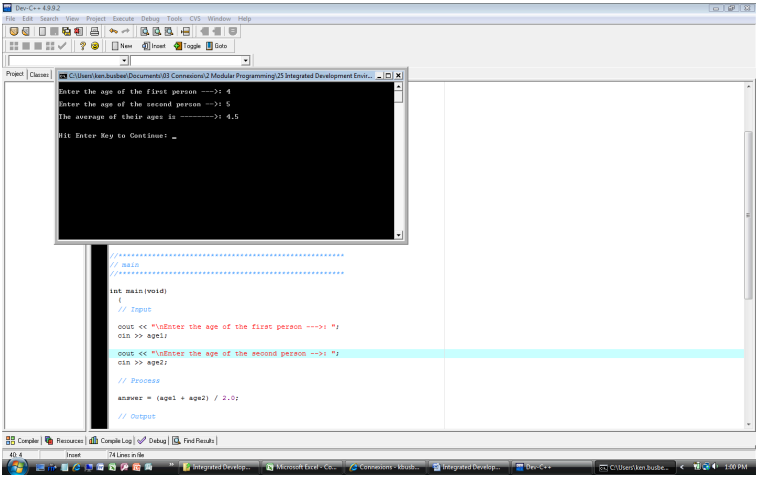
The next three figures show IDE monitor interaction for the **Bloodshed Dev-C++ 5 compiler/IDE**.



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

- archive.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

1. Wikipedia: Version control

'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

2. Programming Historian: An Introduction to Version Control Using GitHub Desktop
3. Programming Historian: An Introduction to Version Control Using GitHub Desktop

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
- 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:

4. Programming Historian: An Introduction to Version Control Using GitHub Desktop
5. G2Crowd: Best Version Control Systems
6. Wikipedia: Git

- `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 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

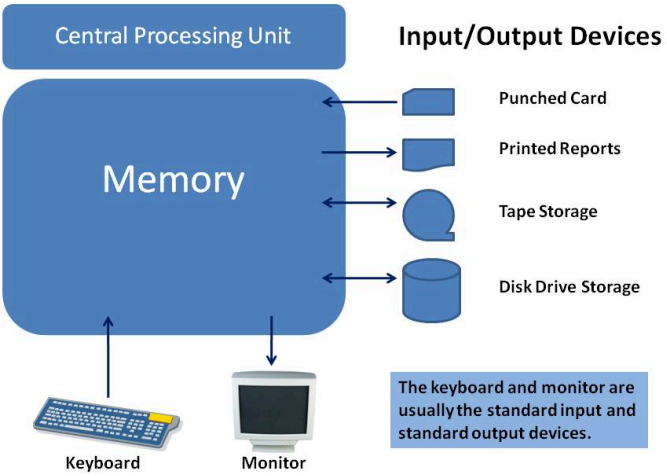
Overview

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

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

Hello World

DAVE BRAUNSCHWEIG

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

1. Wikipedia: "Hello, World!" program
2. Wikipedia: "Hello, World!" program

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]

Program Plan

This program displays “Hello world!”

Input:
None

Process:
None

Output:
Hello world!

Pseudocode

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

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

```
End
```

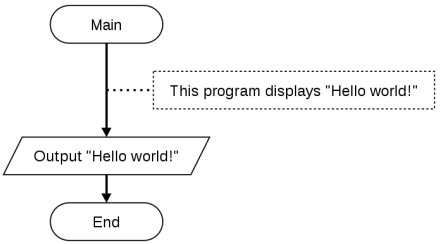
Output

```
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



5. Wikibooks: Programming Fundamentals/Hello World

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.

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:
```

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

```
// http://www.cplusplus.com/doc/tutorial/program_structure/

#include <iostream>

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

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

3. Wikibooks: Programming Fundamentals/Hello World

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

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.

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

Example

Hello World

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

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

Output

```
Hello world!
```

Discussion

Each code element represents:[3]

- `//` begins a comment
- `public class Hello` begins the Hello World program

3. Wikibooks: Programming Fundamentals/Hello World

- `{` 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

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.

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

Example

Hello World

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

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

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

3. Wikibooks: Programming Fundamentals/Hello World

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

References

- Wikiversity: Computer Programming

JavaScript Examples

DAVE BRAUNSCHWEIG

JS 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.

1. Wikipedia: JavaScript
2. TIOBE: Index

Example

Hello World – Console Log

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

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

Output

```
Hello world!
```

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-writ
```

```
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:
```

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

```
# https://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for

print("Hello world!")
```

Output

```
Hello world!
```

Discussion

Each code element represents:[3]

- `#` 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

3. Wikibooks: Programming Fundamentals/Hello World

- Ideone
- paiza.IO
- Python Fiddle
- PythonTutor
- repl.it
- TutorialsPoint

Local IDEs

- IDLE
- Thonny

References

- Wikiversity: Computer Programming

Swift Examples

DAVE BRAUNSCHWEIG

File:Swift logo with text.svg Overview

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/documentatio
```

1. Swift (programming language)
2. TIOBE: Index

```
print("Hello world!")
```

Output

```
Hello world!
```

Discussion

Each code element represents:[3]

- `//` 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

3. Wikibooks: Programming Fundamentals/Hello World

- iSwift
- paiza.IO
- repl.it

Local IDEs

- AppCode
- Atom
- Xcode

References

- Wikiversity: Computer Programming

Practice: Introduction to Programming

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

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

CHAPTER 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

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. Wikipedia: Constant (computer programming)

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.

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

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

Identifier Names

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

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

Data Types

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.

1. Wikipedia: Data type

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.

Pseudocode

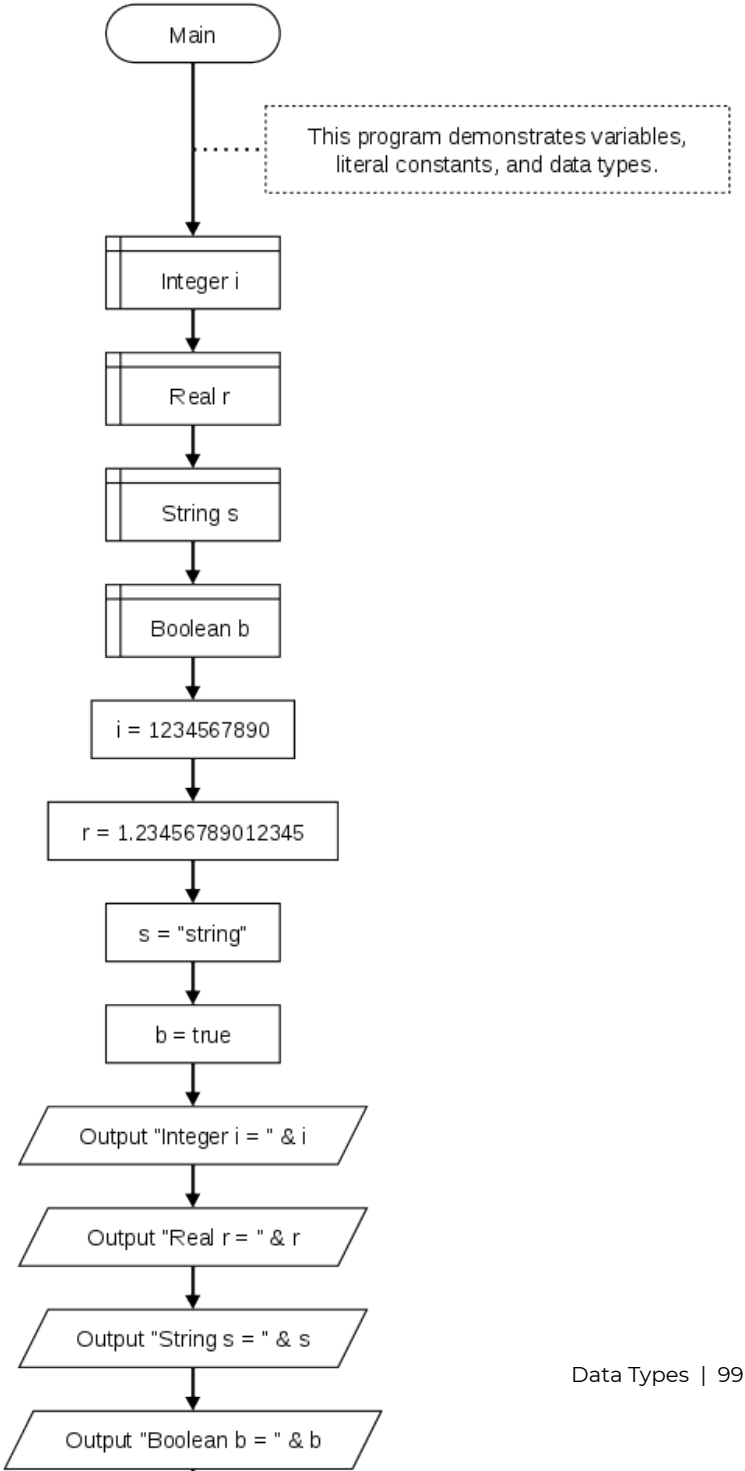
```
Function Main
    ... This program demonstrates variables, literal constants,

    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

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

Integer Data Type

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.

References

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

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

Floating-Point Data Type

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.

1. Wikipedia: Floating-point arithmetic

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`	64 bits / 8 bytes	15 decimal digits	±1.79769313486231570E+308
Java	`float`	32 bits / 4 bytes	7 decimal digits	±3.40282347E+38
Java	`double`	64 bits / 8 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

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

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

String Data Type

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

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

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.

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

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

Boolean Data Type

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,

1. Wikipedia: Boolean data type

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`

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

- archive.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

1. Wikipedia: Nullable type

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.

Order of Operations

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. Wikipedia: Order of operations

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]

- Parentheses
- Exponents

2. Wikipedia: Order of operations

- 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

- archive.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.

1. Wikipedia: Assignment (computer science)

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.

Key Terms

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

References

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

Arithmetic Operators

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

1. Wikipedia: Arithmetic operators

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.

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;`

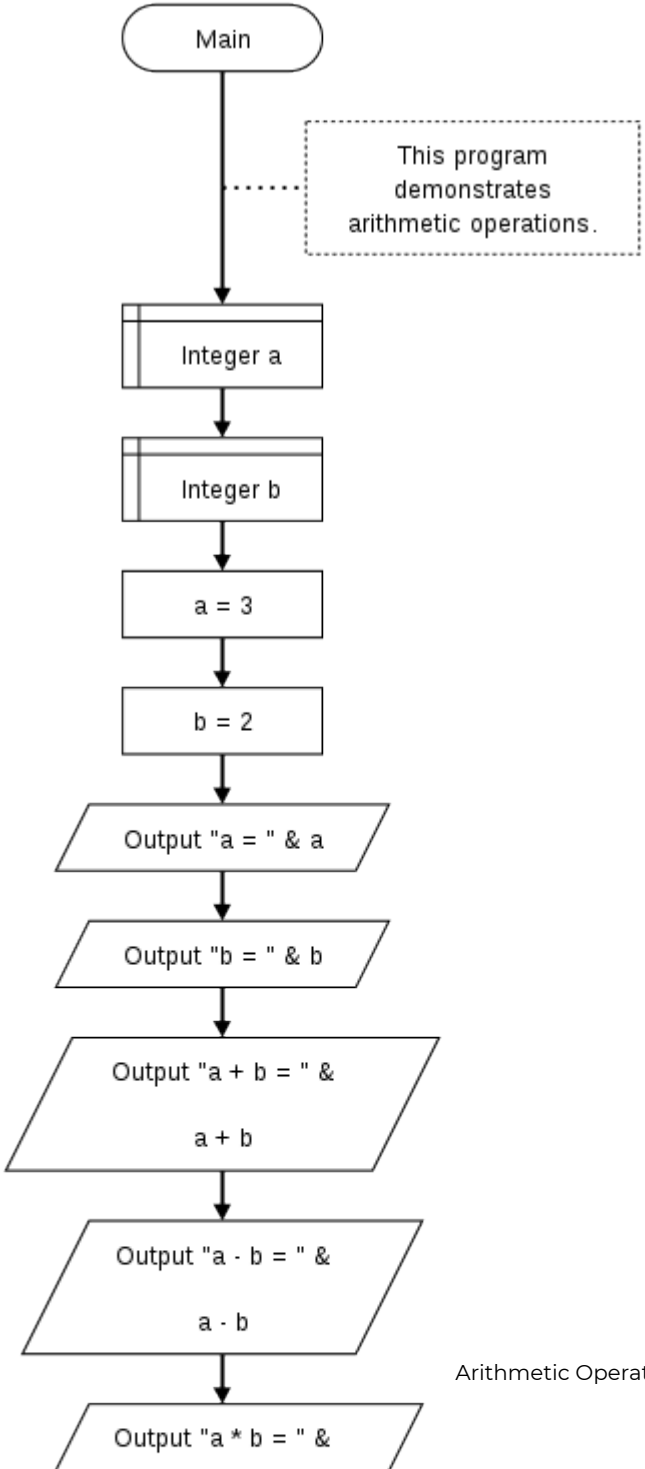
Pseudocode

```
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 = " &

References

- archive.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

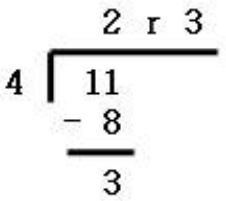
By the time we reach adulthood, we normally think of division as resulting in an answer that might have a fractional part (a floating-point data type). This type of division is known as floating-point division. However, division, when both operands are of the integer data type, may act differently, depending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

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

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 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

- archive.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
```

1. Wikipedia: Unary operation

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 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

- archive.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.

1. Wikipedia: Value (computer science)

```
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;
```

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

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

Data Type Conversions

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

- archive.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)

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

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)

Program Plan

This program converts an input Fahrenheit temperature to Celsius.

Input:
  Display prompt
  Get Fahrenheit temperature

Process:
  Convert Fahrenheit temperature to Celsius

Output:
  Display Fahrenheit and Celsius temperatures

Pseudocode

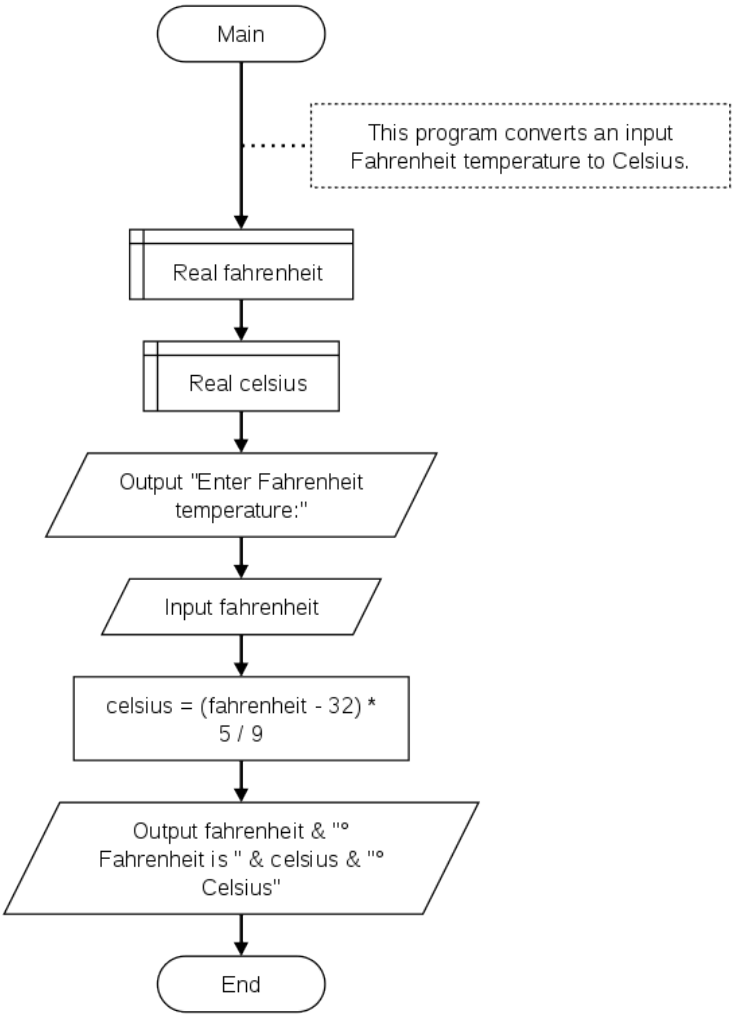
```
Function Main
    ... This program converts an input Fahrenheit temperature t

    Declare Real fahrenheit
    Declare Real celsius
```

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

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

    Output fahrenheit & "° Fahrenheit is " & celsius & "° Celsi
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

#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 Cel
//
// 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 << "° C

    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

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 Cel

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

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 Cel

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) + "° Fah
    }
}
```

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

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 Cel

var fahrenheit;
var celsius;

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

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

output(fahrenheit.toString() + "° Fahrenheit is " + 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 d

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 Cels

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

celsius = (fahrenheit - 32) * 5 / 9

print(str(fahrenheit) + "° Fahrenheit is " + str(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

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/documentatio
```

```
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)
```

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

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 worked per week 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`

1. PythonLearn: Variables, expressions, and statements

`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

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

CHAPTER III

FUNCTIONS

Overview

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

Chapter Outline

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

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

1. Wikipedia: Modular programming

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. 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> <
    //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

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

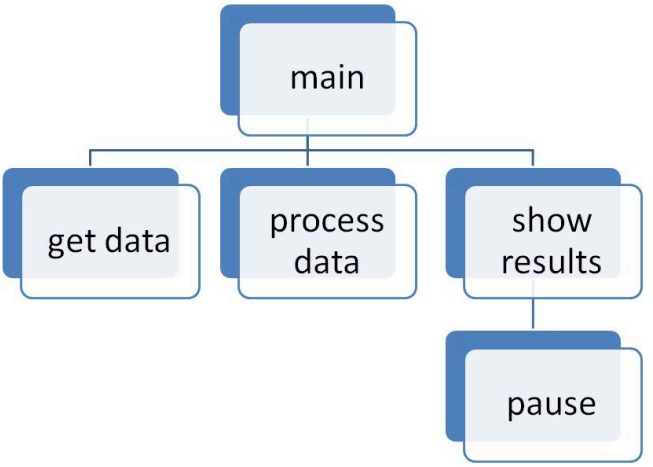
Hierarchy or Structure Chart

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

- archive.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

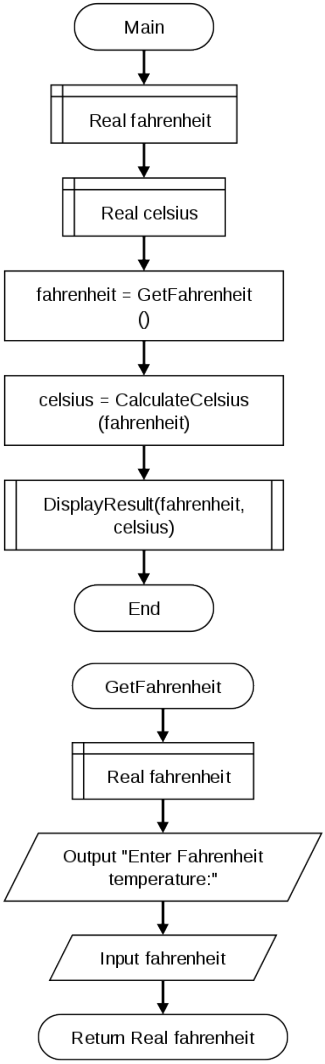
Pseudocode

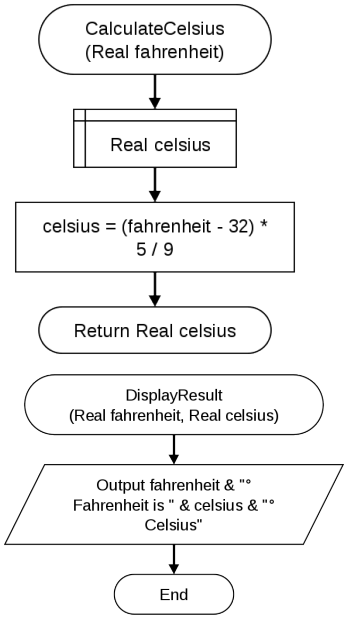
```
Function Main
    ... This program asks the user for a Fahrenheit temperature
    ... converts the given temperature to Celsius,
    ... and displays the results.

    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 & "° Celsi
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)
```

1. Wikipedia: Parameter (computer programming)

```
    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 & "° Celsi
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 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

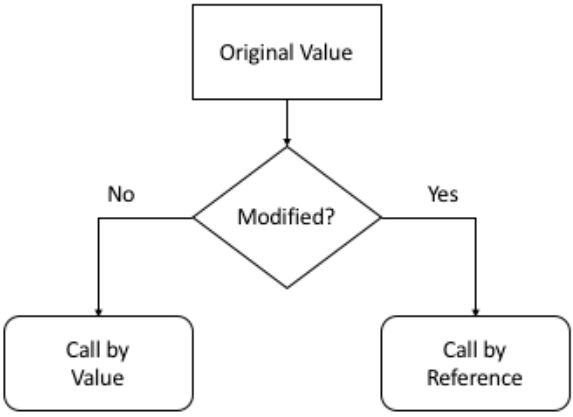
- Wikiversity: Computer Programming

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

1. Wikipedia: Parameter (computer programming)

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
    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:"
```

1. Wikipedia: Return statement

```
    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.

```
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

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

Void Data Type

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

1. Wikipedia: Void type

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

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

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

1. Wikipedia: Scope (computer science)

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 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

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

Programming Style

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

1. Wikipedia: Programming style

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 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]

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

Indentation

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

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

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

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

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

Standard Libraries

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

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

Program Plan

This program converts an input Fahrenheit temperature to Celsius.

Main Program
  Get Fahrenheit
  Calculate Celsius
  Display Result

Get Fahrenheit
  Parameters:
    None
  Process:
    Display Prompt
    Get Fahrenheit temperature
  Return Value:
    Fahrenheit temperature

Calculate Celsius
  Parameters:
    Fahrenheit temperature
  Process:
    Convert Fahrenheit temperature to Celsius
  Return Value:
    Celsius temperature

Display Result
  Parameters:
    Fahrenheit temperature
    Celsius temperature
  Process:
    Display Fahrenheit and Celsius temperatures
  Return Value:
    None

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
    {
        Console.WriteLine(fahrenheit.ToString() + "° Fahrenheit
            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
        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/documentatio

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(cels
```

```
}

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

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 worked per week 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

1. PythonLearn: Variables, expressions, and statements

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

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

CHAPTER IV

CONDITIONS

Overview

This chapter introduces conditions and selection control structures.

Chapter Outline

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

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

1. Wikipedia: Structured programming

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 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.

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

References

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

Selection Control Structures

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
```

1. Wikipedia: Conditional (computer programming)

```
    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`

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
```

2. Wikipedia: Conditional (computer programming)

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

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

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
```

1. Wikipedia: Conditional (computer programming)

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:

```
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

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

Code Blocks

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

1. Wikipedia: Block (programming)

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;
}
```

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.

Key Terms

block
: Another name for a compound statement.

compound statement
: A unit of code consisting of zero or more statements.

References

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

Relational Operators

KENNETH LEROY BUSBEE

Overview

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

Discussion

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

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

1. Wikipedia: Relational operator

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

Examples:

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

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

Key Terms

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

References

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

Assignment vs Equality

KENNETH LEROY BUSBEE

Overview

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

Discussion

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

Within various languages, we have the potential added confusion of the equals symbol = as an operator that does

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

not represent the normal math meaning of equality that we have used for most of our life. The equals symbol typically means: assignment. To get the equality concept of math we often use two equal symbols to represent the relational operator of equality. Let's consider:

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

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

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

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

Don't get caught using assignment for equality.

References

- archive.org: Programming Fundamentals – A Modular

Structured Approach using C++

Logical Operators

Overview

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

Discussion

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

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

1. Wikipedia: Logical connective

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

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

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

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

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

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

The final evaluation of the expression is: true.

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

When forming logical expressions programmers often use

parentheses (even when not technically needed) to make the logic of the expression very clear. Consider the above complex Boolean expression rewritten:

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

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

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

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

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

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

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

This would evaluate to false as:

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

Truth Tables

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

Logical and (&&)

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

Logical or (||)

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

Logical not (!)

x	not x
false	true
true	false

Examples

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

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

have some time to play before dinner, but both of them... well, I hated “and”.

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

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

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

Examples:

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

More examples:

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

- 5 > 1 and 7

Key Terms

logical operator
: An operator used to create complex Boolean expressions.

truth tables
: A common way to show logical relationships.

References

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

Nested If Then Else

KENNETH LEROY BUSBEE

Overview

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

Discussion

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

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

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

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

```
        you can't drink
    else
        you can drink
```

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

Multiway Selection

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

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

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

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

Key Terms

multiway selection
: Using control structures to be able to select from more than two choices.

nested control structures
: Placing one control structure inside of another.

References

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

Case Control Structure

KENNETH LEROY BUSBEE

Overview

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

Discussion

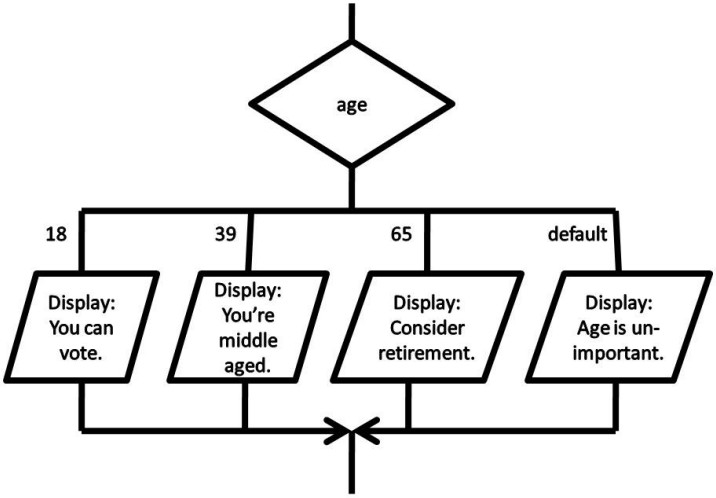
One of the drawbacks of two-way selection is that we can only consider two choices. But what do you do if you have more than two choices? Consider the following which has four choices:

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

You get an appropriate message depending on the value of age. The last item is referred to as the default. If the age is

1. Wikipedia: Switch statement

not equal to 18, 39 or 65 you get the default message. In some situations there is no default action. Consider this flowchart example:



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

Code to Accomplish Multiway Selection

Python does not support a case control structure. But using

the same example as above, here is C++ / C# / Java / JavaScript / Swift code to accomplish the case control structure.

```
switch (age)
{
    case 18:
        message = "You can vote.";
        break;
    case 39:
        message = "You're middle-aged.";
        break;
    case 65:
        message = "Consider retirement.";
        break;
    default:
        message = "Age is unimportant.";
        break;
}
```

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

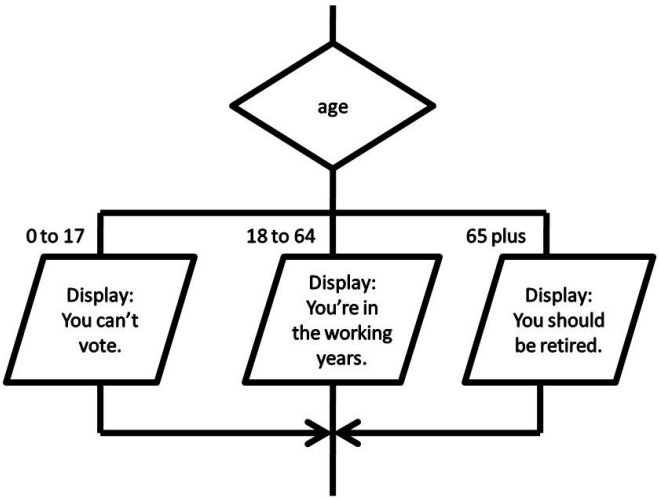
Many programming languages require the listed values for the case control structure be of the integer family of data types. This basically means either an integer or character data type. Consider this example that uses character data type (choice is a character variable):

```
switch (choice)
{
    case 'A':
```

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

Limitations of the Case Control Structure

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



Consider also the following pseudocode for the same logic:

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

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

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

Good Structured Programming Methods

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

Key Terms

case
: A control structure that does multiway selection.

switch
: A control structure that can be made to act like a case control structure.

References

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

Program Plan

This program asks the user to select Fahrenheit or Celsius conversion and input a given temperature. Then the program converts the given temperature and displays the result.

Main Program
  Get Choice
  Either:
    Get Fahrenheit temperature
    Calculate Celsius
    Display Result
  Or:
    Get Celsius temperature
    Calculate Fahrenheit
    Display Result

Get Choice
  Parameters:
    None
  Process:
    Display prompt
    Get choice of Fahrenheit or Celsius conversion
  Return Value:
    Choice

Get Temperature
  Parameters:
    Label (Fahrenheit or Celsius)
  Process:
    Display prompt with label
    Get temperature
  Return Value:
    Temperature

Calculate Celsius
  Parameters:
    Fahrenheit temperature
  Process:
    Convert Fahrenheit temperature to Celsius
  Return Value:
    Celsius temperature

Calculate Fahrenheit
  Parameters:
    Celsius temperature
  Process:
    Convert Celsius temperature to Fahrenheit
  Return Value:
    Fahrenheit temperature

Display Result
  Parameters:
    Input temperature
    From label (Fahrenheit or Celsius)
    Output temperature
    To label (Fahrenheit or Celsius)
  Process:
    Display temperatures and labels
  Return Value:
    None

Condition Examples

DAVE BRAUNSCHWEIG

Temperature

Pseudocode

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

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

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

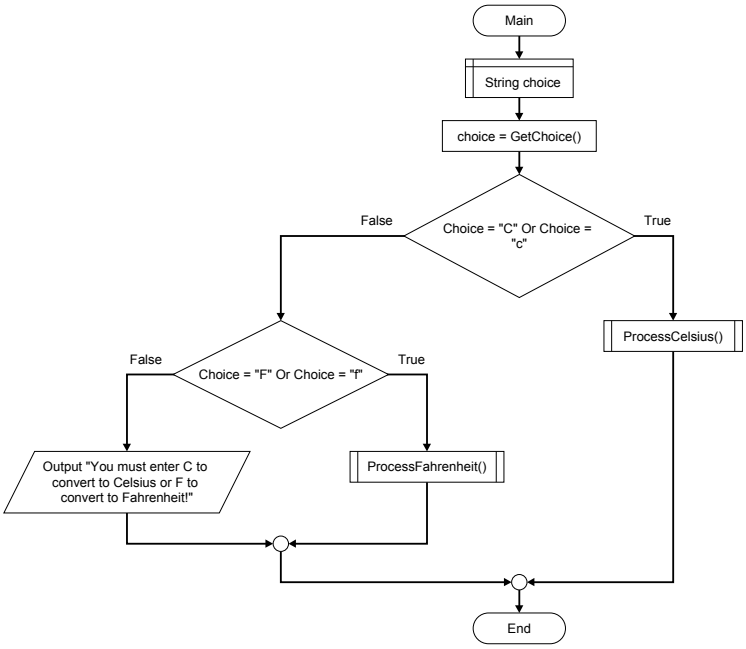
```
    Assign celsius = (fahrenheit - 32) * 5 / 9
Return Real celsius

Function CalculateFahrenheit (Real celsius)
    Declare Real fahrenheit

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

Function DisplayResult (Real temperature, String fromLabel, Rea
    Output temperature & "° " & fromLabel & " is " & result & "
End

Function GetChoice
    Declare String choice

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

Function GetTemperature (String label)
    Declare Real temperature

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

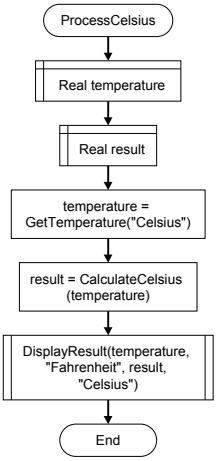
Output

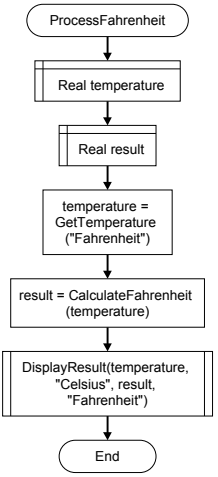
```
Enter C to convert to Celsius or F to convert to Fahrenheit:
 c
Enter Fahrenheit temperature:
 100
100° Fahrenheit is 37.7777777777778° Celsius
```

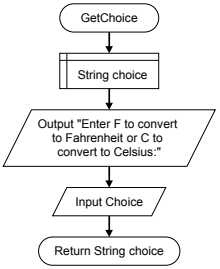
```
Enter C to convert to Celsius or F to convert to Fahrenheit:
 f
Enter Celsius temperature:
 100
100° Celsius is 212° Fahrenheit

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

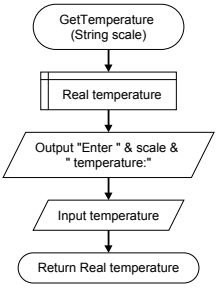
Flowchart

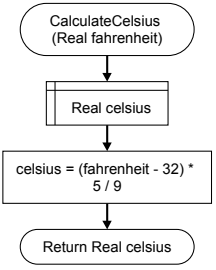


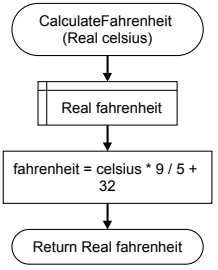

ProcessCelsius
Real temperature
Real result
temperature = GetTemperature("Celsius")
result = CalculateCelsius (temperature)
DisplayResult(temperature, "Fahrenheit", result, "Celsius")
End

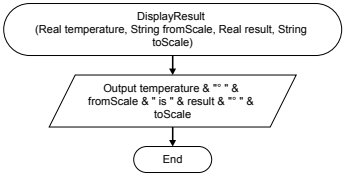

ProcessFahrenheit
Real temperature
Real result
temperature = GetTemperature ("Fahrenheit")
result = CalculateFahrenheit (temperature)
DisplayResult(temperature, "Celsius", result, "Fahrenheit")
End


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


GetTemperature (String scale)
Real temperature
Output "Enter " & scale & " temperature:"
Input temperature
Return Real temperature


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


CalculateFahrenheit (Real celsius)
Real fahrenheit
fahrenheit = celsius * 9 / 5 + 32
Return Real fahrenheit


DisplayResult (Real temperature, String fromScale, Real result, String toScale)
Output temperature & "° " & fromScale & " is " & result & "° " & toScale
End

References

- Wikiversity: Computer Programming

C++ Examples

DAVE BRAUNSCHWEIG

Temperature

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

#include <iostream>

using namespace std;

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

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

    char choice;
    double temperature;
    double result;

    cout << "Enter F to convert to Fahrenheit or C to convert t

    // if-else approach
```

```
    if (choice == 'C' || choice == 'c') {
        temperature = getTemperature("Fahrenheit");
        result = calculateCelsius(temperature);
        displayResult(temperature, "Fahrenheit", result, "Celsi
    }
    else if (choice == 'F' || choice == 'f') {
        temperature = getTemperature("Celsius");
        result = calculateFahrenheit(temperature);
        displayResult(temperature, "Celsius", result, "Fahrenhe
    }
    else {
        cout << "You must enter C to convert to Celsius or F to
    }

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

double getTemperature(string label) {
    double temperature;
```

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

    return temperature;
}

double calculateCelsius(double fahrenheit) {
    double celsius;

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

    return celsius;
}

double calculateFahrenheit(double celsius) {
    double fahrenheit;

    fahrenheit = celsius * 9 / 5 + 32;

    return fahrenheit;
}

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

Output

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

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

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

References

- Wikiversity: Computer Programming

C# Examples

DAVE BRAUNSCHWEIG

Temperature

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

using System;

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

        string choice;
        double temperature;
        double result;

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

        // if-else approach
        if (choice == "C" || choice == "c")
        {
            temperature = GetTemperature("Fahrenheit");
```

```
            result = CalculateCelsius(temperature);
            DisplayResult(temperature, "Fahrenheit", result, "C
        }
        else if (choice == "F" || choice == "f")
        {
            temperature = GetTemperature("Celsius");
            result = CalculateFahrenheit(temperature);
            DisplayResult(temperature, "Celsius", result, "Fahr
        }
        else
        {
            Console.WriteLine("You must enter C to convert to C
        }

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

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

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

        return temperature;
    }

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

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

        return celsius;
    }

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

        fahrenheit = celsius * 9 / 5 + 32;

        return fahrenheit;
    }

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

Output

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

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

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

References

- Wikiversity: Computer Programming

Java Examples

DAVE BRAUNSCHWEIG

Temperature

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

import java.util.*;

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

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

        String choice;
        double temperature;
        double result;

        choice = getChoice();

        // if-else approach
        if (choice.equals("C") || choice.equals("c")) {
            temperature = getTemperature("Fahrenheit");
            result = calculateCelsius(temperature);
            displayResult(temperature, "Fahrenheit", result, "C
```

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

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

public static String getChoice() {
    String choice;

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

    return choice;
}
```

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

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

        return temperature;
    }

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

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

        return celsius;
    }

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

        fahrenheit = celsius * 9 / 5 + 32;

        return fahrenheit;
    }

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

Output

```
Enter C to convert to Celsius or F to convert to Fahrenheit:
```

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

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

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

References

- Wikiversity: Computer Programming

JavaScript Examples

DAVE BRAUNSCHWEIG

Temperature

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

main();

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

    var choice;
    var temperature;
    var result;

    choice = getChoice();

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

```
        temperature = getTemperature("Celsius");
        result = calculateFahrenheit(temperature);
        displayResult(temperature, "Celsius", result, "Fahrenhe
    }
    else {
        output("You must enter C to convert to Celsius or F to
    }

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

function getChoice() {
    var choice;

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

    return choice;
}
```

```
function getTemperature(label) {
    var temperature;

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

    return temperature;
}

function calculateCelsius(fahrenheit) {
    var celsius;

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

    return celsius;
}

function calculateFahrenheit(celsius) {
    var fahrenheit;

    fahrenheit = celsius * 9 / 5 + 32;

    return fahrenheit;
}

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

function input(text) {
  if (typeof window === 'object') {
    return prompt(text)
  }
  else if (typeof console === 'object') {
```

```
    const rls = require('readline-sync');
    var value = rls.question(text);
    return value;
  }
  else {
    output(text);
    var isr = new java.io.InputStreamReader(java.lang.System.in
    var br = new java.io.BufferedReader(isr);
    var line = br.readLine();
    return line.trim();
  }
}

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

Output

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

Enter C to convert to Celsius or F to convert to Fahrenheit:
```

```
 f
Enter Celsius temperature:
 100
100° Celsius is 212° Fahrenheit

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

References

- Wikiversity: Computer Programming

Python Examples

DAVE BRAUNSCHWEIG

Temperature

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


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


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


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


def calculate_fahrenheit(celsius):
```

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


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


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


main()
```

Output

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

Enter C to convert to Celsius or F to convert to Fahrenheit:
 f
```

```
Enter Celsius temperature:
 100
100.0° Celsius is 212.0° Fahrenheit

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

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/documentatio

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

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

    return choice
}

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

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

    return temperature
}

func calculateCelsius(fahrenheit: Double) -> Double {
```

```
    var celsius: Double

    celsius = (fahrenheit - 32) * 5 / 9

    return celsius
}

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

    fahrenheit = celsius * 9 / 5 + 32

    return fahrenheit
}

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

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

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

    choice = getChoice()

    // if-else approach
    if choice == "C" || choice == "c" {
        temperature = getTemperature(label:"Fahrenheit")
        result = calculateCelsius(fahrenheit:temperature)
        displayResult(temperature:temperature, fromLabel:"Fahre
    else if choice == "F" || choice == "f" {
        temperature = getTemperature(label:"Celsius")
```

```
        result = calculateFahrenheit(celsius:temperature)
        displayResult(temperature:temperature, fromLabel:"Celsi
    }
    else {
        print("You must enter C to convert to Celsius or F to c
    }

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

main()
```

Output

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

Enter C to convert to Celsius or F to convert to Fahrenheit:
 f
```

```
Enter Celsius temperature:
 100
100.0° Celsius is 212.0° Fahrenheit

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

References

- Wikiversity: Computer Programming

Practice: Conditions

KENNETH LEROY BUSBEE

Review Questions

True / False

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

Answers:

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

Expressions

Evaluate the following Boolean expressions:

1. 25 < 7
2. 3 < 7

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

Answers:

1. 0
2. 1
3. 1
4. 0
5. 1
6. 0
7. 1
8. Error, the “not greater than” is not a valid operator.
9. 0
10. 1
11. 0
12. 1
13. 0
14. 1
15. 0
16. Error, there needs to be an operand between the operators < and &&.

Short Answer

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

Activities

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

1. Create a program to prompt the user for hours and rate per hour and then compute gross pay (hours * rate). Include a calculation to give 1.5 times the hourly rate for any overtime (hours worked above 40 hours).[1] For example, 50 hours worked at $10 per hour with overtime is $550.
2. Create a program that asks the user how old they are in years. Then ask the user if they would like to know how old they are in (M)onths, (D)ays, (H)ours, or (S)econds. Use if/else conditional statements to calculate and display their approximate age in the selected timeframe. Do not perform any unnecessary calculations.
3. Review MathsIsFun: US Standard Lengths. Create a

1. PythonLearn: Variables, expressions, and statements

program that asks the user for a distance in miles, and then ask the user if they want the distance in US measurements (yards, feet, and inches) or in metric measurements (kilometers, meters, and centimeters). Use if/else conditional statements to determine their selection and then calculate and display the results.

4. Review MathsIsFun: Area of Plane Shapes. Create a program that asks the user what shape they would like to calculate the area for. Use if/else conditional statements to determine their selection and then gather the appropriate input and calculate and display the area of the shape.
5. Review Wikipedia: Aging in dogs. Create a program to prompt the user for the name of their dog and its age in human years. Calculate and display the age of their dog in dog years, assuming the first two years equal 10.5 years each, with subsequent years equaling four human years. Be sure to include the dog's name in the output, such as: `Spike is 25.0 years old in dog years.`
6. Create a program that helps the user determine what sock size to order based on their shoe size:

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

   Use if/else conditional statements to determine their selection and then display the results. Round half-sizes up to the next whole size. One option for rounding is to add 0.5 and then convert to an integer.
7. If your programming language supports it, update one or more of the programs above to replace the if/else conditional statements with case/select conditional statements.
8. Review Wikipedia: Is functions. If your programming language supports it, update one or more of the programs

above to include input validation for all numeric input.

9. If your programming language supports it, extend one or more of the programs above by adding structured exception handling statements (try-catch, try-except, etc.) to handle any runtime errors caused by the user entering invalid values for the input.

References

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

CHAPTER V

LOOPS

Overview

This chapter introduces loops and iteration control structures.

Chapter Outline

Learning Objectives

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

Iteration Control Structures

Overview

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

Discussion

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

Iteration (Repetition) Control Structures

pseudocode: While

1. Wikipedia: Structured programming

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

pseudocode: Do While

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

pseudocode: Repeat Until

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

pseudocode: For

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

References

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

While Loop

KENNETH LEROY BUSBEE

Overview

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

Discussion

Introduction to Test Before Loops

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

Understanding Iteration in General – while

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

1. Wikipedia: While loop

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

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

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

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

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

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

sometimes called a pre-test loop, meaning the test is pre (or Latin for before) the action and update.

Human Example of the while Loop

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

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

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

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

Infinite Loops

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

Pseudocode infinite loop

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

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

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

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

Counting Loops

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

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

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

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

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

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

Infinite Loops

Consider:

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

Missing the flag update usually causes an infinite loop.

Variations on Counting

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

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

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

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

This loop is a counting loop similar to our first counting loop example. The only difference is instead of using a literal constant (in other words 5) in our expression, we used the variable age (and thus the value stored in age) to determine how many times to execute the loop. However, unlike our first

counting loop example which will always execute exactly 5 times; it is possible that the person's age is 0 (zero) and the first time we test the expression it will be false and the action part of the loop would never be executed.

Key Terms

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

event controlled
: Using user input to control a loop.

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

initialize item
: An attribute of iteration control structures.

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

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

while
: A test before iteration control structure.

2. Wikipedia: Infinite loop

References

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

Do While Loop

Overview

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

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

Discussion

Introduction to Test After Loops

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

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

Understanding Iteration in General – do while

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

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

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

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

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

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

The English phrasing is, "You do the action while the expression is true". This is looping on the true. When the test expression

is false, you stop the loop and go on with the next item in the program. Notice, because this is a test after loop the action will always happen **at least once**. It is called a test after loop because the test comes after the action. It is also sometimes called a post-test loop, meaning the test is post (or Latin for after) the action and update.

Understanding Iteration in General – repeat until

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

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

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

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

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

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

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

An Example

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

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

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

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

Infinite Loops

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

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

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

Consider the following code:

```
do
    Output "What is your age? "
```

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

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

Key Terms

action item
: An attribute of iteration control structures.

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

do while
: A test after iteration control structure.

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

3. Wikipedia: Infinite loop

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

test item
: An attribute of iteration control structures.

update item
: An attribute of iteration control structures.

References

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

Flag Concept

KENNETH LEROY BUSBEE

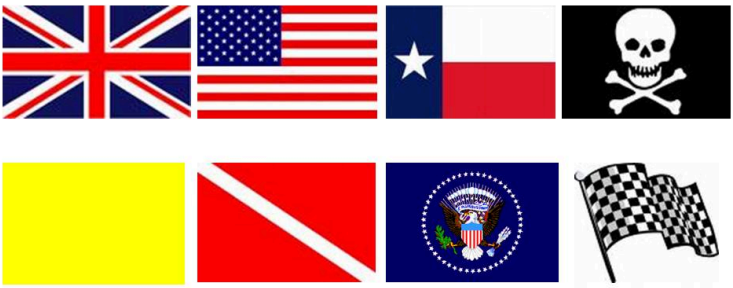
Overview

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

Discussion

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

1. Wikipedia: Bit field



Various Flags

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

Computer Implementation

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

flag controlling an if then control structure

```
if gender equals 'F'
```

```
display "Are you pregnant?"
get answer from user store in pregnant variable
```

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

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

Two Flags with the Same Meaning

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

simple flag comparison

```
loop_response equals 'y'
```

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

complex flag comparison

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

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

Multiple Flags in One Byte

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

Key Terms

flag

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

References

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

For Loop

KENNETH LEROY BUSBEE

Overview

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

Discussion

Introduction to Test Before Loops

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

1. Wikipedia: For loop

Understanding Iteration in General – for

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

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

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

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

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

- Initializing the flag – done once

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

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

An Example

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

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

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

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

Infinite Loops

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

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

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

Key Terms

for

A test before iteration control structure typically used for counting.

References

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

Branching Statements

KENNETH LEROY BUSBEE

Overview

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

Discussion

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

1. Wikipedia: Branch (computer science)

Examples

break

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

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

continue

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

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

return

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

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

goto

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

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

exit

Although exit is technically a pre-defined function, it is covered here because of its common usage in programming. A good example is the opening a file and then testing to see if the file was actually opened. If not, we have an error that usually indicates that we want to prematurely stop the execution of the program. The exit function terminates the running of the program and in the process returns an integer value back to the operating system. It fits the definition of branching which is to jump to some other place in the program.

Key Terms

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

break
: A branching statement that terminates the existing structure.

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

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

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

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

References

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

Increment and Decrement Operators

KENNETH LEROY BUSBEE

Overview

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

Discussion

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

```
increment the counter
```

The concept is:

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

That is you fetch the existing value of the counter and add one then store the answer back into the variable counter. Many programming languages allow their increment and decrement operators to only be used with the integer data

1. Wikipedia: Increment and decrement operators

type. Programmers will sometimes use inc and dec as abbreviations for increment and decrement respectively.

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

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

Code Examples

Basic Concept

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

```
counter = counter + 1;

counter += 1;

counter++;

++counter;
```

As statements, the four examples all do the same thing. They add 1 to the value of whatever is stored in counter. The decrement operator is represented by two minus signs in a row. They would subtract 1 from the value of whatever was in the variable being decremented. The precedence of increment and decrement depends on if the operator is attached to the right

of the operand (postfix) or to the left of the operand (prefix). Note that postfix and prefix do not have the same precedence.

Postfix Increment

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

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

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

Prefix Increment

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

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

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

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

am important so increment me first and use my new value for the rest of the evaluations).

Allowable Data Types

Within some programming languages, increment and decrement can be used only on the integer data type. Other languages expand this not only to all of the integer family but also to the floating-point family (float and double). Incrementing 3.87 will change the value to 4.87. Decrementing 'C' will change the value to 'B'. Remember the ASCII character values are really one-byte unsigned integers (domain from 0 to 255).

Exercises

Evaluate the following items using increment or decrement:

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

Key Terms

decrement
: Subtracting one from the value of a variable.

increment
: Adding one to the value of a variable.

postfix
: Placing the increment or decrement operator to the right of the operand.

prefix
: Placing the increment or decrement operator to the left of the operand.

References

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

Integer Overflow

KENNETH LEROY BUSBEE

Overview

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

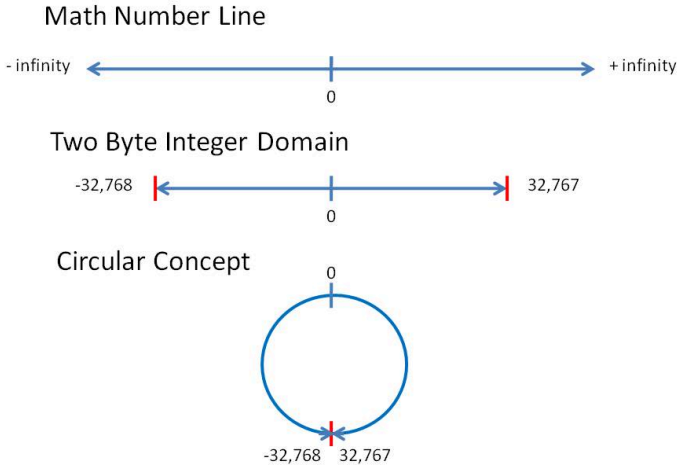
The most common result of an overflow is that the least significant representable bits of the result are stored; the result is said to wrap around the maximum (i.e. modulo power of two). An overflow condition may give results leading to unintended behavior. In particular, if the possibility has not been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

Discussion

There are times when character and integer data types are lumped together because they both act the same (often called the integer family). Maybe we should say they act differently than the floating-point data types. The integer family values jump from one value to another. There is nothing between 6 and 7 nor between 'A' and 'B'. It could be asked why not

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

make all your numbers floating-point data types. The reason is twofold. First, some things in the real world are not fractional. A dog, even with only 3 legs, is still one dog not three-fourths of a dog. Second, the integer data type is often used to control program flow by counting (counting loops). The integer family has a circular wrap-around feature. Using a two-byte integer, the next number bigger than 32767 is negative 32768 (character acts the same way going from 255 to 0. We could also reverse that to be the next smaller number than negative 32768 is positive 32767. This can be shown by using a normal math line, limiting the domain and then connecting the two ends to form a circle.

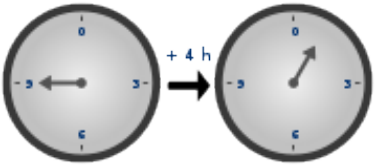


This circular nature of the integer family works for both integer and character data types. In theory, it should work for the Boolean data type as well; but in most programming languages it does not for various technical reasons.

“In mathematics, modular arithmetic (sometimes called clock arithmetic) is a system of arithmetic for integers where

numbers “wrap around” after they reach a certain value — the modulus. ...

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



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

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

Implications When Executing Loops

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

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

properly handled. Usually, the direction of counting (increment or decrement) need to be switched.

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

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

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

Key Terms

circular nature
: Connecting the negative and positive ends of the domain of an integer family data type.

loop control
: Making sure the attributes of a loop are properly handled.

modular arithmetic
: A system of arithmetic for integers where numbers "wrap around".

References

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

Nested For Loops

KENNETH LEROY BUSBEE

Overview

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

Discussion

Nested Control Structures

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

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

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

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

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

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

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

An Example – Nested for loops

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

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

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

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

Key Terms

complex logic
: Often solved with nested control structures.

References

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

Program Plan

This program demonstrates While, Do, and For loop counting using user-designated start, stop, and increment values.

Main Program
  Get Starting value
  Get Ending value
  Get Increment value
  Demonstrate While Loop
  Demonstrate Do Loop
  Demonstrate For Loop

Get Value
  Parameters:
    Label
  Process:
    Display prompt with label
    Get value
  Return Value:
    Value

Demonstrate While Loop
  Parameters:
    Start
    Stop
    Increment
  Process:
    Initialize count
    Loop while count < stop
      Display count
      Increment count
  Return Value:
    None

```
Demonstrate Do Loop
  Parameters:
    Start
    Stop
    Increment
  Process:
    Initialize count
    Loop
      Display count
      Increment count
      While count <= stop
  Return Value:
    None

Demonstrate For Loop
  Parameters:
    Start
    Stop
    Increment
  Process:
    Initialize count
    Loop for count from start to stop by increment
      Display count
  Return Value:
    None
```

Loop Examples

DAVE BRAUNSCHWEIG

Counting

Pseudocode

```
... This program demonstrates While, Do, and For loop counting

Function Main
    Declare Integer start
    Declare Integer stop
    Declare Integer increment

    Assign start = GetValue("starting")
    Assign stop = GetValue("ending")
    Assign increment = GetValue("increment")
    Call DemonstrateWhileLoop(start, stop, increment)
    Call DemonstrateDoLoop(start, stop, increment)
    Call DemonstrateForLoop(start, stop, increment)
End

Function GetValue (String name)
    Declare Integer value

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

Function DemonstrateWhileLoop (Integer start, Integer stop, Int
    Output "While loop counting from " & start & " to " & stop
```

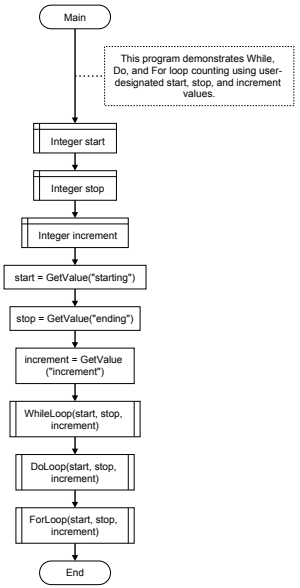
```
    Declare Integer count

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

Function DemonstrateDoLoop (Integer start, Integer stop, Intege
    Output "Do loop counting from " & start & " to " & stop & "
    Declare Integer count

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

Function DemonstrateForLoop (Integer start, Integer stop, Integ
    Output "For loop counting from " & start & " to " & stop &
    Declare Integer count

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

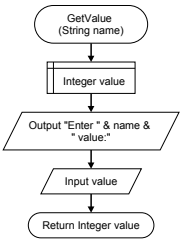
Output

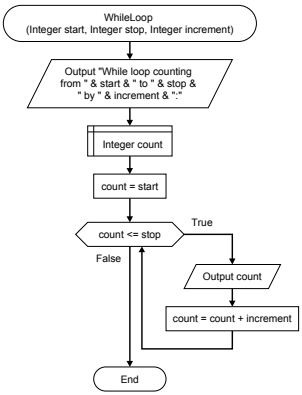
```
Enter starting value:
1
Enter ending value:
```

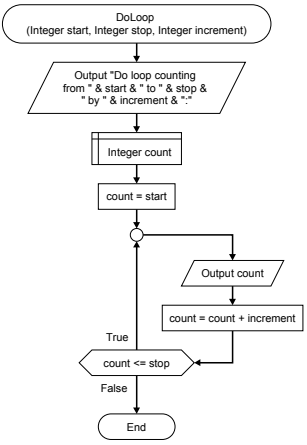
```
3
Enter increment value:
1
While loop counting from 1 to 3 by 1:
1
2
3
Do loop counting from 1 to 3 by 1:
1
2
3
For loop counting from 1 to 3 by 1:
1
2
3
```

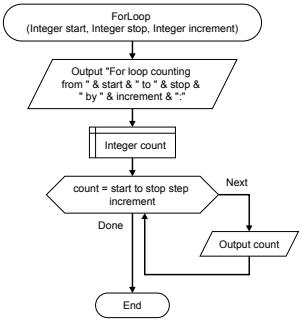
Flowchart











References

- [Wikiversity: Computer Programming]

C++ Examples

DAVE BRAUNSCHWEIG

Counting

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

#include

using namespace std;

int getValue(string name);
void demonstrateWhileLoop(int start, int stop, int increment);
void demonstrateDoLoop(int start, int stop, int increment);
void demonstrateForLoop(int start, int stop, int increment);

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

    demonstrateWhileLoop(start, stop, increment);
    demonstrateDoLoop(start, stop, increment);
    demonstrateForLoop(start, stop, increment);

    return 0;
}
```

```
int getValue(string name) {
    int value;

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

    return value;
}

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

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

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

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

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

    for (int count = start; count <= stop; count += increment)
```

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

Output

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

References

- Wikiversity: Computer Programming

C# Examples

DAVE BRAUNSCHWEIG

Counting

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

using System;

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

        DemonstrateWhileLoop(start, stop, increment);
        DemonstrateDoLoop(start, stop, increment);
        DemonstrateForLoop(start, stop, increment);
    }

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

```
    return value;
}

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

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

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

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

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

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

Output

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

References

- Wikiversity: Computer Programming

Java Examples

DAVE BRAUNSCHWEIG

Counting

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

import java.util.*;

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

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

        demonstrateWhileLoop(start, stop, increment);
        demonstrateDoLoop(start, stop, increment);
        demonstrateForLoop(start, stop, increment);
    }

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

        return value;
    }
```

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

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

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

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

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

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

Output

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

References

- Wikiversity: Computer Programming

JavaScript Examples

DAVE BRAUNSCHWEIG

Counting

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

main()

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

    demonstrateWhileLoop(start, stop, increment);
    demonstrateDoLoop(start, stop, increment);
    demonstrateForLoop(start, stop, increment);
}

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

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

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

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

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

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

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

function input(text) {
  if (typeof window === 'object') {
    return prompt(text)
  }
  else if (typeof console === 'object') {
    const rls = require('readline-sync');
    var value = rls.question(text);
```

```
    return value;
  }
  else {
    output(text);
    var isr = new java.io.InputStreamReader(java.lang.System.in
    var br = new java.io.BufferedReader(isr);
    var line = br.readLine();
    return line.trim();
  }
}

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

Output

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

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

References

- Wikiversity: Computer Programming

Python Examples

DAVE BRAUNSCHWEIG

Counting

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


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


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


def demonstrate_do_loop(start, stop, increment):
    print("Do loop counting from " + str(start) + " to " +
        str(stop) + " by " + str(increment) + ":")
    count = start
    while True:
```

```
        print(count)
        count = count + increment
        if not(count <= stop):
            break

def demonstrate_for_loop(start, stop, increment):
    print("For loop counting from " + str(start) + " to " +
        str(stop) + " by " + str(increment) + ":")
    for count in range(start, stop + increment, increment):
        print(count)


def main():
    start = get_value("starting")
    stop = get_value("ending")
    increment = get_value("increment")
    demonstrate_while_loop(start, stop, increment)
    demonstrate_do_loop(start, stop, increment)
    demonstrate_for_loop(start, stop, increment)


main()
```

Output

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

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

References

- Wikiversity: Computer Programming

Swift Examples

DAVE BRAUNSCHWEIG

Counting

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

import Foundation

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

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

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

    var count : Int

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

```
}

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

    var count : Int

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

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

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

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

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

    demonstrateWhileLoop(start: start, stop: stop, increment: i
    demonstrateDoLoop(start: start, stop: stop, increment: incr
```

```
    demonstrateForLoop(start: start, stop: stop, increment: inc
}

main()
```

Output

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

References

- Wikiversity: Computer Programming

Practice: Loops

KENNETH LEROY BUSBEE

Review Questions

True / False

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

Answers:

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

8. true
9. false
10. true

Activities

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

While Loops

Complete the following using a while loop structure.

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

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

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

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

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

Do While / Repeat Until Loops

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

1. Review MathsIsFun: Definition of Average. Create a program that asks the user to enter grade scores. Use a loop to request each score and add it to a total. Continue accepting scores until the user enters either a negative value or no value (your choice). Finally, calculate and display the average for the entered scores.
2. Review Khan Academy: A guessing game. Write a program that allows the user to think of a number between 0 and 100, inclusive. Then have the program try to guess the user's number. Start at the midpoint (50) and ask the user if their number is (h)igher, (l)ower, or (e)qual to the guess. If they indicate lower, guess the new midpoint (25). If they indicate higher, guess the new midpoint (75). Continue efficiently guessing higher or lower until they indicate equal, then print the number of guesses required to guess their number and end the

program.

3. Add a do while / repeat until loop to any activity from a previous chapter. Continue running the program while the user wants to continue or until the user wants to stop.
4. Add an input validation loop to any activity from a previous chapter. Verify that the input is valid before returning the value. Ask the user to input the value again while the input is invalid.

For Loops

Complete the following using a for loop structure.

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

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

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

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

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

3. Review MathsIsFun: Pi. Write a program that uses the Nilakantha series to calculate Pi based on a given number of iterations entered by the user.
4. Review MathsIsFun: Fibonacci Sequence. Write a program that displays the Fibonacci sequence based on a given number of iterations entered by the user.

Nested Loops

Complete the following using a nested loop structure.

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

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

   The output from 3 to 5 might look like:

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

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

References

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

CHAPTER VI

ARRAYS

Overview

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

Chapter Outline

Learning Objectives

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

Arrays and Lists

Overview

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

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

Discussion

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

Arrays can have multiple axes (more than one axis). Each axis is a **dimension**. Thus a single-dimension array is also known as a **list**. A two-dimension array is commonly known as a **table** (a spreadsheet like Excel is a two dimension array). In real life, there are occasions to have data organized into multiple-

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

dimension arrays. Consider a theater ticket with section, row, and seat (three dimensions). Most single-dimension arrays are visualized vertically.

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

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

Defining an Array

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

This is the **defining of storage space**. The square brackets `[]` are used here to create the array with five integer members

and the identifier name of ages. The assignment with braces (that is a block) establishes the initial values assigned to the members of the array. Note the use of the sequence or comma operator. We could have also done something similar to:

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

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

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

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

Key Terms

dimension
: An axis of an array.

list
: A single dimension array.

table
: A two-dimension array.

References

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

Index Notation

Overview

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

Discussion

Example:

1. Wikipedia: Index notation

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

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

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

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

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

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

Key Terms

array member
: An element or value in an array.

index
: An operator that allows us to reference a member of an array.

offset
: The method of referencing array members by starting at zero.

References

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

Displaying Array Members

Overview

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

Discussion

Accessing Array Members

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

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

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

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

While this works for short arrays, it is not very efficient, and quickly becomes overwhelming for longer arrays. One of the

principles of software development is **don't repeat yourself (DRY)**. Violations of the DRY principle are typically referred to as WET solutions, which is commonly taken to stand for either "write everything twice", "we enjoy typing" or "waste everyone's time".[1]

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

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

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

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

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

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

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

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

1. Wikipedia: Don't repeat yourself

```
    Output ages[index]
End
```

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

Key Terms

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

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

References

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

2. Wikipedia: Don't repeat yourself

Arrays and Functions

Overview

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

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

Discussion

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

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

Pseudocode

```
Function Main
```

```
    Declare Integer Array ages[5]

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

Function DisplayArray (Integer Array array)
    Declare Integer index

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

Output

```
49
48
26
19
16
```

Key Terms

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

References

- archive.org: Programming Fundamentals – A Modular

Structured Approach using C++

- Wikiversity: Computer Programming

Math Statistics with Arrays

Overview

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

Discussion

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

This module covers the totaling of the members of an integer array member. The Latin name for totaling is summa, sometimes shortened to the word **sum**. In the example below, the sum function totals the array passed to it. Other mathematical functions often associated with statistics such

1. Wikipedia: Statistics

as: average, count, minimum, maximum, standard deviation, etc. are often developed for processing arrays.

Pseudocode

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

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

    Assign total = sum(ages)

    Output "Total age is: " & total
End

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

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

Output

```
Total age is: 158
```

Key Terms

sum
: Latin for summa or a total.

References

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

Searching Arrays

Overview

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

Discussion

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

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

Two specific linear searches can be made for the maximum (largest) value in the array or the minimum (smallest) value in

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

the array. Maximum and minimum are also known as max and min. Note that the following max and min functions assume an array size >= 1.

Pseudocode

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

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

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

    Output "Maximum age is: " & maximum
    Output "Minimum age is: " & minimum
End

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

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

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

```
    Declare Integer index

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

Output

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

Key Terms

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

maximum
: Aka max or the largest member of an array.

minimum
: Aka min or the smallest member of an array.

References

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

Sorting Arrays

Overview

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

Discussion

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

1. Wikipedia: Sorting algorithm

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

Key Terms

sorting
: Arranging data according to their values.

References

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

Parallel Arrays

DAVE BRAUNSCHWEIG

Overview

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

Discussion

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

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

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

Pseudocode

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

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

    DisplayArrays(names, ages)
End

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

    For index = 0 to Size(array) - 1
        Output names[index] & " is " & ages[index] & " years ol
    End
End
```

Output

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

Key Terms

parallel array
An implicit data structure that uses multiple arrays to

represent a singular array of records.

References

Multidimensional Arrays

KENNETH LEROY BUSBEE

Overview

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

Discussion

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

1. Wikipedia: Array data type

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

Pseudocode

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

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

    DisplayGame(game)
End

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

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

Output

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

Key Terms

array member
: An element or value in an array.

dimension
: An axis of an array.

index
: An operator that allows us to reference a member of an array.

list
: A single dimension array.

offset
: The method of referencing array members by starting at zero.

table
: A two-dimension array.

References

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

Fixed and Dynamic Arrays

DAVE BRAUNSCHWEIG

Overview

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

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

Discussion

Static arrays have their size or length determined when the array is created and/or allocated. For this reason, they may also be referred to as fixed-length arrays or fixed arrays. Array values may be specified when the array is defined, or the array size may be defined without specifying array contents. Depending on the programming language, an uninitialized array may

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

contain default values, or it may contain whatever values were left in memory from previous allocation.

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

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

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

Key Terms

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

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

References

Program Plan

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

Main Program
  Create name and age arrays
  Display arrays
  Calculate sum of ages
  Calculate maximum age
  Calculate minimum age
  Display sum, maximum, and minimum
  Sort and display ages
  Display parallel arrays
  Demonstrate fixed array
  Demonstrate dynamic array
  Demonstrate multidimensional array

Calculate Sum
  Parameters:
    Array
  Process:
    Initialize total
    Loop for index from 0 to array length by 1
      Add array index value to total
  Return Value:
    Sum

Calculate Maximum
  Parameters:
    Array
  Process:
    Initialize maximum to first array value

```
    Loop for index from 1 to array length by 1
      If maximum < array index value
        maximum = array index value
  Return Value:
    Maximum

Calculate Minimum
  Parameters:
    Array
  Process:
    Initialize minimum to first array value
    Loop for index from 1 to array length by 1
      If minimum > array index value
        minimum = array index value
  Return Value:
    Minimum

Demonstrate Parallel Arrays
  Parameters:
    Name Array
    Age Array
  Process:
    Loop for index from 0 to array length by 1
      Display array index name and age
  Return Value:
    None

Demonstrate Fixed Array
  Parameters:
    None
  Process:
    Initialize array with 5 null values
    For index from 0 to array length by 1
      Set array index value to a random number
    Display array
```

```
  Return Value:
    None

Demonstrate Dynamic Array
  Parameters:
    None
  Process:
    Initialize empty array
    For index from 0 to 5 by 1
      Append a random number to the array
    Display array
  Return Value:
    None

Demonstrate Multidimensional Array
  Parameters:
    None
  Process:
    Initialize multidimensional array as a tic-tac-toe game
    For row from 0 to 2 by 1
      For column from 0 to 2 by 1
        Display array element
        If column < 2
          Display separator
      Go to next output line
  Return Value:
    None
```

C++ Examples

DAVE BRAUNSCHWEIG

Arrays

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

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

using namespace std;

void displayArray(int [], int);
int calculateSum(int [], int);
int calculateMaximum(int [], int);
int calculateMinimum(int [], int);
void demonstrateParallelArrays(string [], int [], int);
void demonstrateFixedArray();
void demonstrateDynamicArray();
void demonstrateMultidimensionalArray();

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

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

    int total = calculateSum(ages, sizeof(ages) / sizeof(int));
    int maximum = calculateMaximum(ages, sizeof(ages) / sizeof(
```

```
    int minimum = calculateMinimum(ages, sizeof(ages) / sizeof(

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

    demonstrateParallelArrays(names, ages, sizeof(ages) / sizeo

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

    demonstrateFixedArray();
    demonstrateDynamicArray();
    demonstrateMultidimensionalArray();

    return 0;
}

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

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

int calculateMaximum(int arry[], int size) {
    int maximum = arry[0];
    for (int index = 1; index < size; index++) {
```

```
        if (maximum < arry[index]) {
            maximum = arry[index];
        }
    }
    return maximum;
}

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

void demonstrateParallelArrays(string names[], int ages[], int
    for (int index = 0; index < size; index++) {
        cout << names[index] << " is " << ages[index] << " year
    }
}

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

    displayArray(arry, 5);
}

void demonstrateDynamicArray() {
    list arry;
```

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

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

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

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

Output

```
array[0] = 49
array[1] = 48
array[2] = 26
array[3] = 19
```

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

References

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

C# Examples

DAVE BRAUNSCHWEIG

Arrays

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

using System;
using System.Collections.Generic;

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

        DisplayArray(ages);

        int total = CalculateSum(ages);
        int maximum = CalculateMaximum(ages);
        int minimum = CalculateMinimum(ages);

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

        DemonstrateParallelArrays(names, ages);

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

```
    DemonstrateFixedArray();
    DemonstrateDynamicArray();
    DemonstrateMultidimensionalArray();
}

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

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

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

```
}

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

public static void DemonstrateParallelArrays(String[] names
{
    for (int index = 0; index < names.Length; index++)
    {
        Console.WriteLine(names[index] + " is " +
            ages[index] + " years old");
    }
}

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

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

```
}

public static void DemonstrateDynamicArray()
{
    List<int> array = new List<int>();

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

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

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

```
                }
            }
            Console.WriteLine();
        }
    }
}
```

Output

```
array[0] = 49
array[1] = 48
array[2] = 26
array[3] = 19
array[4] = 16
total: 158
maximum: 49
minimum: 16
Lisa is 49 years old
Michael is 48 years old
Ashley is 26 years old
Jacob is 19 years old
Emily is 16 years old
array[0] = 16
array[1] = 19
array[2] = 26
array[3] = 48
array[4] = 49
array[0] = 65
array[1] = 45
array[2] = 78
array[3] = 32
array[4] = 4
array[0] = 24
array[1] = 62
```

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

References

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

Java Examples

DAVE BRAUNSCHWEIG

Arrays

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

import java.util.*;

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

        displayArray(ages);

        int total = calculateSum(ages);
        int maximum = calculateMaximum(ages);
        int minimum = calculateMinimum(ages);

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

        demonstrateParallelArrays(names, ages);

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

        demonstrateFixedArray();
```

```
    demonstrateDynamicArray();
    demonstrateMultidimensionalArray();
}

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

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

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

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

```
    }

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

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

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

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

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

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

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

Output

```
array[0] = 49
array[1] = 48
array[2] = 26
array[3] = 19
array[4] = 16
total: 158
maximum: 49
minimum: 16
Lisa is 49 years old
Michael is 48 years old
Ashley is 26 years old
Jacob is 19 years old
Emily is 16 years old
array[0] = 16
array[1] = 19
array[2] = 26
array[3] = 48
array[4] = 49
array[0] = 28
array[1] = 30
```

```
array[2] = 28
array[3] = 75
array[4] = 21
[56, 50, 63, 82, 15]
X | O | X
O | O | O
X | O | X
```

References

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

JavaScript Examples

DAVE BRAUNSCHWEIG

Arrays

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

main()

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

    displayArray(names);
    displayArray(ages);

    var total = calculateSum(ages);
    var maximum = calciulateMaximum(ages);
    var minimum = calculateMinimum(ages);

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

    demonstrateParallelArrays(names, ages);

    ages.sort();
    displayArray(ages);

    demonstrateFixedArray();
```

```
    demonstrateDynamicArray();
    demonstrateMultidimensionalArray();
}

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

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

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

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

```
    return minimum;
}

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

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

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

function demonstrateDynamicArray() {
    var array = [];

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

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

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

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

Output

```
array[0] = Lisa
array[1] = Michael
array[2] = Ashley
array[3] = Jacob
array[4] = Emily
array[0] = 49
array[1] = 48
array[2] = 26
```

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

References

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

Python Examples

DAVE BRAUNSCHWEIG

Arrays

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

import random

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


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


def calculate_maximum(array):
    maximum = array[0]
    for index in range(1, len(array)):
        if maximum < array[index]:
            maximum = array[index]
    return maximum
```

```
def calculate_minimum(array):
    minimum = array[0]
    for index in range(1, len(array)):
        if minimum > array[index]:
            minimum = array[index]
    return minimum


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


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


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


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

    for row in range (0, 3):
```

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


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

    display_array(names)
    display_array(ages)

    total = calculate_sum(ages)
    maximum = calculate_maximum(ages)
    minimum = calculate_minimum(ages)

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

    demonstrate_parallel_arrays(names, ages)

    ages.sort()
    display_array(ages)

    demonstrate_fixed_array()
    demonstrate_dynamic_array()
    demonstrate_multidimensional_array()


main()
```

Output

```
array[0] = Lisa
array[1] = Michael
array[2] = Ashley
array[3] = Jacob
array[4] = Emily
array[0] = 49
array[0] = Lisa
array[1] = Michael
array[2] = Ashley
array[3] = Jacob
array[4] = Emily
array[0] = 49
array[1] = 48
array[2] = 26
array[3] = 19
array[4] = 16
total: 158
maximum: 49
minimum: 16
Lisa is 49 years old
Michael is 48 years old
Ashley is 26 years old
Jacob is 19 years old
Emily is 16 years old
array[0] = 16
array[1] = 19
array[2] = 26
array[3] = 48
array[4] = 49
array[0] = 18
array[1] = 14
array[2] = 59
array[3] = 99
```

```
array[4] = 61
array[0] = 85
array[1] = 4
array[2] = 35
array[3] = 45
array[4] = 93
X | O | X
O | O | O
X | O | X
```

References

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

Swift Examples

DAVE BRAUNSCHWEIG

Arrays

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

import Foundation

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

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

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

    srand(UInt32(time(nil)))
    for index in 0...4 {
```

```
        array[index] = random() % 100
    }
    print(array)
}

func demonstrateDynamicArray() {
    var array: [Int] = []

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

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

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

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

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

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

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

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

    demonstrateParallelArrays(names:names, ages:ages)

    ages.sort()
    displayArray(array:ages)

    demonstrateFixedArray()
    demonstrateDynamicArray()
    demonstrateMultidimensionalArray()
}

main()
```

Output

```
array[0] = 49
```

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

References

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

Practice: Arrays

KENNETH LEROY BUSBEE

Review Questions

True / False

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

Answers:

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

8. true
9. false
10. true

Short Answer

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

Activities

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

Defined-Value Arrays

1. Review MathsIsFun: Leap Years. Create a program that displays the number of days in a given month. Start by asking the user to enter a year and month number. Use a defined array to look up the corresponding month name (January, February, etc.). Use another defined array to look up the corresponding number of days in the month (January = 31, February = 28 or 29 depending on year, etc.). Display results similar to the following:

   ```
   February 2020 has 29 days
   ```

2. Review Wikipedia: Zeller's congruence. Create a program that asks the user for their birthday (year, month, and day) and then calculate and display the day of the week on which they were born. Use a defined array to look up the numeric day of the week and display the corresponding string representation (Monday, Tuesday, Wednesday, etc.).

Fixed-Length Arrays

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

Dynamic Arrays / Lists

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

References

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

CHAPTER VII

STRINGS AND FILES

Overview

This chapter introduces string and file processing.

Chapter Outline

- Strings
- String Functions
- String Formatting
- File Input and Output
- Exception Handling
- Loading an Array from a File
- Code Examples
  - Program Plan
  - 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 processes strings to solve a given problem.

3. Given example pseudocode, flowcharts, and source code, create a program that processes a text file to solve a given problem.

Strings

Overview

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

Discussion

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

Most data is more complex than just one character, integer, etc. Programming languages develop other methods to represent and store data that are more complex. A complex data type of array is the first most students encounter. An array

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

is a sequenced collection of elements of the same data type with a single identifier name. This definition perfectly describes our string data type concept. The simplest array is called a one-dimensional array; also know as a list because we usually list the members or elements vertically. However, strings are viewed as a one-dimensional array that visualize as listed horizontally. Strings are an array of character data.

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

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

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

3. Wikipedia: Buffer overflow

Key Terms

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

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

concatenation
: Combining two strings into one string.

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

References

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

String Functions

DAVE BRAUNSCHWEIG

Overview

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

Discussion

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

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

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

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

Key Terms

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

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

References

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

String Formatting

Overview

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

Discussion

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

1. Wikipedia: String interpolation

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

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

Key Terms

code injection

The exploitation of a computer bug that is caused by

2. Wikipedia: String interpolation

processing invalid data.[3]

formatting

Modifying the way the output is displayed.

string interpolation

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

References

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

3. Wikipedia: Code injection

File Input and Output

KENNETH LEROY BUSBEE

Overview

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

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

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

Discussion

In computer programming, standard **streams** are pre-connected input and output communication channels between a computer program and its environment when it begins execution. The three input/output (I/O) connections are called standard input (**stdin** – keyboard), standard output (**stdout** – originally a printer) and standard error (**stderr** –

1. Wikipedia: Computer file

monitor). Streams may be redirected to other devices and/or files. In current environments, stdout is usually redirected to the monitor.[2]

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

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

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

Filename – The name and its extension. Most operating

2. Wikipedia: Standard streams

systems have restrictions on which characters can be used in filenames. Example Lab_05.txt

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

Path (Filespec) – The location of a file along with its filename. Filespec is short for file specification. Most operating systems have a set of rules on how to specify the drive and directory (or path through several directory levels) along with the filename. Example: C:\myfiles\cosc_1436\Lab_05.txt

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

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

```
Declare File datafile

datafile = open(filespec, mode)
```

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

It is considered good programming practice to determine if the file was opened properly. The reason the operating system usually can't open a file is because the filespec is wrong

(misspelled or not typed case consistent in some operating systems) or the file is not stored in the location specified. Accessing files stored on a network or the Internet may fail due to a network error.

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

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

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

or

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

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

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

or

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

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

```
close(datafile)
```

or

```
datafile.close()
```

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

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

or

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

Key Terms

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

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

filename
: The name and its extension.

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

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

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

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

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

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

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

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

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

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

References

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

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

Loading an Array from a Text File

Overview

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

Discussion

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

Either:

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

Or:

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

Processing Records

There are two options for file processing:

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

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

Processing Fields

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

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

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

Pseudocode

Static Array Processing

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

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

Dynamic Array Processing

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

Key Terms

dynamic memory
: Aka stack created memory associated with local scope.

static memory
: Aka data area memory associated with global scope.

References

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

Program Plan

This program demonstrates string functions.

Main Program
  Demonstrate string concatenation
  Demonstrate lower case
  Demonstrate upper case
  Demonstrate find
  Demonstrate length
  Demonstrate replace
  Demonstrate reverse
  Demonstrate slice
  Demonstrate strip
  Demonstrate number formatting

This program demonstrates reading a text file with exception handling.

Main Program
  Read File

Read File
  Parameters:
    Filename
  Process:
    Create exception handler
    Open file
    While not End-of-File
      Read line
      Display line
    Close file

Handle exceptions
Return Value:
None

C++ Examples

DAVE BRAUNSCHWEIG

Strings

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

using namespace std;

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

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

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

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

}

string toLower(string str) {
```

```
    transform(str.begin(), str.end(), str.begin(), ::tolower);

    return str;
}

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

    return str;
}
```

Output

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

Files

```
// This program demonstrates reading a text file with error han

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

#include <fstream>
#include <iostream>
#include <string>
```

```
using namespace std;

void readFile(string);

int main() {
    string FILENAME = "temperature.txt";

    readFile(FILENAME);
}

void readFile(string filename)
{
    fstream file;
    string line;

    file.open(filename, fstream::in);
    if (file.is_open()) {
        while (getline(file, line))
        {
            cout << line << endl;
        }
        file.close();
    } else {
        cout << "Error reading " << filename << endl;
    }
}
```

Output

```
Celsius,Fahrenheit
0,32
10,50
20,68
```

```
...
80,176
90,194
100,212
```

References

- Wikiversity: Computer Programming

C# Examples

DAVE BRAUNSCHWEIG

Strings

```
// This program demonstrates string functions.

using System;

class Strings
{
    public static void Main (string[] args)
    {
        String str = "Hello";

        Console.WriteLine("string: " + str);
        Console.WriteLine("string.ToLower(): " + str.ToLower())
        Console.WriteLine("string.ToUpper(): " + str.ToUpper())
        Console.WriteLine("string.IndexOf('e'): " + str.IndexOf
        Console.WriteLine("string.Length: " + str.Length);
        Console.WriteLine("string.Replace('H', 'j'): " + str.Re
        Console.WriteLine("string(Substring(2, 2)): " + str.Sub
        Console.WriteLine("string.Trim(): " + str.Trim());

        String name = "Bob";
        double value = 123.456;
        Console.WriteLine(String.Format("{0} earned {1:$0.00}",
    }
}
```

Output

```
string: Hello
string.ToLower(): hello
string.ToUpper(): HELLO
string.IndexOf('e'): 1
string.Length: 5
string.Replace('H', 'j'): jello
string(Substring(2, 2)): ll
string.Trim(): Hello
Bob earned $123.46
```

Files

```
// This program demonstrates reading a text file with exception

// References:
//     https://en.wikibooks.org/wiki/C_Sharp_Programming

using System;

public class Files
{
    public static void Main(String[] args)
    {
        string FILENAME = "temperatures.txt";

        ReadFile(FILENAME);
    }

    private static void ReadFile(string filename)
    {
        System.IO.StreamReader file;
```

```
        string line;

        try
        {
            using (file = System.IO.File.OpenText(filename))
            {
                while (true)
                {
                    line = file.ReadLine();
                    if (line == null)
                    {
                        break;
                    }
                    Console.WriteLine(line);
                }
            }
        }
        catch(Exception exception)
        {
            Console.WriteLine("Error reading " + filename);
            Console.WriteLine(exception.Message);
        }
    }
}
```

Output

```
Celsius,Fahrenheit
0,32
10,50
20,68
...
80,176
90,194
```

```
100,212
```

References

- Wikiversity: Computer Programming

Java Examples

DAVE BRAUNSCHWEIG

Strings

```
// This program demonstrates string functions.

class Main {
    public static void main(String[] args) {
        String str = "Hello";

        System.out.println("string: " + str);
        System.out.println("string.toLowerCase(): " + str.toLow
        System.out.println("string.toUpperCase(): " + str.toUpp
        System.out.println("string.indexOf('e'): " + str.indexO
        System.out.println("string.length(): " + str.length());
        System.out.println("string.replace('H', 'j'): " + str.r
        System.out.println("string(substring(2,4): " + str.subs
        System.out.println("string.trim(): " + str.trim());

        String name = "Bob";
        double value = 123.456;
        System.out.println(String.format("%s earned $%.2f", nam
    }
}
```

Output

```
string: Hello
string..toLowerCase(): hello
```

```
string.toUpperCase(): HELLO
string.indexOf('e'): 1
string.length(): 5
string.replace('H', 'j'): jello
string(substring(2,4): ll
string.trim(): Hello
Bob earned $123.46
```

Files

```
// This program demonstrates reading a text file with exception

// References:
//     https://en.wikibooks.org/wiki/Java_Programming

import java.util.*;

class Main {
    public static void main(String[] args) {
        String FILENAME = "temperature.txt";

        readFile(FILENAME);
    }

    private static void readFile(String filename) {
        try {
            java.io.File file = new java.io.File(filename);
            java.io.BufferedReader reader =
                new java.io.BufferedReader(new java.io.FileRead
            String line;

            while(true) {
                line = reader.readLine();
                if (line == null) {
```

```
                    break;
                }
                System.out.println(line);
            }
            reader.close();
            System.out.println("");
        } catch(Exception exception) {
            System.out.println("Error reading " + filename);
            exception.printStackTrace();
        }
    }
}
```

Output

```
Celsius,Fahrenheit
0,32
10,50
20,68
...
80,176
90,194
100,212
```

References

- Wikiversity: Computer Programming

JavaScript Examples

DAVE BRAUNSCHWEIG

Strings

```
// This program demonstrates string functions.

main();

function main()
{
    var str = "Hello";

    output("string: " + str);
    output("string.toLowerCase(): " + str.toLowerCase());
    output("string.toUpperCase(): " + str.toUpperCase());
    output("string.indexOf('e'): " + str.indexOf('e'));
    output("string.length: " + str.length);
    output("string.replace('H', 'j'): " + str.replace('H', 'j')
    output("string(substring(2,4): " + str.substring(2, 4));
    output("string.trim(): " + str.trim());

    var name = "Bob";
    var value = 123.456;
    output(`string.format(): ${name} earned $${value.toFixed(2)
}

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

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

Output

```
string: Hello
string..toLowerCase(): hello
string.toUpperCase(): HELLO
string.indexOf('e'): 1
string.length: 5
string.replace('H', 'j'): jello
string(substring(2,4): ll
string.trim(): Hello
string.format(): Bob earned $123.46
```

Files

Note: For security reasons, JavaScript in a browser requires the user to select the file to be processed. This example is based on node.js rather than browser-based JavaScript.

```
// This program demonstrates reading a text file with exception

const fs = require('fs');

main();

function main() {
    const filename = "temperature.txt";
```

```
    readFile(filename);
}

function readFile(filename) {
    try {
        const text = fs.readFileSync(
            filename,
            {encoding:"utf8"});

        const lines = text.split("\n");

        for (const line of lines) {
            console.log(line);
        }
    } catch (exception) {
        console.log(exception)
    }
}
```

Output

```
Celsius,Fahrenheit
0,32
10,50
20,68
...
80,176
90,194
100,212
```

References

- Wikiversity: Computer Programming

Python Examples

DAVE BRAUNSCHWEIG

Strings

```
# This program demonstrates string functions.


def main():
    string = "Hello"

    print("string: " + string)
    print("string.lower(): " + string.lower())
    print("string.upper(): " + string.upper())
    print("string.find('e'): " + str(string.find('e')))
    print("len(string): " + str(len(string)))
    print("string.replace('H', 'j'): " + string.replace('H', 'j
    print("string[::-1]: " + string[::-1])
    print("string[2:4]: " + string[2:4])
    print("string.strip('H'): " + string.strip('H'))

    name = "Bob"
    value = 123.456
    print("string.format(): {0} earned ${1:.2f}".format(name, v


main()
```

Output

```
string: Hello
string.lower(): hello
string.upper(): HELLO
string.find('e'): 1
len(string): 5
string.replace('H', 'j'): jello
string[::-1]: olleH
string[2:4]: ll
string.strip('H'): ello
string.format(): Bob earned $123.46
```

Files

```
# This program demonstrates reading a text file with exception
#
# References:
#     https://en.wikibooks.org/wiki/Python_Programming


def read_file(filename):
    try:
        with open(filename, "r") as file:
            for line in file:
                line = line.strip()
                print(line)
    except Exception as exception:
        print(exception)


def main():
    filename = "temperature.txt"
```

```
    read_file(filename)


main()
```

Output

```
Celsius,Fahrenheit
0,32
10,50
20,68
...
80,176
90,194
100,212
```

References

- Wikiversity: Computer Programming

Swift Examples

DAVE BRAUNSCHWEIG

Strings

```
// This program demonstrates string functions.

import Foundation

func main() {
    let string:String = "Hello"

    print("string: " + string)
    print("string.lowercased(): " + string.lowercased())
    print("string.uppercased(): " + string.uppercased())
    print("find(string, \"e\"): " + String(find(string:string,
    print("string.count: " + String(string.count))
    print("string.replacingOccurrences(of:\"H\", with:\"j\"): "
    print("string.reversed(): " + String(string.reversed()))
    print("substring(2, 2): " + substring(string:string, start:
    print("string.trimmingCharacters(\"H\"): " + string.trimmin

    let name:String = "Bob"
    let value:Double = 123.456
    print("\(name) earned $" + String(format:"%.2f", value))
}

func find(string:String, character:Character) -> Int {
    var result: Int

    if let index = string.firstIndex(of:character) {
        result = string.distance(from: string.startIndex, to: i
```

```
    } else {
        result = -1
    }
    return result
}

func substring(string:String, start:Int, length:Int) -> String
    let startIndex = string.index(string.startIndex, offsetBy:
    let endIndex = string.index(string.startIndex, offsetBy: st
    return String(string[startIndex...endIndex])
}

main()
```

Output

```
string: Hello
string.lowercased(): hello
string.uppercased(): HELLO
find(string, "e"): 1
string.count: 5
string.replacingOccurrences(of:"H", with:"j"): jello
string.reversed(): olleH
substring(2, 2): ll
string.trimmingCharacters("H"): ello
Bob earned $123.46
```

Files

```
// This program demonstrates reading a text file with exception

// References:
```

```
//  https://developer.apple.com/library/content/documentation/S

import Foundation

func readFile(filename:String) {
    var text = ""

    do {
        text = try String(contentsOfFile: filename, encoding: .

        let lines = text.components(separatedBy:"\n")
        for line in lines {
            print(line)
        }
    } catch {
        print("Error reading " + filename)
        print(error.localizedDescription)
    }
}

func main() {
    let filename:String = "temperature.txt"

    readFile(filename:filename)
}

main()
```

Output

```
Celsius,Fahrenheit
0,32
10,50
20,68
```

```
...
80,176
90,194
100,212
```

References

- Wikiversity: Computer Programming

Practice: Strings and Files

KENNETH LEROY BUSBEE

Review Questions

True / False

1. The character data type in C++ uses the double quote marks, like: char grade = “A”;
2. Sizeof is an operator that tells you how many bytes a data type occupies in storage.
3. Typedef helps people who can’t hear and is one of the standard accommodation features of a programming language for people with a learning disability.
4. The sequence operator should be used when defining variables in order to save space.
5. Programming can be both enjoyable and frustrating.
6. Text files are hard to create.
7. A filespec refers to a very small (like a spec dust) file.
8. A device token is a special non zero value the operating system gives your program and is associated with the file that you requested to be opened.
9. Programmers should not worry about closing a file.
10. Where you define an item, that is global or local scope, is rarely important.

Answers:

1. false

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

Short Answer

1. Describe the normal operations allowed with the string data type.
2. Describe why unary positive is worthless.
3. Describe how unary negative works.

Activities

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

String Activities

1. Create a program that asks the user for a single line of text containing a first name and last name, such as `Firstname`

`Lastname`. Use string functions/methods to parse the line and print out the name in the form last name, first initial, such as `Lastname, F.` Include a trailing period after the first initial. Handle invalid input errors, such as extra spaces or missing name parts.

2. Create a program that asks the user for a line of text. Use string functions/methods to delete leading, trailing, and duplicate spaces, and then print the line of text backwards. For example:

```
   the   cat   in   the   hat
tah eht ni tac eht
```

3. Create a program that asks the user for a line of comma-separated-values. It could be a sequence of test scores, names, or any other values. Use string functions/methods to parse the line and print out each item on a separate line. Remove commas and any leading or trailing spaces from each item when printed.
4. Create a program that asks the user for a line of text. Then ask the user for the number of characters to print in each line, the number of lines to be printed, and a scroll direction, right or left. Using the given line of text, duplicate the text as needed to fill the given number of characters per line. Then print the requested number of lines, shifting the entire line's content by one character, left or right, each time the line is printed. The first or last character will be shifted / appended to the other end of the string. For example:

```
Repeat this. Repeat this.
epeat this. Repeat this. R
peat this. Repeat this. Re
```

File Activities

Note: Each of the following activities uses code only to read the file. It is not necessary to use code to create the file.

1. Using a text editor or IDE, copy the following list of names and grade scores and save it as a text file named `scores.txt`:

   ```
   Name,Score
   Joe Besser,70
   Curly Joe DeRita,0
   Larry Fine,80
   Curly Howard,65
   Moe Howard,100
   Shemp Howard,85
   ```

   Create a program that displays high, low, and average scores based on input from `scores.txt`. Verify that the file exists and then use string functions/methods to parse the file content and add each score to an array. Display the array contents and then calculate and display the high, low, and average score. Format the average to two decimal places. Note that the program must work for any given number of scores in the file. Do not assume there will always be six scores.
2. Create a program that displays high, low, and average scores based on input from `scores.txt`. Verify that the file exists and then use string functions/methods to parse the file content and add each score to an array. Display the array contents and then calculate and display the high, low, and average score. Format the average to two decimal places. Include error handling in case the file is formatted incorrectly. Note that the program must work for any given number of scores in the file. Do not assume there will always be six scores.

3. Create a program that asks the user for the name of a text/HTML file that contains HTML tags, such as:
   ```
   <p><strong>This is a bold
   paragraph.</strong></p>
   ```
   Verify that the file exists and then use string methods to search for and remove all HTML tags from the text, saving each removed tag in an array. Display the untagged text and then display the array of removed tags. For example:
   ```
   This is a bold paragraph.
   <p>
   <strong>
   </strong>
   </p>
   ```
4. Using a text editor or IDE, create a text file of names and addresses to use for testing based on the following format:
   ```
   Firstname Lastname
   123 Any Street
   City, State/Province/Region PostalCode
   ```
   Include a blank line between addresses, and include at least three addresses in the file. Create a program that verifies that the file exists, and then processes the file and displays each address as a single line of comma-separated values in the form:
   ```
   Lastname, Firstname, Address, City, State/
   Province/Region, PostalCode
   ```

References

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

Exception Handling

Overview

Exception handling is the process of responding to the occurrence of exceptions – anomalous or exceptional conditions requiring special processing – during the execution of a program. In general, an exception breaks the normal flow of execution and executes a pre-registered exception handler.[1]

Discussion

One of the challenges of file processing is that the actual input and output is beyond program control. Behind the scenes, the program is paused while the operating system uses interrupts to request that the storage device read from or write to the file. If a read is successful, data is returned to the program. If the read is unsuccessful, an exception occurs. Possible exception reasons include File Not Found, OS Error, IO Error, Permission Error, Timeout Error, Memory Error, Buffer Error, Encoding Error, etc.

Most current programming languages provide some type of support for exceptions and exception handling.

1. Wikipedia: Exception handling

Language	Key words	Examples

C++
C#
Java
JavaScript
Swift
	throw
	try
	catch
	finally
		throw // an exception
		try {
		// code statements
		} catch {
		// handle exception
		} finally {
		// clean up
		}

Python	raise try except finally	raise # an exception try: # code statements except: # handle exception finally: # clean up

Swift	defer
	throw
	do try
	catch
		defer {
		// clean up
		}
		throw // an exception
		do {
		try // code statements
		} catch {
		// handle exception
		}

Exception handling should always be used with any type of processing that is beyond program control.

Key Terms

exception

Anomalous or exceptional conditions requiring special processing.[2]

interrupt

A request for the processor to interrupt currently executing code (when permitted), so that the event can be processed in a timely manner.[3]

References

2. Wikipedia: Exception handling
3. Wikipedia: Interrupt

CHAPTER VIII

OBJECT-ORIENTED PROGRAMMING

Overview

This chapter introduces object-oriented programming, with a focus on understanding object-oriented concepts and terminology. It includes short examples of objects and classes in different programming languages.

Chapter Outline

Learning Objectives

1. Understand key terms and definitions.

2. Gain exposure to object-oriented programming.
3. Given example source code, create a program that uses object-oriented programming concepts to solve a given problem.

Objects and Classes

DAVE BRAUNSCHWEIG

Overview

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). There is significant diversity of OOP languages, but the most popular ones are class-based, meaning that objects are instances of classes, which typically also determine their type.[1]

Discussion

Thus far, we have focused on procedural programming. Based on structured programming, procedures (routines, subroutines, or functions) contain a series of computational steps to be carried out. Any given procedure might be called at any point during a program's execution, including by other procedures or itself. The focus of procedural programming is to break down a programming task into a collection of variables, data structures, and subroutines.[2] Small programs and scripts

1. Wikipedia: Object-oriented programming
2. Wikipedia: Object-oriented programming

tend to be easier to develop using a simple procedural approach.

Object-oriented programming instead breaks down a programming task into objects that expose behavior (methods) and data (members or attributes) using interfaces. The most important distinction is that while procedural programming uses procedures to operate on separate data structures, object-oriented programming bundles the two together, so an "object", which is an instance of a class, operates on its "own" data structure.[3] Larger programs benefit from better code and data isolation and reuse provided by an object-oriented approach.

Objects and classes are often designed to represent real-world objects. Consider a door as an example of a real-world object. Most doors have limited functionality. They may be opened and closed, and locked and unlocked. In procedural programming, we might design functions to open, close, lock, and unlock a door, such as:

```
Procedural Programming - Functions
OpenDoor(door)
CloseDoor(door)
LockDoor(door)
UnlockDoor(door)
```

Object-oriented programming combines code and data, so that, rather than having separate functions act on doors, we design doors that have methods that can act on themselves. Methods represent something the object can do, and are typically defined using verbs. Object-oriented door pseudocode might look like:

3. Wikipedia: Object-oriented programming

```
Object-Oriented Programming - Methods
door.Open()
door.Close()
door.Lock()
door.Unlock()
```

Objects may also have attributes, something the object is or has, and are typically defined using nouns or adjectives. Door attributes might include:

```
Object-Oriented Programming - Attributes
door.Height
door.Width
door.Color
door.Closed
door.Locked
```

When we write code to define a generic door, we would create a door class. The door class would contain all of the methods a door can perform and all of the attributes a door might have. We would then create instances of the class (objects) to represent specific doors, such as a front door, back door, or room door on a house, or a left door and right door on a car.

Key Terms

attribute
: A specification that defines a property of an object.[4]

class
: An extensible program-code-template for creating objects, providing initial values for state (member

4. Wikipedia: Attribute (computing)

variables) and implementations of behavior (member functions or methods).[5]

instance

:A concrete occurrence of an object.[6]

method

A specification that defines a procedure or behavior of an object.[7]

object

A particular instance of a class where the object can be a combination of variables, functions, and data structures.[8]

this, self, or Me

Keywords used in some computer programming languages to refer to the object, class, or other entity that the currently running code is part of.[9]

References

- Wikibooks: Object-Oriented Programming
- Wikipedia: Object-oriented programming
- Wikiversity: Computer Programming

5. Wikipedia: Class (computer programming)
6. Wikipedia: Instance (computer science)
7. Wikipedia: Method (computer programming)
8. Wikipedia: Object (computer science)
9. Wikipedia: this (computer programming)

Encapsulation

DAVE BRAUNSCHWEIG

Overview

Encapsulation is one of the fundamentals of OOP (object-oriented programming). It refers to the bundling of data with the methods that operate on that data. Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties' direct access to them. Publicly accessible methods are generally provided in the class (so-called getters and setters) to access the values, and other client classes call these methods to retrieve and modify the values within the object.[1]

Discussion

The most important principle of object orientation is *encapsulation*: the idea that data inside the object should only be accessed through a public *interface* – that is, the object's methods.

If we want to use the data stored in an object to perform an action or calculate a derived value, we define a method associated with the object which does this. Then whenever we want to perform this action we call the method on the object. We consider it bad practice to retrieve the information from

1. Wikipedia: Encapsulation (computer programming)

inside the object and write separate code to perform the action outside of the object.

Encapsulation is a good idea for several reasons:

- the functionality is defined *in one place* and not in multiple places.
- it is defined in a logical place – the place where the data is kept.
- data inside our object is not modified unexpectedly by external code in a completely different part of our program.
- when we use a method, we only need to know what result the method will produce – we don't need to know details about the object's internals in order to use it. We could switch to using another object which is completely different on the inside, and not have to change any code because both objects have the same interface.

We can say that the object "knows how" to do things with its own data, and it's a bad idea for us to access its internals and do things with the data ourselves. If an object doesn't have an interface method which does what we want to do, we should add a new method or update an existing one.

Some languages have features which allow us to enforce encapsulation strictly. In Java or C++, we can define access permissions on object attributes, and make it illegal for them to be accessed from outside the object's methods. In Java it is also considered good practice to write setters and getters for all attributes, even if the getter simply retrieves the attribute and the setter just assigns it the value of the parameter which you pass in.

In Python, encapsulation is not enforced by the language, but there is a convention that we can use to indicate that a

property is intended to be private and is not part of the object's public interface: we begin its name with an underscore. Python also supports the use of a property decorator to replace a simple attribute with a method without changing the object's interface.

Key Terms

abstraction
: A technique for arranging complexity of computer systems so that functionality may be separated from specific implementation details.[2][3]

accessor
: A method used to return the value of a private member variable, also known as a getter method.[4]

encapsulation
: A language mechanism for restricting direct access to some of an object's components.[5]

information hiding
: The principle of segregation of the design decisions in a computer program from other parts of the program. See encapsulation.[6]

mutator
: A method used to control changes to a private member variable, also known as a setter method.[7]

2. Wikipedia: Object-oriented programming
3. Wikipedia: Abstraction (computer science)
4. Wikipedia: Mutator method
5. Wikipedia: Encapsulation (computer programming)
6. Wikipedia: Information hiding
7. Wikipedia: Mutator method

private
: An access modifier that restricts visibility of a property or method to the class in which it is defined.[8]

public
: An access modifier that opens visibility of a property or method to all other classes.[9]

References

- Read the Docs: Object-Oriented Programming in Python
- Wikiversity: Computer Programming

8. Wikipedia: Access modifiers
9. Wikipedia: Access modifiers

Inheritance and Polymorphism

DAVE BRAUNSCHWEIG

Overview

In object-oriented programming, **inheritance** is the mechanism of basing an object or class upon another object (prototypical inheritance) or class (class-based inheritance), retaining similar implementation. In most class-based object-oriented languages, an object created through inheritance (a "child object") acquires all the properties and behaviors of the parent object (except: constructors, destructor, overloaded operators and friend functions of the base class). Inheritance allows programmers to create classes that are built upon existing classes, to specify a new implementation while maintaining the same behaviors (realizing an interface), to reuse code and to independently extend original software via public classes and interfaces.[1]

Discussion

Inheritance is a way of arranging objects in a hierarchy from the most general to the most specific. An object which *inherits* from another object is considered to be a *subtype* of that object. An example might include Instructor

1. Wikipedia: Inheritance (object-oriented programming)

and Student, each of which inherit from Person. When we can describe the relationship between two objects using the phrase *is-a*, that relationship is inheritance.

We also often say that a class is a *subclass* or *child class* of a class from which it inherits, or that the other class is its *superclass* or *parent class*. We can refer to the most generic class at the base of a hierarchy as a *base class*.

Inheritance can help us to represent objects which have some differences and some similarities in the way they work. We can put all the functionality that the objects have in common in a base class, and then define one or more subclasses with their own custom functionality.

Inheritance is also a way of reusing existing code easily. If we already have a class which does *almost* what we want, we can create a subclass in which we partially override some of its behavior, or perhaps add some new functionality.

In some statically typed languages inheritance is very popular because it allows the programmer to work around some of the restrictions of static typing. If an instructor and a student are both a kind of person, we can write a function which accepts a parameter of type Person and have it work on both instructor and student objects because they both inherit from Person. This is known as *polymorphism*.

Key Terms

inheritance

An object or class being based on another object or class, using the same implementation or specifying a new implementation to maintain the same behavior.[2]

polymorphism

The provision of a single interface to entities of different types.[3]

References

- Read the Docs: Object-Oriented Programming in Python
- Wikiversity: Computer Programming

2. Wikipedia: Inheritance (object-oriented programming)
3. Wikipedia: Polymorphism (computer science)

C++ Examples

DAVE BRAUNSCHWEIG

Objects

```
// This class converts temperature between Celsius and Fahrenhe
// It may be used by assigning a value to either Celsius or Fah
// and then retrieving the other value, or by calling the ToCel
// ToFahrenheit methods directly.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/C%2B%2B_Programming

#include <iostream>

using namespace std;

class Temperature {
    public:
        double getCelsius(void);
        void setCelsius(double value);
        double getFahrenheit(void);
        void setFahrenheit(double value);
        double toCelsius(double fahrenheit);
        double toFahrenheit(double celsius);

    private:
        double celsius;
        double fahrenheit;
};
```

```
double Temperature::getCelsius(void) {
    return celsius;
}

void Temperature::setCelsius(double value) {
    celsius = value;
    fahrenheit = toFahrenheit(celsius);
}

double Temperature::getFahrenheit(void) {
    return fahrenheit;
}

void Temperature::setFahrenheit(double value) {
    fahrenheit = value;
    celsius = toCelsius(fahrenheit);
}

double Temperature::toCelsius(double fahrenheit) {
    return (fahrenheit - 32) * 5 / 9;
}

double Temperature::toFahrenheit(double celsius) {
    return celsius * 9 / 5 + 32;
}

// This program creates instances of the Temperature class to c
// and Fahrenheit temperatures.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/C%2B%2B_Programming

int main() {
    Temperature temp1;
```

```
    temp1.setCelsius(100.0);
    cout << "temp1.celsius = " << temp1.getCelsius() << endl;
    cout << "temp1.fahrenheit = " << temp1.getFahrenheit() << e
    cout << endl;

    Temperature temp2;
    temp2.setFahrenheit(100.0);
    cout << "temp2.fahrenheit = " << temp2.getFahrenheit() << e
    cout << "temp2.celsius = " << temp2.getCelsius() << endl;
}
```

Output

```
temp1.celsius = 100
temp1.fahrenheit = 212

temp2.fahrenheit = 100
temp2.celsius = 37.7778
```

References

- Wikiversity: Computer Programming

C# Examples

DAVE BRAUNSCHWEIG

Objects

```
// This program creates instances of the Temperature class to c
// and Fahrenheit temperatures.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/C_Sharp_Programming

using System;

public class Objects
{
    public static void Main(String[] args)
    {
        Temperature temp1 = new Temperature(celsius: 0);
        Console.WriteLine("temp1.Celsius = " + temp1.Celsius.To
        Console.WriteLine("temp1.Fahrenheit = " + temp1.Fahrenh
        Console.WriteLine("");

        temp1.Celsius = 100;
        Console.WriteLine("temp1.Celsius = " + temp1.Celsius.To
        Console.WriteLine("temp1.Fahrenheit = " + temp1.Fahrenh
        Console.WriteLine("");

        Temperature temp2 = new Temperature(fahrenheit: 0);
        Console.WriteLine("temp2.Fahrenheit = " + temp2.Fahrenh
        Console.WriteLine("temp2.Celsius = " + temp2.Celsius.To
        Console.WriteLine("");
```

```
        temp2.Fahrenheit = 100;
        Console.WriteLine("temp2.Fahrenheit = " + temp2.Fahrenh
        Console.WriteLine("temp2.Celsius = " + temp2.Celsius.To
    }
}

// This class converts temperature between Celsius and Fahrenhe
// It may be used by assigning a value to either Celsius or Fah
// and then retrieving the other value, or by calling the ToCel
// ToFahrenheit methods directly.

public class Temperature
{
    double _celsius;
    double _fahrenheit;

    public double Celsius
    {
        get
        {
            return _celsius;
        }

        set
        {
            _celsius = value;
            _fahrenheit = ToFahrenheit(value);
        }
    }

    public double Fahrenheit
    {
        get
        {
```

```
            return _fahrenheit;
        }

        set
        {
            _fahrenheit = value;
            _celsius = ToCelsius(value);
        }
    }

    public Temperature(double? celsius = null, double? fahrenhe
    {
        if (celsius.HasValue)
        {
            this.Celsius = Convert.ToDouble(celsius);
        }

        if (fahrenheit.HasValue)
        {
            this.Fahrenheit = Convert.ToDouble(fahrenheit);
        }
    }

    public double ToCelsius(double fahrenheit)
    {
        return (fahrenheit - 32) * 5 / 9;
    }

    public double ToFahrenheit(double celsius)
    {
        return celsius * 9 / 5 + 32;
    }
}
```

Output

```
temp1.Celsius = 0
temp1.Fahrenheit = 32

temp1.Celsius = 100
temp1.Fahrenheit = 212

temp2.Fahrenheit = 0
temp2.Celsius = -17.7777777777778

temp2.Fahrenheit = 100
temp2.Celsius = 37.7777777777778
```

References

- Wikiversity: Computer Programming

Java Examples

DAVE BRAUNSCHWEIG

Objects

```
// This program creates instances of the Temperature class to c
// and Fahrenheit temperatures.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/Java_Programming

import java.util.*;

class Main {
    public static void main(String[] args) {
        Temperature temp1 = new Temperature();
        temp1.setCelsius(100.0);
        System.out.println("temp1.celsius = " + temp1.getCelsiu
        System.out.println("temp1.fahrenheit = " + temp1.getFah
        System.out.println("");

        Temperature temp2 = new Temperature();
        temp2.setFahrenheit(100.0);
        System.out.println("temp2.fahrenheit = " + temp2.getFah
        System.out.println("temp2.celsius = " + temp2.getCelsiu
    }
}

// This class converts temperature between Celsius and Fahrenhe
// It may be used by assigning a value to either Celsius or Fah
// and then retrieving the other value, or by calling the ToCel
```

```
// ToFahrenheit methods directly.

class Temperature {
    Double celsius;
    Double fahrenheit;

    public Double getCelsius() {
        return celsius;
    }

    public void setCelsius(Double value) {
        celsius = value;
        fahrenheit = toFahrenheit(celsius);
    }

    public Double getFahrenheit() {
        return fahrenheit;
    }

    public void setFahrenheit(Double value) {
        fahrenheit = value;
        celsius = toCelsius(fahrenheit);
    }

    public Double toCelsius(Double fahrenheit) {
        return (fahrenheit - 32) * 5 / 9;
    }

    public Double toFahrenheit(Double celsius) {
        return celsius * 9 / 5 + 32;
    }
}
```

Output

```
temp1.celsius = 100.0
temp1.fahrenheit = 212.0

temp2.fahrenheit = 100.0
temp2.celsius = 37.77777777777778
```

References

- Wikiversity: Computer Programming

JavaScript Examples

DAVE BRAUNSCHWEIG

Objects

```
// This class converts temperature between Celsius and Fahrenhe
// It may be used by assigning a value to either Celsius or Fah
// and then retrieving the other value, or by calling the ToCel
// ToFahrenheit methods directly.

class Temperature {
    constructor() {
        this._celsius = 0;
        this._fahrenheit = 32;
    }

    get celsius() {
        return this._celsius;
    }

    set celsius(value) {
        this._celsius = value;
        this._fahrenheit = this.toFahrenheit(value);
    }

    get fahrenheit() {
        return this._fahrenheit;
    }

    set fahrenheit(value) {
        this._fahrenheit = value;
        this._celsius = this.toCelsius(value);
```

```
    }

    toCelsius(fahrenheit) {
        return (fahrenheit - 32) * 5 / 9
    }

    toFahrenheit(celsius) {
        return celsius * 9 / 5 + 32
    }
}

// This program creates instances of the Temperature class to c
// and Fahrenheit temperatures.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://en.wikibooks.org/wiki/JavaScript

main()

function main() {
    var temp1 =  new Temperature();
    temp1.celsius = 0
    output("temp1.celsius = " + temp1.celsius);
    output("temp1.fahrenheit = " + temp1.fahrenheit);
    output("");

    temp1.celsius = 100;
    output("temp1.celsius = " + temp1.celsius);
    output("temp1.fahrenheit = " + temp1.fahrenheit);
    output("");

    var temp2 = new Temperature();
    temp2.fahrenheit = 0
    output("temp2.fahrenheit = " + temp2.fahrenheit);
```

```
    output("temp2.celsius = " + temp2.celsius);
    output("");

    temp2.fahrenheit = 100;
    output("temp2.fahrenheit = " + temp2.fahrenheit);
    output("temp2.celsius = " + temp2.celsius);
}

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

Output

```
temp1.celsius = 0
temp1.fahrenheit = 32

temp1.celsius = 100
temp1.fahrenheit = 212

temp2.fahrenheit = 0
temp2.celsius = -17.77777777777778

temp2.fahrenheit = 100
temp2.celsius = 37.77777777777778
```

References

- Wikiversity: Computer Programming

Python Examples

DAVE BRAUNSCHWEIG

Objects

```
# This class converts temperature between Celsius and Fahrenhei
# It may be used by assigning a value to either Celsius or Fahr
# and then retrieving the other value, or by calling the to_cel
# to_fahrenheit methods directly.
#
# References:
#     https://www.mathsisfun.com/temperature-conversion.html
#     https://en.wikibooks.org/wiki/Python_Programming

class Temperature:
    _celsius = None
    _fahrenheit = None

    @property
    def celsius(self):
        return self._celsius

    @celsius.setter
    def celsius(self, value):
        self._celsius = float(value)
        self._fahrenheit = self.to_fahrenheit(self._celsius)

    @property
    def fahrenheit(self):
        return self._fahrenheit

    @fahrenheit.setter
```

```
    def fahrenheit(self, value):
        self._fahrenheit = float(value)
        self._celsius = self.to_celsius(self._fahrenheit)

    def __init__(self, celsius=None, fahrenheit=None):
        if celsius != None:
            self._celsius = celsius
            self._fahrenheit = self.to_fahrenheit(celsius)
        if fahrenheit != None:
            self._fahrenheit = fahrenheit
            self._celsius = self.to_celsius(fahrenheit)

    def to_celsius(self, fahrenheit):
        return (fahrenheit - 32) * 5 / 9

    def to_fahrenheit(self, celsius):
        return celsius * 9 / 5 + 32


# This program creates instances of the Temperature class to co
# and Fahrenheit temperatures.

def main():
    temp1 = Temperature(celsius=0)
    print("temp1.celsius =", temp1.celsius)
    print("temp1.fahrenheit =", temp1.fahrenheit)
    print("")

    temp1.celsius = 100
    print("temp1.celsius =", temp1.celsius)
    print("temp1.fahrenheit =", temp1.fahrenheit)
    print("")

    temp2 = Temperature(fahrenheit=0)
    print("temp2.fahrenheit =", temp2.fahrenheit)
```

```
    print("temp2.celsius =", temp2.celsius)
    print("")

    temp2.fahrenheit = 100
    print("temp2.fahrenheit =", temp2.fahrenheit)
    print("temp2.celsius =", temp2.celsius)

main()
```

Output

```
temp1.celsius = 0
temp1.fahrenheit = 32.0

temp1.celsius = 100.0
temp1.fahrenheit = 212.0

temp2.fahrenheit = 0
temp2.celsius = -17.77777777777778

temp2.fahrenheit = 100.0
temp2.celsius = 37.77777777777778
```

References

- Wikiversity: Computer Programming

Swift Examples

DAVE BRAUNSCHWEIG

Objects

```
// This class converts temperature between Celsius and Fahrenhe
// It may be used by assigning a value to either Celsius or Fah
// and then retrieving the other value, or by calling the ToCel
// ToFahrenheit methods directly.

class Temperature {
    var _celsius:Double = 0
    var _fahrenheit:Double = 32

    init(celsius:Double?=nil, fahrenheit:Double?=nil) {
        if celsius != nil {
            self.celsius = celsius!
        }

        if fahrenheit != nil {
            self.fahrenheit = fahrenheit!
        }
    }

    var celsius: Double {
        get {
            return self._celsius
        }
        set {
            self._celsius = newValue
            self._fahrenheit = toFahrenheit(celsius:self._celsi
        }
```

```
    }

    var fahrenheit: Double {
        get {
            return self._fahrenheit
        }
        set {
            self._fahrenheit = newValue
            self._celsius = toCelsius(fahrenheit:self._fahrenhe
        }
    }

    func getCelsius() -> Double {
        return self.celsius
    }

    func setCelsius(celsius:Double) {
        self.celsius = celsius
        self.fahrenheit = toFahrenheit(celsius:celsius)
    }

    func getFahrenheit() -> Double {
        return self.fahrenheit
    }

    func setFahrenheit(fahrenheit:Double) {
        self.fahrenheit = fahrenheit
        self.celsius = toCelsius(fahrenheit:fahrenheit)
    }

    func toCelsius(fahrenheit:Double) -> Double {
        return (fahrenheit - 32) * 5 / 9
    }

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

```
        return celsius * 9 / 5 + 32
    }
}

// This program creates instances of the Temperature class to c
// and Fahrenheit temperatures.
//
// References:
//     https://www.mathsisfun.com/temperature-conversion.html
//     https://developer.apple.com/library/content/documentatio

func main() {
    let temp1 =  Temperature(celsius:0);
    print("temp1.celsius = " + String(temp1.celsius));
    print("temp1.fahrenheit = " + String(temp1.fahrenheit));
    print("");

    temp1.celsius = 100;
    print("temp1.celsius = " + String(temp1.celsius));
    print("temp1.fahrenheit = " + String(temp1.fahrenheit));
    print("");

    let temp2 = Temperature(fahrenheit:0);
    print("temp2.fahrenheit = " + String(temp2.fahrenheit));
    print("temp2.celsius = " + String(temp2.celsius));
    print("");

    temp2.fahrenheit = 100;
    print("temp2.fahrenheit = " + String(temp2.fahrenheit));
    print("temp2.celsius = " + String(temp2.celsius));
}

main()
```

Output

```
temp1.celsius = 0.0
temp1.fahrenheit = 32.0

temp1.celsius = 100.0
temp1.fahrenheit = 212.0

temp2.fahrenheit = 0.0
temp2.celsius = -17.7777777777778

temp2.fahrenheit = 100.0
temp2.celsius = 37.7777777777778
```

References

- Wikiversity: Computer Programming

Practice

KENNETH LEROY BUSBEE

Review Questions

Answer the following statements as either true or false:

1. Procedural programming and object-oriented programming cannot be done with the same compiler/IDE.
2. Object-oriented programming encapsulates data and functions.

Answers:

1. false
2. true

Short Answer

1. Describe the fundamental differences between procedural (modular structured) programming and object-oriented programming.

Activities

Complete the following activities using your selected programming language. Use separate functions for input, each type of processing, and output. Avoid global variables by

passing parameters and returning results. Create test data to validate the accuracy of each program. Add comments at the top of the program and include references to any resources used.

1. Review MathsIsFun: Area of Plane Shapes. Create a program that asks the user what shape they would like to calculate the area for. Use if/else conditional statements to determine their selection and then gather the appropriate input and calculate and display the area of the shape. Perform all area calculations using a ShapeArea class that has separate methods to calculate and return the area for different shapes. Include data validation in the class and error handling in the main program.
2. Create a program that asks the user how old they are in years. Then ask the user if they would like to know how old they are in months, days, hours, or seconds. Use if/else conditional statements to display their approximate age in the selected timeframe. Perform all calculations using an AgeConverter class that accepts the age in years during initialization and has separate properties and methods to calculate and return the age in months, days, hours, and seconds. Include data validation in the class and error handling in the main program.
3. Review Wikipedia: Zeller's congruence. Create a program that asks the user for their birthday (year, month, and day) and then calculate and display the day of the week on which they were born. Use if/else conditional statements to convert the numeric day of the week to the correct string representation (Monday, Tuesday, Wednesday, etc.). Perform all calculations using a DayOfWeek class that accepts the year, month, and day during initialization and has separate properties and methods to calculate and return the day of week as a number, as an abbreviated string (Mon, Tue, etc.), and as a full string (Monday,

Tuesday, etc.). Include data validation in the class and error handling in the main program.

References

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