Programming Fundamentals

Programming Fundamentals

A Modular Structured Approach, 2nd Edition

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

DAVE BRAUNSCHWEIG

Contents

Part IV. Often Used Data Types

Part V. Program Control Functions

Part XI. Multiway Selection

Part XII. Test After Loops

Part XIII. Test Before Loops

Part XIV. Counting Loops

Part XV. String Class, Unary Positive and Negative

Part XVI. Conditional Operator and Recursion

Part XVII. Introduction to Arrays

Part XVIII. File I/O and Array Functions

Part XIX. More Array Functions

Part XX. More on Typedef

Part XXI. Pointers

Part XXII. More Arrays & Compiler Directives

Part XXIII. OOP & HPC

Part XXIV. Review Materials

Part XXV. Appendix

A Note to Contributors

DAVE BRAUNSCHWEIG

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

If you teach an introductory programming course in any programming language, or you are a student who has recently completed an introductory programming course, your contributions are needed to make this free textbook as widely inclusive, accessible, and available as possible!

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 others.

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,

To maintain an edit history for CC-BY licensing

attribution, Kenneth Leroy Busbee's original content was copied from https://cnx.org/contents/MDgA8wfz@22.2:YzfkjC2r@17 as is, and then edited for sequence, content, and programming language support.

As you begin to review and contribute to 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.

Make heavy use of Comments at the bottom of each page to help other contributors understand how to improve the page content from your perspective.

Thank you for being willing to share your expertise to make fundamental programming concepts free, accessible, and available to all!

Dave Braunschweig

Preface

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

About this Textbook/Collection

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 this textbook/collection were originally developed by Busbee and others as independent modules for publication within the Connexions environment. The original source is available at https://cnx.org/contents/MDgA8wfz@22.2:YzfkjC2r@17/ .

This edition, adapted by Dave Braunschweig and others, expands on the original vision by supporting multiple programming languages and including example code in C++, Java, Python, and others. Additional programming languages will be included in future editions.

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/collection 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 project allows the author of a module to create web links to other modules and Internet locations. These links are shown when viewing materials on-line and are categorized into three types: Example, Prerequisite and Supplemental.

Students using this collection for a college course should note that all of the **Prerequisite links** within the modules will be modules that student should have already read and most of the **Supplemental links** will be modules that the student will read shortly. Thus, students should use Prerequisite links for review as needed and not be overly concerned about viewing all of the Supplemental links at the first reading of this textbook/collection.

CONCEPTUAL APPROACH

The learning modules of this textbook/collection 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 and pseudocode format. Language-specific examples follow the general overview.

RE-USE AND CUSTOMIZATION

PDF CONVERSION PROBLEMS

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

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

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

REFERENCES

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

Author Acknowledgements

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

1ST EDITION ACKNOWLEDGEMENTS

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

1. Mr. Abass Alamnehe, who is a fellow faculty member at Houston Community College. He has encouraged the use of Connexions as an “open source” publishing concept. His comments on several modules have led directly to the improvement of the materials in this textbook/collection.
2. The hundreds (most likely a thousand plus) students that I have taken programming courses that I have taught since 1984. The languages include: COBOL, main frame 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 content as CC-BY, making it possible to build on his success.
- Zoe Wake Hyde and the staff and volunteers at Rebus Community for providing a community and platform to create and collaborate on open content.
- Contributors and editors TBD.
- My wife and family for accepting my dedication to open educational resources and loving me anyway.

Dave Braunschweig

REFERENCES

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

Orientation and Syllabus

KENNETH LEROY BUSBEE

Note to Contributors:

Strikethrough text will be replaced or deleted The checkoff list at the bottom of the page will need to be updated.

ORIENTATION

Textbook/Collection Layout

The approach of this course will be to take the student through a progression of materials that will allow the student to develop the skills of programming. The basic unit of study is a Pressbooks page (chapter). Several pages are collected into a chapter (part).

~~The chapters are divided into five groups.~~

Group Title	Chapters	Modules
Pre-Chapter Items	N/A	4
Foundation Topics	1-5	27
Modular Programming	6-9	17
Structured Programming	10-16	30
Intermediate Topics	17-21	17
Advanced Topics	22-24	11
Review Materials	N/A	5
Appendix	N/A	7
Total Modules	**N/A**	**118**

Some professors using this textbook/collection might decide to eliminate certain modules or chapters. Some may eliminate the entire Advanced Topics group. Other professors may choose to add additional study materials. The advantage of this textbook/collection is that it may be adapted by professors to suit the needs of their students.

Chapter Layout

Each chapter will usually flow from:

1. One or more modules built for independent delivery.
2. A Practice module built specifically for this textbook/collection.

As you proceed with the modules that comprise a chapter, you should:

- Complete any tasks/demos that require downloading items.
- Do any exercises.
- Create study cards for all definitions. When this material is

used as a textbook for a course the definitions are to be memorized. Confirm this with your professor.

As you start the Practice module you will usually encounter:

- Learning Objectives
- Exercises – In addition to any exercises within the study modules that you completed before the practice module, there will be at least one exercise for students to complete.
- Lab Assignment – Usually, completed on one's own efforts. Review the instructions/restrictions from your professor/teacher if using this for a high school or college credit course.
- Problems – The intent of this activity is for students to formulate their own answers. Thus, solutions to the problems will not be provided. When the materials are used as a textbook for a course, the professor/teacher may assign students to a "Study Group" or let students form study groups to discuss their solutions with each other. If you are using this for a high school or college credit course, verify that you may work as team at solving the problems. This type of approved activity is called "authorized collusion" and is not a violation of "Academic or Scholastic Dishonesty" rules.

A professor using this textbook/collection/course will most likely have additional lab assignments, quizzes and exams that would be used in calculating your grade.

Reading List

The modules in this textbook/collection have had content reviewed and are believed to be sufficient, thus **no additional**

textbook is required. However, some students desire additional references or reading. The author has used several textbooks over the years for teaching "COSC1436 – Programming Fundamentals I" course at Houston Community College and at the Community College of Qatar. A reading reference list has been prepared and includes references for the following textbooks:

1. Starting Out with C++ Early Objects, by: Tony Gaddis et. al., 7th Edition, International Edition, ISBN: 978-0-13-137714-1
2. Starting Out with C++ Early Objects, by: Tony Gaddis et. al., 6th Edition, ISBN: 0-321-51238-3
3. Starting Out with C++ Early Objects, by: Tony Gaddis et. al., 5th Edition, ISBN: 0-321-38348-6
4. Computer Science – A structured Approach using C++, by: Behrouz A. Forouzan et. al., 2nd Edition, ISBN: 0-534-37480-8

These textbooks are typically available in the used textbook market at a reasonable price. You may use any one of the three books. If you acquire one of the above **optional** traditional textbooks, you may want to download and store the following file to your storage device (disk drive or flash drive) in an appropriate folder.

Download from Connexions: Connexions_Module_Reading_List_col10621.pdf

SYLLABUS

The syllabus for a course that is for credit will be provided by your specific course professor. If you are using this textbook/ collection for non-credit as self-study, we have some suggestions:

1. Plan regular study periods
2. Review the three (3) Pre-Chapter Items modules
3. Review the last four (4) modules in the Appendix
4. Proceed with Chapter 1 going through all 24 chapters
5. Do all of the demo programs as you encounter them
6. Memorize all of the terms and definitions
7. Do all lab assignments
8. Prepare answers to all of the problems in the Practice modules
9. At the end of every section, do the Review module

These is no magic way to learn about computer programming other than to immerse yourself into regular study and **study includes more than casual reading**. To help you keep track of your study, we have included a check off list for the textbook/collection.

Check	Description	# Modules
	Pre-Chapter Items	4
	Last four Appendix Items	4
	Chapters 1 to 5	27
	Review Materials for 1 to 5	1
	Chapters 6 to 9	17
	Review Materials for 6 to 9	1
	Chapters 10 to 16	30
	Review Materials for 10 to 16	1
	Chapters 17 to 21	17
	Review Materials for 17 to 21	1
	Chapters 22 to 24	11
	Review Materials for 22 to 24	1
	First three Appendix Items	3
N/A	**Total Modules**	**118**

REFERENCES

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

PART I

INTRODUCTION TO PROGRAMMING

Note to Contributors:

We'll need to decide what to do with each chapter introduction. A chapter overview and learning objectives might be nice. For now, I've just listed the module titles contained in the chapter.

Also, please note Pressbooks terminology. What we might think of as a chapter is called a Part. What we might think of as a module or subchapter item is called a Chapter.

- Systems Development Life Cycle
- Program Design
- Pseudocode
- Flowcharting
- Testing
- Practice

Systems Development Life Cycle

KENNETH LEROY BUSBEE

Note to Contributors:

I'm trying to start each module with an **Overview** paragraph followed by **Discussion**, and then end with **Key Terms** as appropriate. This will provide some consistency across the different modules.

OVERVIEW

The **Systems Development Life Cycle** describes a process for planning, creating, testing, and deploying an information system.[1]

DISCUSSION

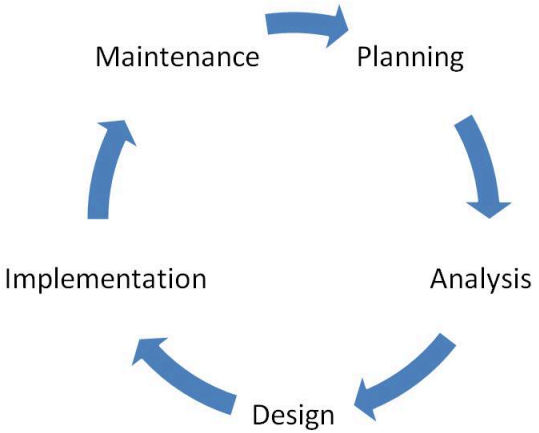
The Systems Development Life Cycle is the big picture of creating an information system that handles a major task (referred to as an application). The **applications** usually consist of many programs. An example would be the Department of Defense supply system, the customer system used at your local

1. Wikipedia: Systems development life cycle

bank, the repair parts inventory system used by car dealerships. There are thousands of applications that use an information system created just to help solve a business problem.

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

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



Systems Development Life Cycle

During the **Design** phase the System Analyst will document the inputs, processing and outputs of each program within the

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

KEY TERMS

applications
An information system or collection of programs that handles a major task.
implementation
The phase of a Systems Development Life Cycle where the programmers would be assigned to write specific programs.
life cycle
Systems Development Life Cycle: Planning – Analysis – Design – Implementation – Maintenance

system analyst
Computer professional in charge of creating applications.

REFERENCES

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

Program Design

KENNETH LEROY BUSBEE

OVERVIEW

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

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

Understanding the Program

If you are working on a project as a 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 it's:

- Inputs
- Processing
- Outputs

This **IPO** approach works very well for beginning programmers. Sometimes, it might help to visualize the programming 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 others 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 applications software that helps create an information system

and/or programs. This type of software is called Computer-aided Software Engineering.

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

Develop Test Data

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 to check the model to see if it produces the correct results.

KEY TERMS

IPO
Inputs – Processing – Outputs
pseudocode
English-like statements used to convey the steps of an algorithm or function.
test data
Providing input values and predicting the outputs.

REFERENCES

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

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 complier exists that will translate the pseudocode to any machine language. Pseudocode is used for documenting the program or module design (also known as the algorithm).

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

Outline using Pseudocode

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

1. Wikipedia: Pseudocode

```
    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 div

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

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

Flowcharts

KENNETH LEROY BUSBEE

OVERVIEW

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

DISCUSSION

Flowcharting Symbols

Terminal

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

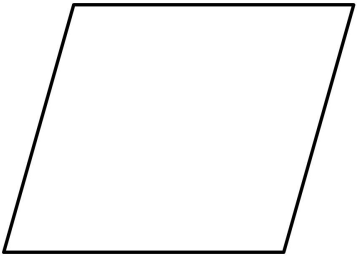
1. Wikipedia: Flowchart

Process

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

Input/Output

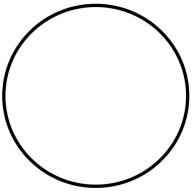
The parallelograms designate input or output operations.



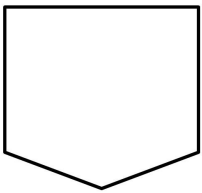
Connectors

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

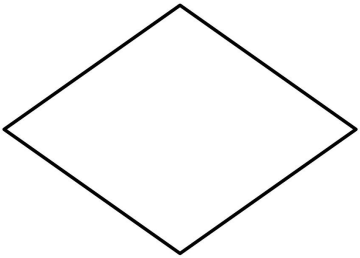


Off-Page Connector



Decision

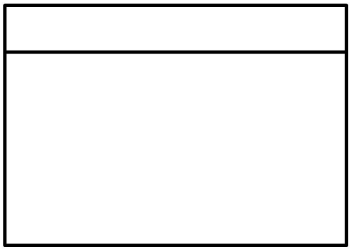
The diamond is used to represent the true/false statement being tested in a decision symbol.



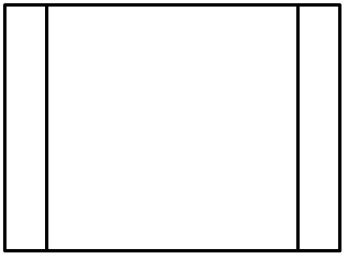
Module Call

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

Local module: usually a program control function.

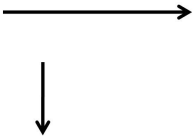


Library module: usually a specific task function.



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.



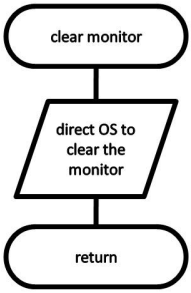
Examples

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

Functions

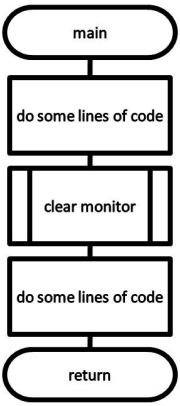
pseudocode: Function with no parameter passing

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



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
Endfunction
```



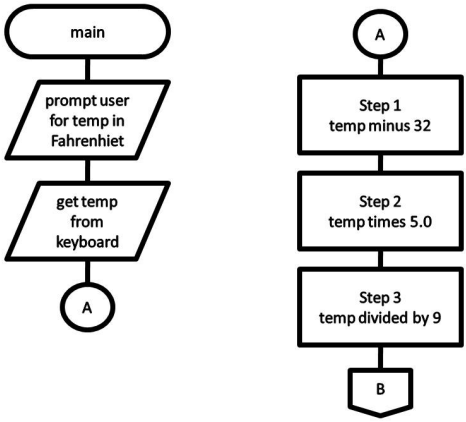
Function main

Sequence Control Structures

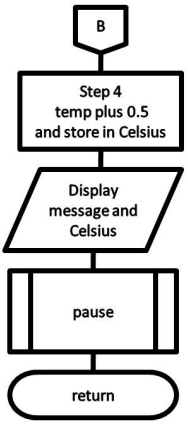
The next item is pseudocode for a simple temperature conversion program. This demonstrates the use of both the on-page and off-page connectors. It also illustrates the sequence control structure where nothing unusually happens. Just do one instruction after another in the sequence listed.
pseudocode: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

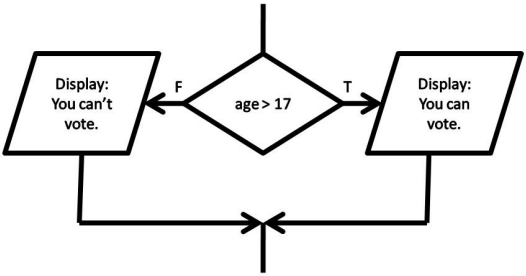


Sequence control structured continued

Selection Control Structures

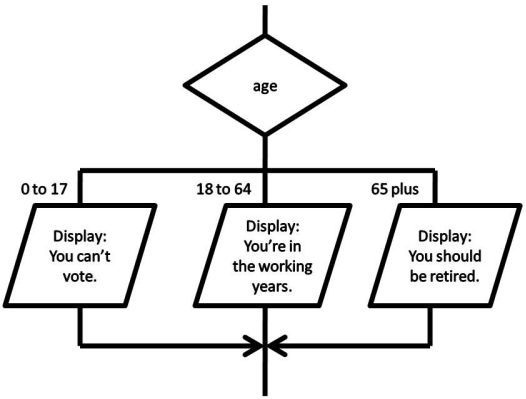
pseudocode: If then Else

```
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 "Your in your working years."
    65 +      Display "You should be retired."
Endcase
```

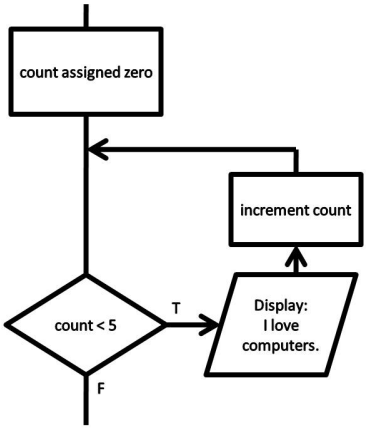


Case control structure

Iteration (Repetition) Control Structures

pseudocode: While

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

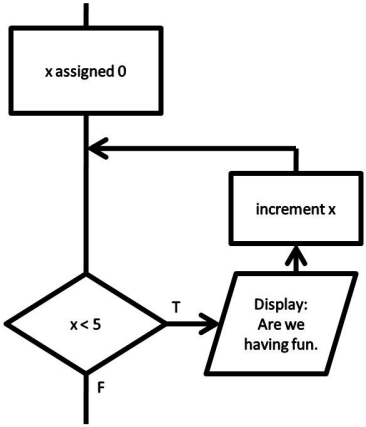


While control structure
pseudocode: For

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

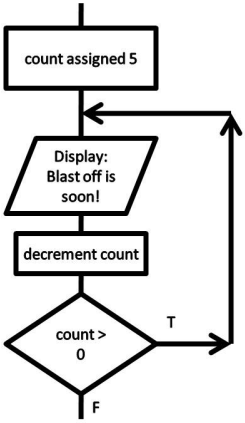
The for loop does not have a standard flowcharting method and you will find it done in different ways. The for loop as a counting

loop can be flowcharted similar to the while loop as a counting loop.



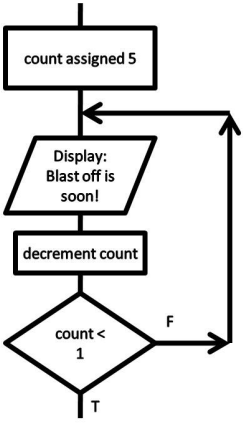
For control structure
pseudocode: Do While

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



Do While control structure
pseudocode: Repeat Until

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



Repeat Until control structure

KEY TERMS

decision symbol
A diamond used in flowcharting for asking a question and making a decision.
flow lines
Lines (sometimes with arrows) that connect the various flowcharting symbols.

flowcharting
A programming design tool that uses graphical elements to visually depict the flow of logic within a function.
input/output symbol
A parallelogram used in flowcharting for input/output interactions.

process symbol
A rectangle used in flowcharting for normal processes such as assignment.

REFERENCES

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

Software Testing

KENNETH LEROY BUSBEE

Note for Contributors:

Is the example here more complicated than necessary to introduce software testing? I personally might go with an example with a single input, such as a temperature for temperature conversion.

Also, do we want to get into the idea of unit testing vs. system testing, etc., or is that beyond the scope or current introductory point in the course?

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

1. Wikipedia: Software testing

- can be installed and run in its intended environments
- achieves the general result its stakeholders desire

DISCUSSION

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

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

Test data is developed by using the algorithm of the program. This algorithm is usually documented during the program design with either flowcharting or pseudocode. Here is the pseudocode in outline form describing the inputs, processing and outputs for a program used for painting rectangular buildings.
Pseudocode using an IPO Outline for Painting a Rectangular Building

```
Input
    display a message asking user for the length of the buildin
    get the length from the keyboard
    display a message asking user for the width of the building
    get the width from the keyboard
    display a message asking user for the height of the buildin
    get the height from the keyboard
    display a message asking user for the price per gallon of pa
    get the price per gallon of paint from the keyboard
    display a message asking user for the sq ft coverage of a ga
    get the sq ft coverage of a gallon of paint from the keyboar
```

```
Processing
    calculate the total area of the building by:
        multiplying the length by height by 2
        then multiply the width by height by 2
        then add the two results together
    calculate the number of gallons of paint needed by:
        dividing the total area by the coverage per gallon
        then round up to the next whole gallon
    calculate the total cost of the paint by:
        multiplying the total gallons needed by the price of one

Output
    display the number of gallons needed on the monitor
    display the total cost of the paint 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: My building is 100 feet long by 40 feet wide and 10 feet in height and I selected paint costing $28.49 per gallon that will cover 250 square feet per gallon. We should verify that the pseudocode is prompting the user for this data.

Processing: Using my solar powered hand held calculator, I can calculate (or predict) the total area would be: (100 x 10 x 2 plus 40 x 10 x 2) or 2,800 sq ft. The total gallons of paint would be: (2800 / 250) or 11.2 gallons. But rounded up, I would need twelve

(12) gallons of paint. The total cost would be: (28.49 times 12) or $341.88. We should verify that the pseudocode is performing the correct calculations.

Output: Only the significant information (number of gallons to buy and the total cost) are 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 (in our case C++)? 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 other item) could be wrong
2. The conversion of the plan to code might be wrong
3. The test data results were calculated wrong

Resolving problems of this nature can be the most difficult

problems a programmer encounters. You must review each of the above to determine where the error is lies. Fix the error and re-test your program.

KEY TERMS

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

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

REFERENCES

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

Practice

KENNETH LEROY BUSBEE AND DAVE BRAUNSCHWEIG

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Create a pseudocode document for a programming problem.
3. Create a flowchart for a programming problem.
4. Perform software testing for a programming problem.

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. Coding the program in a language like C++ is the first task of planning. You plan as you code.
2. Pseudocode is the only commonly used planning tool.
3. Test data is developed for testing the program once it is code into a language like C++.
4. The word pseudo means false and includes the concepts of fake or imitation.
5. Many programmers pick up the bad habit of not completing the planning step before starting to code the program.

Answers:

1. false
2. false

3. false
4. true
5. true

ACTIVITIES

The following activities focus on software planning and testing using pseudocode and / or flowcharts. Programming language code will be written in future assignments.

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 prepared meal. Test your program by having someone else follow the instructions and guide them as they use your program. Test your pseudocode by reading the pseudocode out loud as someone else follows your instructions.
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 prepared meal. Test your flowchart by reading the flowchart out loud while someone else follows your instructions.
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 he purchased. The quantity (Example: 2 cases) and the price he 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.

REFERENCES

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

PART II

INTEGRATED DEVELOPMENT ENVIRONMENT

- Integrated Development Environment
- Standard Input and Output
- Compiler Directives
- Practice

Integrated Development Environment

KENNETH LEROY BUSBEE

Note to Contributors:

The idea for revising this chapter / part is to first focus on free online IDEs so students can start coding without having to deal with platform installation issues.

We also need to address interpreted languages vs. focusing exclusively on compiled languages.

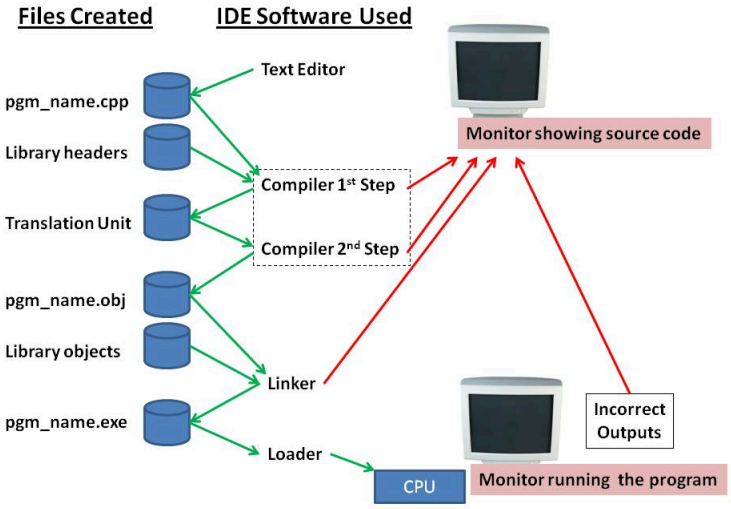
“Hello world!” examples will be added in several languages to provide starting code for readers to test.

IDE Overview

High-level language programs are usually written (coded) as ASCII text into a source code file. A unique file extension (Examples: .asm .cob .for .pas .c .cpp) is used to identify it as a source code file. As you can guess for our examples – Assembly, COBOL, FORTRAN, Pascal, “C” and “C++” 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 micro 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 usually called a compiler. However, this one compiler program is really many software items that create an **environment** used by programmers to **develop** software. Thus the name: Integrated Development Environment or IDE.

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.



Integrated Development Environment or IDE

Upon starting the IDE software the programmer usually indicates he wants to open a file 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 encounted (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 his test data, he might discover that the outputs are not correct. These types of errors are called logic errors and would require him to return to the source code to change the algorithm.

Resolving Errors

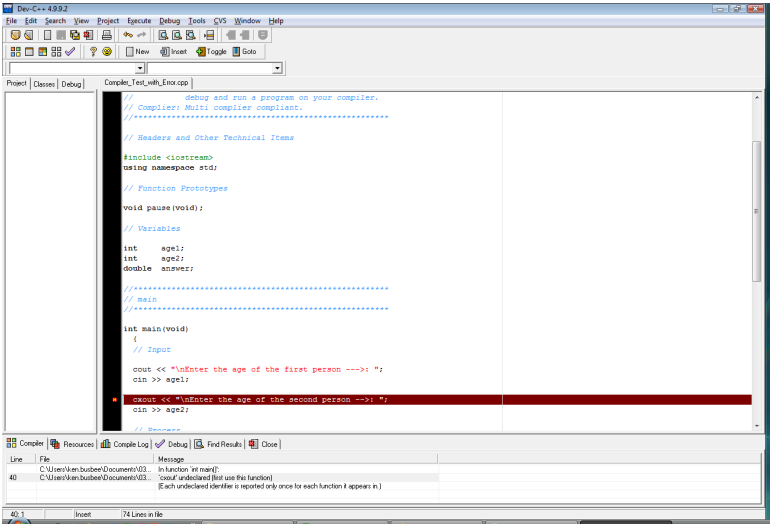
Despite our best efforts at becoming perfect programmers, we will create errors. Solving these errors is known as **debugging** your program. The three types of errors in the order that they occur are:

1. Compiler
2. Linker
3. Logic

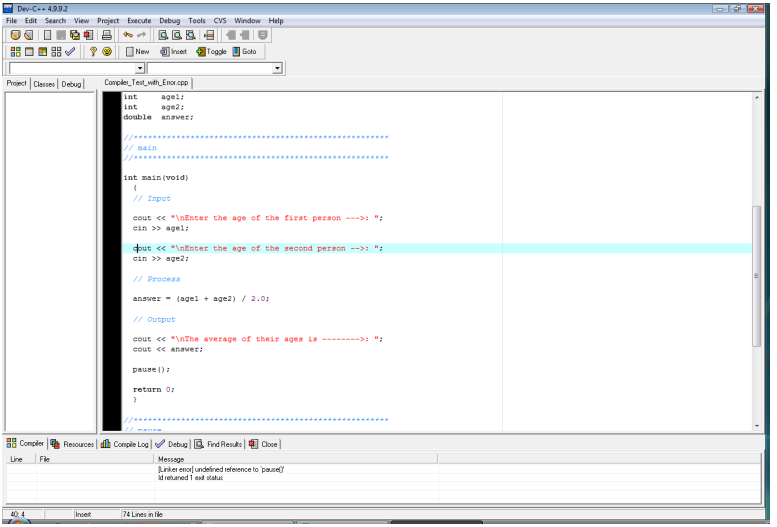
There are two types of compiler errors; pre-processor (1st step) and conversion (2nd step). A review of Figure 1 above shows the four arrows returning to the source code so that the programmer can correct the mistake.

During the conversion (2nd step) the complier 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.

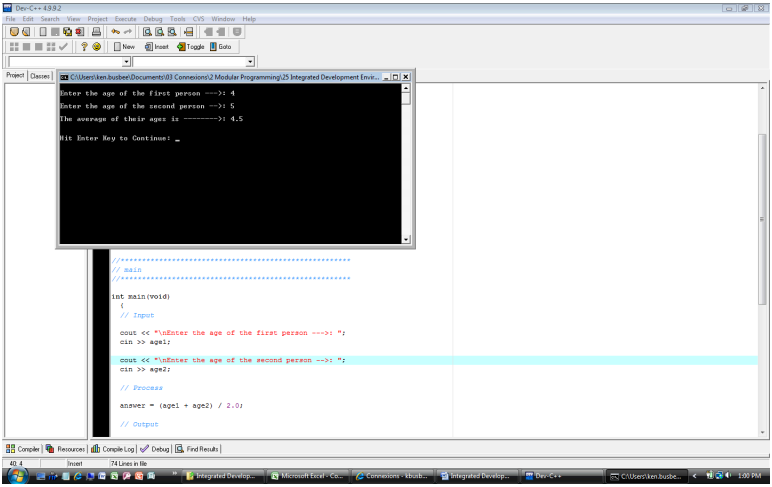
The next three figures show IDE monitor interaction for the **Bloodshed Dev-C++ 5 compiler/IDE**.



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



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



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

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device

in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the soruce code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Pre_Processor_Compiler_Errors.cpp

Download from Connexions: Demo Compiler_Conversion_Errors.cpp

Download from Connexions: Demo_Linker_Errors.cpp

Download from Connexions: Demo_Logic_Errors.cpp

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 direct the CPU to start running the program.
pre-processor
The first step the compiler does in converting source code to object code.

text editor
A software program for creating and editing ASCII text files.

warning
A compiler alert that there might be a problem.

REFERENCES

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

Standard Input and Output

KENNETH LEROY BUSBEE

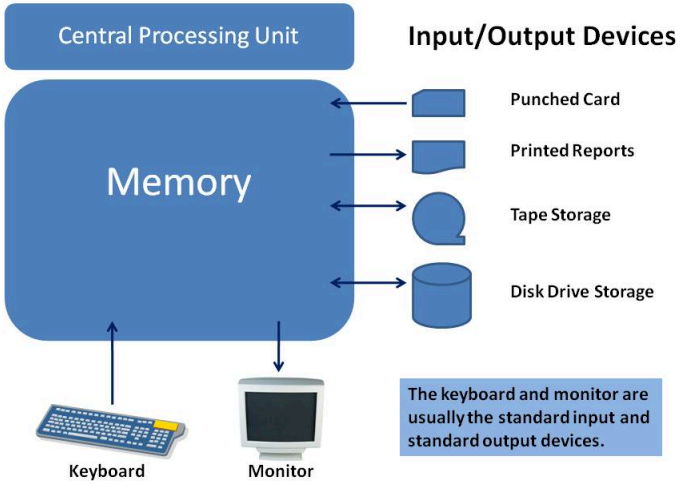
OVERVIEW

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]

GENERAL 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 located in the 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.

STANDARD I/O WITHIN C++

The developers of the C++ programming language decided to provide some of the more technical code needed to interact with the operating system and the I/O devices. In the following example the include directive inserts a file that contains code from the Input-Output Stream library. This file contains necessary code to use cout and cin for sending data to the monitor or getting data from the keyboard.

```
#include <iostream>
```

You should think of **cout** and **cin** as being locations that you can send to or receive data from; similar in concept to any other variable storage location within the data area of our program. The C++ programming language has two operators to use in conjunction with I/O devices.

Action	C ++ operator symbol	Used with
insertion operator (write)	<< (a pair of less than symbols)	cout
extraction operator (read)	>> (a pair of greater than symbols)	cin

Consider the following code:
Insertion and Extraction

```
int age1;     // variable set up
   then later on in our program
cout << "\nEnter the age of the first person --->: ";
cin >> age1;
```

Using the **cout** the programmer displays (or inserts) a prompting message on the monitor for the user to see. Using the **cin** the user types an integer value and hits the enter key and the computer extracts the value from the keyboard and stores it into the variable named age1. Within the computer all data are stored as numbers and thus part of the technical code provided by the developers of the C++ programming language that is within the **Input-Output Stream** library converts data from numbers to those symbols we are used to seeing as humans and vice versa. Example: If the user entered the numeral digits 57 and hit the enter key – the extraction operator would convert the 57 into a binary number and move the binary number into the integer storage place named age1.

The **cout** which uses the standard output device does not format the output into a Graphical User Interface (GUI) where you have a mouse to use. A modern operating system using GUI normally opens a black screen output box that would be similar to how the monitor was used when first developed in the 1960's. That is the default of how **cout** is normally implemented by most compilers.

The output message has a unique item worth mentioning. At the very front of the message is a backslash followed by the letter n. They do not get printed on the monitor. It is a special code (called a printer **escape code**) telling the printer to go to a new line. Printer! I thought we were using a monitor? We are but the code is a left over from the early days of printer output. The backslash tells the printer or monitor that the next letter is a

command. The letter n is used for telling the printer or monitor to go to the front of a new line.

KEY TERMS

device
A piece of equipment that is electronically connected to the memory so that data can be transferred between the memory and the device.
escape code
A code directing an output device to do something.

extraction
Aka reading or getting data from an input device.
insertion
Aka writing or sending data to an output device.

standard input
The keyboard.
standard output
The monitor.

REFERENCES

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

Bloodshed Dev-C++ 5 Compiler/IDE

KENNETH LEROY BUSBEE

INTRODUCTION

Microsoft and Borland are the two reputable names within the programming world for compilers. They sell compiler software for many programming languages. For the C++ programming language, the Microsoft Visual Studio which includes C++ and Borland C++ Builder are excellent compilers. Often with textbooks or free via the internet; you can get Microsoft's Visual C++ Express or Borland's Personal Edition version of a compiler. However, installing either of these compliers can be complex. Microsoft's Visual Studio compiler often creates a variety of installation problems (such as making sure the operating system and .net components are current) thus making it difficult for students to install at home. These compliers require you to build a project to encompass every program. Using a commercially sold compiler that professional programmers would consider using for project development is fine for professionals but often confusing to beginners. Eventually, if you are going to become a professional programmer, you will need to become familiar with the commercially sold compilers.

We suggest that beginning students consider one of the easier to install compiler software packages for use in a programming fundamentals course. The best option we have found is an **open source** compiler/IDE (Integrated Development Environment) named: Bloodshed Dev-C++ 5 compiler/IDE.

open source
Group development of source code for software that is made available to the public at no cost.

BLOODSHED DEV-C++ 5 COMPILER/IDE

Advantages: Can be installed on Windows 95/98/NT/2000/XP operating systems. I have it installed on Windows Vista operating system, thus it can work with slower processors and almost any Windows operating system. It only requires about 80 MB of storage space (usually enough for the compiler with all of its files and storage room for several of your programs). It is very easy to install and easy to use. Does not require the use of a "project"; thus individual source code files can be easily compiled.

Disadvantages: Would not normally be used by professional programmers, but is sufficient for a beginning computer programming course and is a full-featured compiler/IDE.

Unique Advantage: Can be installed and run on a flash drive, thus giving the student the ability to work on their lab assignments on any computer that has a USB port. This can give the student **portability**, being able to do lab assignments at home, work, library, open lab, classroom, friend's house, etc.

portability
The ability to transport software on a flash drive and thus use it on various machines.

PREPARATION BEFORE INSTALLATION

Creating the Needed Folders and Sub-Folders

You need to get the **software** and a C++ **source code** program that has been tested and is error free. You will need about 80MB

of storage space. We suggest that you create **two folders** on your hard drive or flash drive depending on which installation you choose. If on a flash drive create them at the root level of the drive. If on your home machine, you can use the folder area set up by the operating system for you as a user. Name them:

- Cpp_Software_Download
- Cpp_Source_Code_Files

Within the Cpp_Source_Code_Files folder, create a sub-folder named:

- Compiler_Test

To help you keep files organized, you will want to create other sub-folders for storing source code files. We suggest you create at least two other sub-folder to be used with Connexions' related modules. Within the Cpp_Source_Code_Files, create sub-folders named:

- Demo_Programs
- Monitor_Header

folder
A named area for storage of documents or other files on a disk drive or flash drive.
source code
Any collection of statements or declarations written in some human-readable computer programming language.

Getting the Software

The full version of the software is named: Dev-C++ 5.0 beta 9.2 (4.9.9.2) (9.0 MB) with Mingw/GCC 3.4.2 You can either download

it from Bloodshed or download the version as of 12/8/2008 that is stored on the Connexions web site. Store it in the Cpp_Software_Download folder you created. The software is approximately 9.1 MB and will take several minutes to download if you are using a dial-up modem connection.
The software has not significantly changed since 2007 and the Connexions version will be sufficient for most users. The Bloodshed link requires some additional navigation to get to the software download. Thus, because it is significantly easier, we recommend that you download the software from the Connections web site.

Link to Bloodshed: http://www.bloodshed.net/dev/devcpp.html
Download from Connexions: devcpp-4.9.9.2_setup.exe

Getting a C++ Source Code File

Listed below is a C++ source code file titled: Compiler_Test.cpp It has been prepared for Connexions web delivery. Download and store it in the Compiler_Test sub-folder you created. You may need to right click on the link and select "Save Target As" in order to download the file.
Download from Connexions: Compiler_Test.cpp

INSTALLATION INSTRUCTIONS FOR BLOODSHED DEV-C++ 5 COMPILER/IDE

The Version 5 which is well tested (don't let the beta release scare you) and should work on a variety of machines and various Microsoft Operating systems including Windows 98, Windows 2000, Windows XP and Windows Vista. Below are installation instructions for installing it on a machine or installing it on a flash drive. We don't suggest trying to switch between the

machine drive and flash drive. If it is installed on the machine drive and you try installing it on a flash drive, it creates problems and will not work perperly. Either install it on the flash drive to gain your portability or install it on your machine.

Computer Installation Instructions

1. Navigate to the Cpp_Software_Download folder and run the devcpp-4.9.9.2_setup.exe software by double cliking on the filename.
2. Use common sense and answer the installation prompts. NOTE THE FOLLOWING TWO ITEMS:
3. When it gets to the “Choose Install Location” use the default software location of: C:\Dev-Cpp\ (or select the location you want to store the installed program but use the default unless you are familiar with installing software).
4. When it asks: “Do you want to install Dev C++ for all users on this computer?” answer “Yes”.
5. After it installs, it will ask some “first time configuration” questions. Again, use common sense and answer the questions. NOTE THE FOLLOWING ITEM:
6. Answer “No” to the retrieve information from header files.
7. It will start your compiler/IDE with a “Tip of the day”. We suggest you check the box in the lower left and select “Close”.
8. Close your compiler/IDE by using the normal red “X” box. We want to show you how to start your compiller normally.
9. You start your compiler software similar to starting any software loaded on your machine (“Start” then “All Programs” then “Bloodshed Dev-C++” then “Dev-C++”).
10. On the menus at the top – Select “File” then “Open project or file” then navigate to where your source code file (Compiler_Test.cpp) is stored on your hard drive. See the

suggested folder name above. Select the source code file and open it.

11. You should see the source code listing. Press F9 key or select the “Execute” then “Compile & Run” from the menus at the top. A black screen box should appear and you answer questions appropriately to run the program. When you are done running your program the black screen box goes away.

Flash Drive Installation Instructions

1. Navigate to the Cpp_Software_Download folder and run the devcpp-4.9.9.2_setup.exe software by double cliking on the filename.
2. Use common sense and answer the installation prompts. NOTE THE FOLLOWING TWO ITEMS:
3. When it gets to the “Choose Install Location” you can see that the default software location of: C:\Dev-Cpp\ however, it needs to be changed. Change the “Destination Directory” by selecting changing the default software location from: C:\Dev-Cpp\ to DriveLetter:\Dev-Cpp\ (where the DriveLetter is the drive that represents your flash drive).
4. When it asks: “Do you want to install Dev C++ for all users on this computer?” answer “No”.
5. After it installs, it will ask some “first time configuration” questions. Again, use common sense and answer the questions. NOTE THE FOLLOWING ITEM:
6. Answer “No” to the retrieve information from header files.
7. It will start your compiler/IDE with a “Tip of the day”. We suggest you check the box in the lower left and select “Close”.
8. Close your compiler/IDE by using the normal red “X” box. We want to show you how to start your compiller normally.

9. To start your compiler software you navigate to the “Dev-Cpp” folder on your flash drive and select the “devcpp.exe” application. NOTE: When using the flash drive you should not try starting the compiler by double clicking on a C++ source code file. This method works on a machine installation but does not work on a flash drive installation.
10. On the menus at the top – Select “File” then “Open project or file” then navigate to where your source code file (Compiler_Test.cpp) is stored on your flash drive. See the suggested folder name above. Select the source code file and open it.
11. You should see the source code listing. Press F9 key or select the “Execute” then “Compile & Run” from the menus at the top. A black screen box should appear and you answer questions appropriately to run the program. When you are done running your program the black screen box goes away.

REFERENCES

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

Compiler Directives

KENNETH LEROY BUSBEE

General Discussion

A **compiler directive** is an instruction to the compiler to complete a task before formally starting to compile the program, thus they are sometimes called pre-processor directives. Among other items, during the pre-processor step the compiler is looking for compiler directives and processes them as they are encountered. After completing the tasks as directed, the compiler proceeds to its second step where it checks for syntax errors (violations of the rules of the language) and converts the source code into an object code that contains machine language instructions, a data area, and a list of items to be resolved when he object file is linked to other object files.

Within C++ the pound symbol or # as the first character of a line indicates that the next word is a directive (or command word) to be evaluated. The two most common compiler directives are:

1. **include** – with the item following include being the name of a file that is to be inserted at that place in the file. The files are often called “Header Files” because the include directive is normally inserted toward the top of the file (at the head) as one of the first items.
2. **define** – with the item followed by an identifier name and a value. This identifier name and value is stored by the compiler and when it encounters the identifier name in the program it substitutes the value for the identifier name.

In the following example the include directive is inserting a file that contains code from the Input-Output Stream library. This file contains necessary code to use cout and cin for sending data to the monitor or getting data from the keyboard.

```
#include <iostream>
```

In the next example the define directive is being used to handle a constant (called a defined constant).
Subtituting PI

```
#define PI 3.14159
....Later on in the program when it encounters PI
....it will replace or substitute PI with the value 3.14159
....For example:
area_circle = radius * radius * PI;
    would become:
area_circle = radius * radius * 3.14159;
```

Of note, compiler directives in C++ do not have a semi-colon after them. Within C++ programming instructions or statements end with a semi-colon, but not compiler directives.

KEY TERMS

compiler directive
An instruction to the compiler to complete a task before formally starting to compile the program.
include
A compiler directive to insert the contents of a file into the program.

REFERENCES

- cnx.org: Programming Fundamentals – A Modular

Practice

KENNETH LEROY BUSBEE

Note to Contributors:

This practice page came from the original Chapter 1 and needs to be merged with the other practice page consistent with the revised content.

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Install the Bloodshed Dev-C++ 5 compiler
3. Make minor modifications to an existing program

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. Beginning programmers participate in all phases of the Systems Development Life Cycle.
2. The Bloodshed Dev-C++ 5 compiler/IDE is the preferred compiler for this textbook/collection, however any C++ compiler will work.
3. Most compilers can be installed on a flash drive.
4. In addition to function as the name of a sub-program, the computer industry also uses macro, procedure and module.

5. Generally functions fall into two categories: Program Control and Specific Task.

Answers:

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

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 01 Files

Within the Chapter 1 Connexions modules you were given directions on how to install the **Bloodshed Dev-C++ 5 compiler/ IDE**and to test your installation with the Compiler_Test.cpp source code file. If you have not done this, return to the Connexions materials and complete this task.

In the compiler installation directions you were asked to make a folder named: Cpp_Source_Code_Files. All of your lab assignments in this course assume you have that folder on the same drive as your compiler (either drive C: your hard disk drive, or on a flash drive). If you don't have that folder, go create it now.

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_01 within the folder named: Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Compiler_Test.cpp

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Navigate to your sub-folder: Chapter_01 and rename the Compiler_Test.cpp source code file to: **Lab_01.cpp**
- If you are having problems seeing the file extensions, visit the "Show Hide File Extensions" instructions within the Appendix.
- Start your compiler and open the source code file. Carefully make the following modifications:

- **Change the comments at the top, specifically:**

- The filename should be: Lab_01.cpp
- Purpose should be: Average the weight of three people
- Remove the next 2 lines of comment talking about the main idea
- Author: put your name and erase my name and copyright
- Date: Put today's date
- Remove the next 3 lines of comment dealing with licensing (don't erase the asterisk line)

During the rest of the course you will often use a source code file provided by the instructor as your starting point for a new lab assignment. Sometimes you will use a source code file that you have created as your starting point for a new lab assignment. Either way, you should modify the comments area as appropriate to include at a minimum the four lines of information (filename, purpose, author and date) as established in this lab assignment.

- **We are now going to make simple modifications to this program so that it is able to average the weight of three people. Do the following:**

- Within the variables area, change the variable names for age1 and age2 to weight1 and weight2. Add another variable of integer data type with the identifier name of weight3.
- The input area has two prompts and requests for data from the user. They are paired up – a prompt and getting data from the keyboard. We need to modify the prompt to ask for weight instead of age. We need to change the variable name from age1 to weight1. Do this for the second pair that prompts and gets the second data item. Create a third pair that prompts and gets the third data item.
- The process area has only one line of code and we need to make changes that add the weight3 and divides by 3.0 instead of 2.0. The code should look like this:
- **answer = (weight1 + weight2 + weight3) / 3.0;**
- The output area needs the text modified from ages to weights.

- Build (compile and run) your program. You have successfully written this program if when it run and you put

in the three weights; it tells you the correct average.

- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

PROBLEMS

Problem 01a – Instructions

List the steps of the Systems Development Life Cycle and indicate which step you are likely to work in as a new computer professional.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Be able to list the categories and give examples of errors encountered when using an Integrated Development Environment (IDE).
3. Write the C++ code for a program using appropriate planning documentation that you or another has designed.

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. IDE means Integer Division Expression.
2. Most modern compilers are really an IDE type of software, not just a compiler.
3. cin and cout are used for the standard input and output in C++.
4. Programming errors are extremely easy to understand and fix.
5. All C++ programs will have at least one include type of compiler directive.

Answers:

1. false
2. true

3. true
4. false
5. true

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 05 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_05 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Solution_Lab_02_Pseudocode.txt

Download from Connexions: Solution_Lab_02_Test_Data.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Copy into your sub-folder: Chapter_05 one of the source code listings that we have used. We suggest the Lab 01 source code and rename the copy: **Lab_05.cpp**
- Modify the code to follow the Solution_Lab_02_Pseudocode.txt file.

- Build (compile and run) your program. You have successfully written this program if when it runs and you use the test data [use the test data as supplied as the solution for Lab 02] it gives the predicted results.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 05a – Instructions

List and describe what might cause the four (4) types of errors encountered in a program using an Integrated Development Environment software product.

Problem 05b – Instructions

Identify four (4) problems with this code listing (HINT: The four (4) types of errors encountered in a program using an Integrated Development Environment software product).
C++ Source Code Listing

```
//*******************************************************
// Filename: Compiler_Test.cpp
// Purpose:  Average the ages of two people
// Author:   Ken Busbee; © Kenneth Leroy Busbee
```

```
// Date:     Jan 5, 2009
// Comment:  Main idea is to be able to
//           debug and run a program on your compiler.
//*******************************************************

// Headers and Other Technical Items

#include <iostrern>
using namespace std;

// Function Prototypes

void pause(void);

// Variables

int     age1;
int     age2;
double  answear;

//*******************************************************
// main
//*******************************************************

int main(void)
  {
  // Input
  cout << "\nEnter the age of the first person --->: ";
  cin >> age1;
  cout << "\nEnter the age of the second person -->: ";
  cin >> age2;

  // Process
```

```
  answer = (age1 + age2) / 3.0;

  // Output
  cout << "\nThe average of their ages is --------->: ";
  cout << answer;

  pause();
  return 0;
  }

//*******************************************************
// End of Program
//*******************************************************
```

REFERENCES

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

PART III

DATA & OPERATORS

- Data Types in C++
- Identifier Names
- Constants and Variables
- Data Manipulation
- Assignment Operator
- Arithmetic Operator
- Data Type Conversions
- Practice

Data Types

KENNETH LEROY BUSBEE

Note to Contributors:

2nd Edition revision progress ends here. This page needs to be cleaned up so that the data type discussion is more generic and applicable to all programming languages / environments.

General Discussion

Our interactions (inputs and outputs) of 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: yourname which is a string of characters; your age which is usually an integer; or the amount of money in your pocket which is usually a value measured in dollars and cents (something with a fractional part). A major part of understanding how to design and code programs in centered in understanding the types of data that we want to manipulate and how to manipulate that data.

"A **type** defines a set of values and a set of operations that can be applied on those values. The set of values for each type is

known as the domain for that type."[1] The four major families of data include:

- Nothing
- Integer
- Floating-point
- Complex

The C++ programming language identifies five data types as standard data types:

- Void
- Boolean
- Character
- Integer
- Floating-point

The standard data types and the complex data types within C++ have a series of attributes, which include:

- C++ Reserved or Key Word
- Domain – the allowed values
- Signage – do they allow negative numbers or only positive numbers
- Meaning – i.e. What do they represent
- Rules of Definition – What special characters indicate the data type
- Size – in terms of the number of bytes of storage used in the memory
- Operations Allowed – i.e. Which operators can I use on the data type

Placing some of the above into a summary table, we get:

Family	Data Type	Reserved Word	Represents	Standard Type
Nothing	Null or nothing	void	No data	Yes
Integer	Boolean	bool	Logical true and false	Yes
Integer	Character	char	Single characters	Yes
Integer	Integer	int	Whole numbers	Yes
Floating Point	Floating Point	float	Fractional numbers	Yes
Complex	String	string	A sequence (sting them along) of characters	No
Complex	Array	N/A	A collection of elements of the same data type	No
Complex	Pointer	N/A	A value that points to a location (an address) within the data area	No

The five standard data types usually exist in most programming languages and act or behave similarly from language to language. Most courses of study for a programming course or language will explain the standard data types first. After they are learned, the complex data types are introduced.

The Boolean, character and integer data types are identified as belonging to the Integer Family. These data types are all represented by integer numbers and thus act or behave similarly.

KEY TERMS

data type
Defines a set of values and a set of operations that can be applied on those values.
data type families
1) Nothing 2) Integer 3) Floating-Point 4) Complex

domain
The set of allowed values for a data type.
floating point
A data type representing numbers with fractional parts.

integer
A data type representing whole numbers.

Footnotes

- 1 Behrouz A. Forouzan and Richard F. Gilberg, Computer Science A Structured Approach using C++ Second Edition (United States of America: Thompson – Brooks/Cole, 2004) 33.

REFERENCES

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

Identifier Names

KENNETH LEROY BUSBEE

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

Technical to Language

- Use only allowable characters (for C++ the first character 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 these three technical rules.

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 compiler error. To avoid the problem altogether, we teach students to **be case consistent**. Use an identifier name only once and spell it (upper and lower case) the same way within your program.

Industry Rules

- Do not start with underscore (used for technical programming)
- variables in all lower case
- CONSTANTS IN ALL UPPER CASE

These rules are decided by the industry (those who are using the programming language). The above rules were commonly used

within the “C” programming language and have to large degree carried over to C++.

KEY TERMS

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

REFERENCES

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

Constants and Variables

KENNETH LEROY BUSBEE

Understanding Constants

Various textbooks describe constants using different terminology. Added to the complexity are the explanations from various industry professionals will vary greatly. Let's see if we can clear it up.

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

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

Constants are used in three ways within C++. They are:

1. literal constant
2. defined constant
3. memory 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:
Literal Constants

```
int   age   = 21;
char  grade = 'A';
float money = 12.34;
```

```
bool  rich  = false;

cout << "\nStudents love computers";
age = 57;
```

Additionally, we have learned how to recognize the data types of literal constants. Single quotes for char, double quotes for string, number without a decimal point for integer, number with a decimal point belongs to the floating-point family, and Boolean can use the reserved words of true or false.

In addition to literal constants, most text books refer to either symbolic constants or named constants but these two refer to the same concept. A symbolic constant is represented by a name similar to how we name variables. Let's say it backwards; the identifier name is the symbol that represents the data item. Within C++ identifier names have some rules. One of the rules says those names should be meaningful. Another rule about using ALL CAPS FOR CONSTANTS is an industry rule. There are two ways to create symbolic or named constants:

```
#define PI 3.14159
```

Called a defined constant because it uses a textual substitution method controlled by the compiler pre-processor command word "define".

```
const double PI = 3.14159;
```

The second one is called sometimes called **constant variable** but that name is contradictory all by itself. How can it be constant and vary at the same time? The better name for the second one is a memory constant because they have a "specific storage location in memory".

Defining Constants and Variables

In the above examples we see how to define both variables

and constants along with giving them an initial value. Memory constants must be assigned a value when they are defined. But variables do not have to be assigned initial values.

```
int height;
float value_coins;
```

Variables once defined may be assigned a value within the instructions of the program.

```
height = 72;
value_coins = 2 * 0.25 + 3 * 0.05;
```

KEY TERMS

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

REFERENCES

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

Data Manipulation

KENNETH LEROY BUSBEE

Introduction

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.

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.

An Expression Example with Evaluation

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

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

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

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

Precedence of Operators 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. The precedence rules 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 a Precedence of Operators 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 – that is only have one operand
2. Binary – that is have two operands, one on each side of the operator
3. Trinary – which has two operator symbols that separate three operands

Most operators are binary, that is they require two operands. Within C++ there is only one trinary operator, the conditional. All of the unary operators are on the left side of the operand, except postfix increment and postfix decrement. Some precedence charts indicate of which operators are unary and trinary and thus all others are binary.

Assignment Operator

KENNETH LEROY BUSBEE

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 C++ programming language the symbol used 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

```
int age;     // variable set up
    then later in the program
age = 21;
```

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

```
int total_cousins;     // variable set up
    then later in the program
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 assigned to the variable named: total_cousins.

Assignment with Identifier Names in the Expression

```
int students_period_1 = 25;     // variable set up with initial
int students_period_2 = 19;
int total_students;
    then later in the program
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

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

Arithmetic Operators

KENNETH LEROY BUSBEE

General Discussion

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

Action	C ++ operator symbol
Addition	+
Subtraction	–
Multiplication	*
Division	/
Modulus (associated with integers)	%

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

```
age + 1
```

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

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

Data Type Conversions

KENNETH LEROY BUSBEE

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 the cast operator

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

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 when ever 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

```
int money;      // variable set up
    then later in the program
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 sub set 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 sub set of integer values and integer values are a sub set of floating-point values; and within the floating-point values: float values are a sub set 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**. Within C++ 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.

Explicit Demotion with Truncation

```
(int) 4.234
```

This expression would evaluate to: 4.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the soruce code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Data_Type_Conversions.cpp

KEY TERMS

demotion
Going from a larger domain to a smaller domain.

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

implicit
A value that has its data type changed automatically.

promotion
Going from a smaller domain to a larger domain.
truncation
The fractional part of a floating-point data type that is dropped when converted to an integer.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Understand basic data types and how operators manipulate data.
3. Given pseudocode and test data documents, write the C++ code for a program

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. A data type defines a set of values and the set of operations that can be applied on 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.

Answers:

1. true
2. false

3. true
4. true
5. false – Parentheses change the order of evaluation in an expression.

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 03 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_03 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Solution_Lab_03_Pseudocode.txt
Download from Connexions: Solution_Lab_03_Test_Data.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Navigate to your sub-folder: Chapter_03 and open and study the two files.
- We have learned that a fundamental concept of interaction with computers is to divide the problem/task into three parts – input, processing and output. This problem is simple and we will use the IPO (input – processing – output) approach again. **However this time we are going to think about it backwards.**

- **What output do I want displayed?**
- Number of gallons of paint
- Total cost of the paint

- **Thus, what calculations do I need to make?**
- Total cost of the paint is the Number of gallons needed times price per gallon
- Number of gallons needed is the Total area to be covered (let's use square feet) divided by the coverage per gallon of paint (Note: you must round up to the next full gallon of paint.)
- Total area to be covered is the Length times height times 2 added to the width times height times 2

- **Which leads us to, what data do I need as input?**
- Price of a gallon of paint
- Number of square feet that a gallon will cover
- Length of the house
- Width of the house
- Height of the house

- You can see that by working the logic backwards, we can start to completely see what the program must do. We need to enter some data (input), do some calculations (process) and display the results (output).

- Copy into your sub-folder: Chapter_03 one of the source code listings that we have used (we suggest the Lab 01 source code) and rename the copy to: **Lab_03.cpp**
- Modify the code to follow the Solution_Lab_03_Pseudocode.txt file.
- I am just going to give you the line of code for rounding up to the next whole gallon of paint (See the "Data Type Conversions" module within Chapter 3 of the Connexions materials. Do you understand why it works?).
- **total_gal_paint = total_area / coverage_gal_paint + 0.9999;**
- Build (compile and run) your program. You have successfully written this program when it runs with your test data and gives the predicted results.

- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 03a – Instructions

Write the C++ code to do the following pseudocode example.
pseudocode

```
Prompt the user for his monthly income.
Get the users monthly income.
Multiply the monthly income by 12.
Display the annual income.
Pause the program so the user can see the answer. (HINT: You may
```

REFERENCES

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

PART IV

OFTEN USED DATA TYPES

- Integer Data Type
- Floating-Point Data Type
- String Data Type
- Arithmetic Assignment Operators
- Lvalue and Rvalue
- Integer Division and Modulus
- Practice

Integer Data Type

KENNETH LEROY BUSBEE

General Discussion

The integer data type has two meanings:

- The integer data type with its various modifiers that create different domains
- The integer family which also includes the Boolean and character data types

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 twofold. 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, 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.

The integer data type has the same attributes and acts or behaves similarly in all programming languages. The most often used integer data type in C++ is the simple integer.

C++ Reserved Word	int
Represent	Whole numbers (no fractional parts)
Size	Usually 4 bytes
Normal Signage	Signed (negative and positive values)
Domain (Values Allowed)	-2,147,483,648 to 2, 147,483,647
C++ syntax rule	Do not start with a 0 (zero)
C++ syntax rule	No decimal point

Within C++ there are various reserved words that can be used to modify the size or signage of an integer. They include: long, short, signed and unsigned. Signed is rarely used because integers are signed by default – you must specify unsigned if you want integers that are only positive. Possible combinations are:

C++ Reserved Word Combination	Signage
short int	signed
unsigned short int	unsigned
int	signed
unsigned int	unsigned
long int	singed
unsigned long int	unsigned

The domain of each of the above data type options varies with the complier being used and the computer. The domains vary because the byte size allocated to the data varies with the compiler and computer. This effect is known as being **machine dependent**. Additionally, there have been some size changes with upgrades to the language. In “C” the int data type was allocated 2 bytes of memory storage on an Intel compatible central processing unit (cpu) machine. In “C++” an int is allocated 4 bytes.

These variations of the integer data type are an annoyance in C++ 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.

KEY TERMS

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

REFERENCES

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

Floating-Point Data Type

KENNETH LEROY BUSBEE

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

The most often used floating-point family data type used in C++ is the **double**. By default, most compilers convert floating-point constants into the double data type for use in calculations. The double data type will store just about any number most beginning programmers will ever encounter.

C++ Reserved Word	double
Represent	Numbers with fractional parts
Size	Usually 8 bytes
Storage	two parts (always treated together)a mantissa and an exponent
Normal Signage	Signed (negative and positive values)
Domain (Values Allowed)	±1.7E-308 to ±1.7E308
C++ syntax rule	the presence of a decimal point means it's floating-point data

Within C++ there are various reserved words that can be used to establish the size in bytes of a floating-point data item. More bytes mean more precision:

C++ Reserved Word	Size
float	4 bytes
double	8 bytes
long double	10 to 12 bytes (varies by machine)

The domain of each of the above data type options varies with the complier being used and the computer. The domains vary because the byte size allocated to the data varies with the compiler and computer. This effect is known as being **machine dependent**.

These variations of the floating-point family of data types are an annoyance in C++ for a beginning programmer. For a beginning programmer it is more important to understand the general attributes of the floating-point family that apply to most programming languages.

KEY TERMS

double
The most often used floating-point family data type used in C++.
mantissa exponent
The two integer parts of a floating-point value.

precision
The effect on the domain of floating-point values given a larger or smaller storage area in bytes.

REFERENCES

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

String Data Type

KENNETH LEROY BUSBEE

General Discussion

Technically, there is no string data type in the C++ programming language. However, the concept of a string data type makes it easy to handle strings of character data. A single character has some limitations. Many data items are not integers or floating-point values. The message **Hi Mom!** is a good example of a string. Thus, the need to handle a series of characters as a single piece of data (in English correctly called a datum).

In the "C" programming language all string 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). Associated with object oriented programming the string class has been added to C++ as a standard part of the programming language. This changed with the implementation with strings being stored as a length controlled item with a maximum length of 255 characters. Included in the C++ string class is the reserved word of **string**as if it were a data type. Some basics about strings include:

C++ Reserved Word	string
Represent	Series of characters (technically an array)
Size	Varies in length
Normal Signage	N/A
Domain (Values Allowed)	Extended ASCII Character Code Set
C++ syntax rule	Double quote marks for constants

For now, we will address only the use of strings as constants. Most modern compliers 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 the monitor. A typical line of C++ code:

```
cout << "Hi Mom!";
```

would have the “Hi Mom” colored (usually red) to emphasize that the item is a string.

KEY TERMS

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

REFERENCES

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

Arithmetic Assignment Operators

KENNETH LEROY BUSBEE

Overview of Arithmetic Assignment

The five arithmetic assignment operators are a form of short hand. Various textbooks call them “compound assignment operators” or “combined assignment operators”. Their usage can be explaned 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;`

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to

downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the soruce code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Arithmetic_Assignment.cpp

REFERENCES

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

Lvalue and Rvalue

KENNETH LEROY BUSBEE

Discussion

They refer to on 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:

```
int age;     // variable set up
    then later in the program
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.

```
int age;              // variable set up
int voting_age = 18;  // variable set up with initialization
    then later in the program
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.

```
const int JACK_BENNYS_AGE = 39;  // constant set up
    then later in the program
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.

```
int oldest = 55;  // variable set up with initialization
    then later in the program
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 where in the value of oldest is incremented from 55 to 56.

KEY TERMS

Lvalue
The requirement that the operand on the left side of the assignment operator is modifiable, usually a variable.
Rvalue
Pulls or fetches the value stored in a variable or constant.

REFERENCES

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

Integer Division and Modulus

KENNETH LEROY BUSBEE

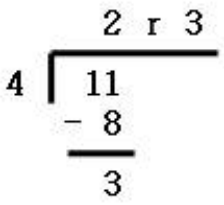
Overview of Integer Division and Modulus

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 acts differently on most computers and is called: **integer division**. Within the C++ programming language the following expression does not give the answer of 2.75 or 2 ¾.

```
11 / 4
```

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

integer division
Division with no fractional parts.



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? 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 C++ programming language 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.

modulus

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 divied by 24 is zero. This is different from: What is the remainder of 6 divided by 24? Six, the remainder part 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

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file. Following the methods of your compiler/IDE, compile and run the program(s). Study the soruce code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Integer_Division_and_Modulus.cpp

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Design a program, to include: understanding the problem, completing Internet research as appropriate, create a pseudocode document and create a test data document.
3. Write the C++ code for a program using appropriate planning documentation that you or another has designed.

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. Integer data types are stored with a mantissa and an exponent.
2. Strings are identified by single quote marks.
3. An operand is a value that receives the operator's action.
4. Arithmetic assignment is a shorter way to write some expressions.
5. Integer division is rarely used in computer programming.

Answers:

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

5. false

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 04 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_04 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- The Problem/Task – I have a friend who is visiting from Europe and he does not understand Fahrenheit temperatures. We need to write a program that allows him to enter the temperature in Fahrenheit (as announced on TV or radio) and convert it to Celsius. Clue 1: Fahrenheit water freezes at 32 degrees and boils at 212 degrees. Celsius water freezes a zero (0) degrees and boils at 100 degrees. Google the internet (how to convert Fahrenheit to Celsius) if you need more help. Clue 2: You can also use Internet sites to do a conversion and thus create your test

data.

- You only need two variables in this program: Fahrenheit and Celsius both of which should be the integer data type. When you convert the Fahrenheit to Celsius you will need to use a floating-point expression doing floating-point calculations for precision. Additionally we want to round up or down the Celsius answer by adding 0.5 to the calculation expression.

- Within your sub-folder: Chapter_04 you will need to create three files: **Lab_04_Pseudocode.txt** and **Lab_04_Test_Data.txt**and **Lab_04.cpp** NOTE: It will be easier to copy some previous files from another assignment and use those copies by renaming them and modifying them as appropriate. The professor is expecting the items you create to have a similar format to those we have been using in the course.
- Create your pseudocode, test data and source code files.
- Build (compile and run) your program. You have successfully written this program when it runs with your test data and gives the predicted results.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 04a – Instructions

Decide on the data type and identifier names for the following:

Problem: A men's clothing store that caters to the very rich wants to create a data base 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.

REFERENCES

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

PART V

PROGRAM CONTROL FUNCTIONS

- Pseudocode Examples for Functions
- Hierarchy or Structure Chart
- Program Control Functions
- Void Data Type
- Documentation and Making Source Code Readable
- Practice

Modularization and Program Layout

KENNETH LEROY BUSBEE

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. Wikipedia: Modular programming

1. **Program Control** – Functions used to simply sub divide and control the program. These functions are unique to the program being written. Other programs may use similar functions maybe even functions with the same name, but the content of the functions are almost always very different.
2. **Specific Task** – Functions designed to be used with several programs. These functions perform a specific task and thus are useable 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 (it's **definition** or the code it will execute)
2. **call** a function
3. declare a function (a **prototype** is a declaration to a complier)

> 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. some communication in with no communication out
3. some communication in with some communication out
4. no communication in with some communication out

INTRODUCTION OF FUNCTIONS WITHIN C++

We are going to consider a simple program that might be used for testing a compiler to make sure that it is installed correctly.
Compiler_Test.cpp source code

```
//*****************************************************
// Filename: Compiler_Test.cpp
// Purpose:  Average the ages of two people
// Author:   Ken Busbee; © Kenneth Leroy Busbee
// Date:     Jan 5, 2009
// Comment:  Main idea is to be able to
//           debug and run a program on your compiler.
//*****************************************************

// Headers and Other Technical Items
```

```
#include <iostream>
using namespace std;

// Function Prototypes

void pause(void);

// Variables

int     age1;
int     age2;
double  answer;

//*******************************************************
// main
//*******************************************************

int main(void)
  {
  // Input
  cout << "\nEnter the age of the first person --->: ";
  cin >> age1;
  cout << "\nEnter the age of the second person -->: ";
  cin >> age2;

  // Process
  answer = (age1 + age2) / 2.0;

  // Output
  cout << "\nThe average of their ages is -------->: ";
  cout << answer;
```

```
  pause();
  return 0;
  }

//*******************************************************
// pause
//*******************************************************

void pause(void)
  {
  cout << "\n\n";
  system("PAUSE");
  cout << "\n\n";
  return;
  }

//*******************************************************
// End of Program
//*******************************************************
```

This program has two functions, one from each of our categories. The technical layout of functions are the same, it is our distinction that creates the two categories based on how a function is being implemented.

Program Control Function

The main program piece in C++ program 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 usually 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. In this simple example, there are no other program control functions.

Specific Task Function

We often have the need to perform a specific task that might be used in many programs. In the Compile_Test.cpp source code above we have such a task that is used to stop the execution of the code until the user hits the enter key. The functions name is: pause. This function is not communicating any information between the calling function and itself, thus the use of the data type void.
general layout of a function

```
<return value data type> function identifier name(<data type> <
  {
  lines of code;

  return <value>;
  }
```

There is no semi-colon after the first line. Semi-colons are used at the end of a statement in C++, but not on the first line when defining a function. Functions have a set of **braces** {} used for identifying a group or block of statements or lines of code. There are normally several lines of code within a function. Lines of code containing the instructions end in a semi-colon. Can you identify the definition of the pause function in the above program example? The pause function definition is after the function main. Though not technically required, most programs list all

functions (program control or specific task) after the function main.

Let's identify the location where the function pause is called. The calling function is the function main and it towards the end of the function. The line looks like:

```
pause();
```

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, and in our example there are none. A semicolon ends the statement or line of code. After our program is compiled and running, the lines of code in the function main are executed and when it gets to the calling of the pause function, the control of the program moves to the pause function and starts executing the lines of code in the pause 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.

Once we know how to define a function and how to call a function, we usually will need to know how to declare a function to the compiler (called a prototype). Because of normal computer programming industry standards, programmers usually list the function main first with other functions defined after it. Then somewhere in the function main, we will call a function. When we convert our source code program to an executable version for running on our computer, the first step of the process is compiling. The compiler program demands to know what the communication will be between two functions when a function is called. It will know the communication (what going in and out as parameters) if the function being called has been defined. But, we have not defined that function yet; it is defined after the function main. To solve this problem, we show the compiler a prototype of what the function will look like

(at least the communication features of the function) when we define it.

```
void pause(void);
```

This line of code looks exactly like the first line in our function definition with one important addition of a semi-colon. Prototypes (or declarations to the compiler of the communications of a function not yet defined) are placed near the top of the program before the function main. **Summary concept: If you call a function before it is defined you must prototype it before it is called.** Looking at our list of the three things you do in conjunction with a function in the order that they normally appear in a program, there is a formal way to:

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

C++ PROGRAM LAYOUT

From the above example, you can see that 2/3 of the program is the two functions. Most C++ programs have several items before the function main. As in the example, they often are:

1. Documentation – Most programs have a comment area at the start of the program with a variety of comments pertinent to the program. Any line starting with two slashes // is a comment and the compiler software disregards everything from the // to the end of the line.
2. #include<iostream> – This line of code inserts a file into the source code. The file contains necessary code to be able to do simple input and output.
3. using namespace std – The C++ compiler has an area where

it keeps the identifier names used in a program organized and it is called a namespace. There is a namespace created in conjunction with the iostream file called: std. This line informs the compiler to use the namespace std where the identifier names in the iostream are established.

4. Function prototypes have already been explained.
5. We need some variables (storage areas) for this program to work. They are defined next.

KEY TERMS

braces
Used to identify a block of code in C++.

function
What modules are called in the two predominant programming languages of today: C++ and Java.
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 sub divide and control the program.
specific task
Functions designed to be used with several programs.

REFERENCES

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

Pseudocode Examples for Functions

KENNETH LEROY BUSBEE

Concept

No standard for pseudocode syntax exists. However, there are some commonly followed conventions to help make pseudocode written by one programmer easily understood by another programmer. The following describes a method for using pseudocode for functions that would be understood by programmers. Five concepts are:

- Use a **beginning phrase word** to start the function
- Use a **communication phrase word** to identify the items being passed into the function
- Use indentation to show the action part of the function
- Use a **communication phrase word** to identify the items being passed out of the function
- Use an **ending phrase word** to end the function
- Use a **calling phrase word** to direct your program to use a fucntion

The following is a suggested outline of function phrase words:

Item/Purpose	Starting Phrase Word	Ending Phrase Word
Beginning	Function	N/A
Communication In	Pass In:	none
Action	N/A	N/A
Communication Out	Pass Out:	none
Ending	N/A	Endfunction
Calling a Function	Call:	none

Examples

Here are some examples showing functions defined in pseudocode using our conventions as described above.
pseudocode: Function with no parameter passing

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

pseudocode: Function with parameter passing

```
Function delay program so you can see the monitor
  Pass In: integer representing tenths of a second
  Using the operating system delay the program
  Pass Out: nothing
Endfunction
```

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
Endfunction
```

KEY TERMS

phrase word
Words used to make pseudocode logic clear to any reader.

REFERENCES

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

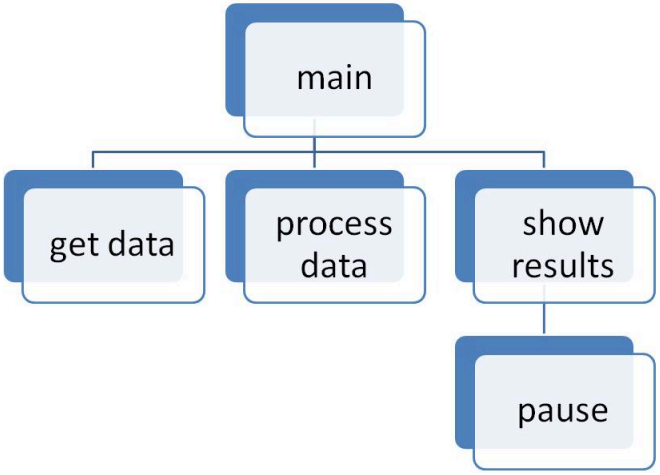
Hierarchy or Structure Chart

KENNETH LEROY BUSBEE

Overview

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

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



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

KEY TERMS

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

REFERENCES

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

Program Control Functions

KENNETH LEROY BUSBEE

Prerequisite Material

Critical to this module is the review of several Connexions modules:

1. Modularization and C++ Program Layout
2. Pseudocode Examples for Functions
3. Hierarchy or Structure Chart

You should review these materials before proceeding. If you are viewing this module on-line, links to these items are in the "Links" box to your right.

Concept of Modularization

The concept is everywhere present in the real world about us. Simply put it is to **take a large complicated problem and to divide it into smaller manageable pieces**. The hierarchy chart of any large organization (government unit, company, university, hospital, etc.) will show levels of people with job titles that indicate a different area of responsibility. Each person is a small piece of the overall workings of the organization. Each person can concentrate on their unique talent or task to make sure it works properly. Collectively they accomplish the goals of the organization.

Additionally, the concept has been around for a long time.

A village of 300 years ago had farmers, tailors, butchers, blacksmiths, etc. Manufacturing is a prime example of not just work being modularized but the product itself is viewed in terms of modules or systems (Example of a automobile: engine, steering, brakes, etc.).

The world of computers, both hardware (equipment) and software (computer programs), also uses this modular concept. Thus, the concept migrates to a single computer program; allowing us to modularize the program into manageable tasks called functions.

Program Control Functions

Program Control functions normally do not communicate information to each other but use a **common area** for variable storage. The rules for how 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. However, with program control functions we use the data communication option of no communication in – with no communication out. Our data variables and constants are placed in a common area available to all functions (called global scope).

The identifier names for program control functions usually imply a task to be accomplished, such as get-data, process-data or show-results. As you learn to write more complicated programs the number of lines of code will increase. Prudence dictates that it would be beneficial to divide the program into functions that perform unique tasks. The larger the program the more need for modularization or creating of program control functions.

Depending on the programming language, there is a formal way to:

1. define a function (it's definition or the code it will execute))
2. call a function
3. declare a function (a prototype is a declaration to a complier)

One of the easier ways to understand program control function is to view an example. Even if you don't know the C++ programming language, you can study the materials to help understand the modularization process.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on some of the links and select "Save Target As" in order to download

some of the files. Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Program_Control_Functions_before_Compiler_Test.cpp

Download from Connexions: Demo_Program_Control_Functions_Pseudocode.txt

Download from Connexions: Demo_Program_Control_Functions_Hierarchy_Chart.jpg

Download from Connexions: Demo_Program_Control_Functions.cpp

Study the Materials Collectively to Understand Modularization

The four items represent a progression from no modularization to modularization:

1. Program code before it is modularized
2. Modularized pseudocode and a hierarchy chart for the program
3. Program code that has been modularized

The simplicity of the program should not be considered during this review. It is obvious that the program does not need modularization. The example is to show or demonstrate how to modularize a program for program control.

KEY TERMS

common area
An area of the program where variables and constants are defined so that they are available to all functions.

REFERENCES

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

Void Data Type

KENNETH LEROY BUSBEE

General Discussion

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

C++ Reserved Word	void
Represent	Nothing
Size	N/A or None
Normal Signage	N/A
Domain (Values Allowed)	None

This data type was added in the transition from "C" to "C++". In "C" by default a function returned an integer data type. Some functions don't return a value of any kind. Thus, the need to have a data type that indicates **nothing** is being returned. The void data type is mainly used in the definition and prototyping of functions to indicate that either nothing is being passed in and/or nothing is being passed out.

KEY TERMS

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

REFERENCES

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

Documentation and Making Source Code Readable

KENNETH LEROY BUSBEE

General Discussion

We are going to consider a simple program that might be used for testing a compiler to make sure that it is installed correctly. Compiler_Test.cpp source code

```
//*****************************************************
// Filename: Compiler_Test.cpp
// Purpose:  Average the ages of two people
// Author:   Ken Busbee; © Kenneth Leroy Busbee
// Date:     Jan 5, 2009
// Comment:  Main idea is to be able to
//           debug and run a program on your compiler.
//*****************************************************

// Headers and Other Technical Items

#include <iostream>
using namespace std;

// Function Prototypes

void pause(void);
```

```
// Variables

int     age1;
int     age2;
double  answer;

//*******************************************************
// main
//*******************************************************

int main(void)
  {
  // Input
  cout << "\nEnter the age of the first person --->: ";
  cin >> age1;
  cout << "\nEnter the age of the second person -->: ";
  cin >> age2;

  // Process
  answer = (age1 + age2) / 2.0;

  // Output
  cout << "\nThe average of their ages is -------->: ";
  cout << answer;

  pause();
  return 0;
  }

//*******************************************************
// pause
//*******************************************************
```

```
void pause(void)
  {
  cout << "\n\n";
  system("PAUSE");
  cout << "\n\n";
  return;
  }

//*******************************************************
// End of Program
//*******************************************************
```

Within the programming industry there is a desire to make software programs easy to maintain. The desire centers in 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. Appropriate use of Comments
4. Banners for Functions
5. Block Markers on Lines by Themselves
6. Indent Block Markers
7. Meaningful Identifier Names Consistently Typed
8. Appropriate use of Typedef

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

programming practice that leads to readable code). Let's cover each item in more detail.

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. Notice the indication of revision dates.

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 the variables are defined. When declaring variable or constants many textbooks put several items on one line; like this:
Common Textbook Defining of Variables

```
float  length, width, height, price_gal_paint, total_area, total
int  coverage_gal_paint, total_gal_paint;
```

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:
Proper Defining of Variables with Vertical Alignment

```
float  length;
float  width;
float  height;
float  price_gal_paint;
int    coverage_gal_paint;
float  total_area;
```

```
int    total_gal_paint;
float  total_cost;
```

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, especially when the number of variables exceeds about twenty.

The lines of code inside either function are also aligned vertically and indented two spaces from the left. The indentation helps set the block off visually.

Appropriate use of Comments

You can see through the source code short little comments that describe an area or section. Note the use of input, processing and output which are part of the IPO concept within the program design.

Banners for Functions

Note the use of comments in the form of a banner before each function.
Comments as a Banner

```
//*******************************************************
// main
//*******************************************************
```

The function name is placed with two lines of asterisks. It makes it extremely easy to find each function definition because you don't have to read the functions to see where the one ends and the next one begins. You can quickly read the function names within the banners.

Block Markers on Lines by Themselves

Within many languages there is a method to identify a group of programming statements as a unit. With C++ the functions use a set of symbols, the braces {}, to identify a block of code, sometimes referred to as a compound statement. Braces are used in other aspects of programs, but for now we will look at this simple example. These braces have a tendency to cause problems, especially when they don't have a proper opening brace associated with a proper closing brace. To solve that problem many programmers simply put a brace on a line by itself and make sure the opening brace and closing brace are in the same vertical column.

Indent Block Markers

A block of code associated with a function or with a control structure is indented two or three spaces. When blocks of code are nested each nesting is indented two or three spaces. In our example above the blocks of code for the function definitions are indented two spaces.

Meaningful Identifier Names Consistently Typed

As the name implies "identifier names" should clearly identify who (or what) you are talking about. Calling you 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, typedefs and other items should use meaningful identifier 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.

Appropriate use of Typedef

Many programming languages have a command that allows for the creation of an identifier name that represents a data type. The new identifier name is described or connected to a real data type. This feature is not demonstrated in the code above and is often a confusing concept. It is a powerful way to help document a program so that it is meaningful, but is often used by more experienced programmers.

KEY TERMS

banners
A set of comment lines used to help separate the functions and other sections of a program.
braces
Used to identify a block of code in C++.

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

comments
Information inserted into a source code file for documentation of the program.
documentation
A method of preserving information useful to others in understanding an information system or part thereof.

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

meaningful
A rule that says identifier names must be easily understood by another reading the source code.
vertical alignment
A method of listing items vertically so that they are easier to read quickly.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Given pseudocode, test data and source code of an existing program, modify the pseudocode and source code to create “program control” functions.

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. Pseudocode has a strict set of rules and is the same everywhere in the computer programming industry.
2. Hierarchy Charts and Structure Charts are basically the same thing.
3. Program Control functions are used to simply sub divide and control the program.
4. The void data type is rarely used in C++.
5. Making source code readable is only used by beginning programmers.

Answers:

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

5. false

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 06 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_06 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select “Save Target As” in order to download the file.

Download from Connexions: Solution_Lab_01.cpp
Download from Connexions: Solution_Lab_01_Pseudocode.txt
Download from Connexions: Solution_Lab_01_Test_Data.txt
Download from Connexions: Solution_Lab_01m_with_Program_Control.cpp
Download from Connexions: Solution_Lab_01m_Pseudocode_with_Program_Control.txt
Download from Connexions: Solution_Lab_01m_Hierarchy_Chart.jpg

Download from Connexions: Solution_Lab_03.cpp
Download from Connexions: Solution_Lab_03_Pseudocode.txt
Download from Connexions: Solution_Lab_03_Test_Data.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Navigate to your sub-folder: Chapter_06. Review the original Lab 01 materials. Compile and run the Lab 01 source code. Then review and compare the original Lab 01 materials to modularized Lab 01 materials taking note of the conversion to "program control" functions. Compile and run the Lab 01m source code. Review as needed the course materials. Email your professor if you have any questions.
- We have supplied the solution to the Lab 03 assignment. Review the Lab 03 assignment by compiling and running the Lab 03 source code.
- You need to copy the Lab 03 source code file and pseudocode file to make the following new files: **Lab_06.cpp** and **Lab_06_Pseudocode.txt**
- Modify the Lab 06 pseudocode file to implement "program control" functions as shown in the demonstration materials.
- Modify the Lab 06 source code file to implement "program control" functions as shown in the demonstration materials.
- Build (compile and run) your program. You have successfully written this program if when it runs and you use the test data [use the same test data as used in Lab 03] it gives the same results as Lab 03.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 06a – Instructions

Create a hierarchy chart for the following pseudocode example.
pseudocode

```
*****************************************************
Filename:  Average_IQ.txt
Purpose:   Average the IQs of two people
Author:    Ken Busbee; © Kenneth Leroy Busbee
Date:      Jan 17, 2009
*****************************************************

Function main
  Pass In: nothing
  Call: get_iqs
  Call: process_iqs
  Call: show_average
  Pass Out: zero to the OS
Endfunction

*******************

Function get_iqs
  Pass In: nothing
  display a message asking user for the IQ of the first person
  get the IQ of the first person from the keyboard
  display a message asking user for the IQ of the second person
  get the IQ of the second person from the keyboard
  Pass Out: nothing
Endfunction

*******************
```

```
Function process_iqs
  Pass In: nothing
  calculate the answer by adding the two IQs and
  dividing by 2.0
  Pass Out: nothing
Endfunction

********************

Function show_average
  Pass In: nothing
  display the answer with an appropriate message
  Call: pause
  Pass Out: nothing
Endfunction

********************

Function pause
  Pass In: nothing
  direct the operating system to pause the program
  Pass Out: nothing
Endfunction

**************************************************

Potential Variables

Data Type         Identifier Name
*********         ***************
integer         iq1
integer         iq2
```

```
double        answer

*************************************************
End of file
```

Problem 06b – Instructions

Identify some problems that make this code “undocumented”, “unreadable” or wrong in some other way.
C++ source code

```
//*******************************************************
// Author:  Ken Busbee; © 2009 Kenneth Leroy Busbee
// Date:    Jan 17, 2009
//*******************************************************

#include <iostream>
using namespace std;

void pause(void);

int         age1, age2;
double         xx;

//*******************************************************
// main
//*******************************************************

int main(void)
  {
  // Input
cout << "\nEnter the age of the first person --->: ";
  cin >> age1;
```

```
    cout << "\nEnter the age of the second person -->: ";
  cin >> age2;

  // Process
  xx = (age1 + age2) / 2.0;

  // Output
  cout << "\nThe average of their ages is -------->: ";
  cout << xx;

  pause();
  return 0;
  }

void pause(void)
  { cout << "\n\n";
  system("PAUSE");
  cout << "\n\n";
  return; }

//*******************************************************
// End of Program
//*******************************************************
```

REFERENCES

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

PART VI

SPECIFIC TASK FUNCTIONS

- Specific Task Functions
- Global vs Local Data Storage
- Using a Header File for User Defined Specific Task Functions
- Practice

Specific Task Functions

KENNETH LEROY BUSBEE

Prerequisite Material

Critical to this module is the review of two Connexions modules:

1. Modularization and C++ Program Layout
2. Program Control Functions

You should review these materials before proceeding. If you are viewing this module on-line, links to these items are in the "Links" box to your right.

General Concept

Program Control functions which might have similar identifier names usually perform slightly different tasks in one program to another. Looking at the organizational chart or hierarchy chart for two companies, both might have a vice president of production, but producing automobiles is different than producing ice cream. Similar but different. As you go down deeper into an organization you might find the job title of security guard. Notice that the security guard at the automobile plant and the security guard at the ice cream plant have exactly the same job. In fact, they are most likely interchangeable. Within programming when a task gets specific it might be useable in several programs. The calculation of leap year is a good example. Needed for the verification of dates, is there or is there not a 29^{th} of February for this year. Needed in thousands of programs.

Specific Task Functions

To create good Specific Task functions you need to do all communication needed via parameter passing. Thus all programs that will use the function will communicate in precisely the same way. In our leap year example, you would communicate into the function the year and the function would return the communication of true or false; meaning it is a leap year and there is a 29th of February (true) or it is not a leap year (false).

The ability to modularize our program into specific task functions means that we can write the specific task function once making sure it works correctly, then reuse it over and over in many programs. As you can guess there is a balance. Most programs will have some program control functions and some specific task functions. The key to deciding if the function should be a specific task function is usually rooted in the uniqueness of the task so that it can be used in many programs. Specific task functions once created are usually placed into a **user defined library** then shared with others for use in many programs.

KEY TERMS

user defined library
A file containing specific task functions created by individuals to be used in many programs.

REFERENCES

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

Global vs Local Data Storage

KENNETH LEROY BUSBEE

General Discussion

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

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

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

Using a Header File for User Defined Specific Task Functions

KENNETH LEROY BUSBEE

Concept: User Defined Specific Task Functions

Most companies have certain tasks that are unique to their company. Collectively the programming staff may decide to build several functions and organize them into one or more user libraries. Specific task functions are often built using a testing shell program. The sole purpose of the testing shell program is to create the specific task functions and to test them to insure that they are working properly. Think of a clam, its shell surrounds the important part, the pearl. A testing shell program surrounds the specific task function (the important part). Usually the testing shell program will be used to create several functions that will be placed into a user defined library. The process flows as follows:

1. The **testing shell** program with the specific task functions is built and thoroughly tested.
2. A copy of the **testing shell** source code is saved as the **header file** that once modified will be placed in the user library. You delete the main part of the program leaving a comments area, any needed include file references and the specific task functions.
   Header File Creation

Monitor_Testing_Shell.cpp	udst_monitor.h
Program Comments	Program Comments Edit as appropriate
Headers & Other	~~Headers & Other~~ Delete all but needed libraries. Keep: #include <ctime>
Prototypes	~~Prototypes~~
Variables	~~Variables~~
main	~~main~~
clear_m	clear_m
delay_m	delay_m the clock function needs: ctime
pause_m	pause_m

Creating a header file from a copy of the testing shell.

3. Another copy of the **testing shell** source code is saved as the prototypes file. This is a text file that retains only the prototypes for the functions that were placed into the **header file**. The functions should be using meaningful identifier names, thus the prototypes should provide adequate information to others on how to call the function with appropriate parameter passing.
   Prototypes File Creation

Creating a prototypes file from a copy of the testing shell.

4. Another copy of the **testing shell** source code is saved as the verify header program. You delete the functions prototypes and definitions then provide an include that points to the header file. This program is compiled and run to make sure the **header file** is working properly.
   Verify Header File Creation

Monitor_Testing_Shell.cpp	Monitor_Verify_Header.cpp
Program Comments	Program Comments Edit as appropriate
Headers & Other	Headers & Other Add user libraries as needed: #include "c:\\Dev-Cpp\\user_library\\monitor.h"
Prototypes	~~Prototypes~~
Variables	Variables
main	main
clear_m	~~clear_m~~
delay_m	~~delay_m~~
pause_m	~~pause_m~~

Creating a verify header file from a copy of the testing shell.

A good way to understand the concept is to review the four files described above that have been created by a programmer. We will be using the C++ programming language, however the code is easy to understand and will serve our needs well at explaining the concepts; even if you are not familiar with C++.

Demonstration Using C++

Creating a Folder or Sub-Folder for the Four Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Monitor_Header

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Four Files

Download and store the following files to your storage device in the appropriate folder. You may need to right click on some of the links and select "Save Target As" in order to download some of the files.

Download from Connexions: Monitor_Testing_Shell.cpp
Download from Connexions: udst_monitor.h
Download from Connexions: udst_monitor_prototypes.txt
Download from Connexions: Monitor_Verify_Header.cpp

Study the Files Collectively to Understand the Concepts

Take a few moments to review the files in conjunction with the concept discussion above. You should compile and run the **Monitor_Testing_Shell.cpp** program.

Creating a Folder or Sub-Folder for your User Library

Depending on your compiler/IDE, you should decide where to create a folder that will hold the header files you create. We suggest that you create the folder in conjunction with the compiler/IDE software. If you were using the Bloodshed Dev-C++ 5 compiler/IDE you most likely installed the compiler/IDE software at: **C:\Dev-Cpp** if you installed it on your machine or at: **DriveLetter:\Dev-Cpp** (where the **DriveLetter** is the drive that represents your flash drive) if you installed it on a flash drive. We suggest that you create a sub-folder at that location named:

- user_library

The path of: **C:\Dev-Cpp\user_library** would be created as the location for your user library if using your machine installation. You can literally place it anywhere and name the library any name, but once you decide on a place and name; you do not want to move or rename the folders.

Placing the Header File into the User Library

You need to copy the **udst_monitor.h** file placing it into the user_library folder just created. As you can guess the udst stands for user defined specific task. The functions within this header file would be used to control the interaction a user has with the monitor. The .h is a convention of the C++ programming language and indicates a header file. Thus the identifier name for the header file is very meaningful and descriptive.

Verify that the Header File Works Properly

Review the **Monitor_Verify_Header.cpp** source code file and note the two include commands are different.

1. The Standard Library uses a less than and a greater than to bracket the Standard Library name of: iostream
2. The user library uses quote marks to bracket the location of the header file. This identifies to the complier that we are specifying the exact file we want. We provide a complete file specification (drive, path information, filename and extension).
3. Because this item is technically a string within C++, we must use two back slashes between the drive, path(s) and filename. This is because the first back slash assumes that

the next character is an escape code and if we really don't want an escape code but a back slash, the second back slash says no I wanted a back slash.This string: "C:\\Dev-Cpp\\user_library\\udst_monitor.h" will be interpreted to mean: **C:\Dev-Cpp\user_library\udst_monitor.h**

Depending on what drive you are using, what path folder structure you are using and what you called your folder; you may need to correct the include reference within the source code so that it properly references the header file.

Compile and run the Monitor_Verify_Header.cpp program. Note: It should work exactly as the Monitor_Testing_Shell.cpp program.

KEY TERMS

header file
A file that contains items we want to have included toward the top of our source code.

testing shell
A program used to create specific task functions.
udst
User Defined Specific Task

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Given a testing shell program already coded and tested, create a user defined specific task header file, a user defined specific task prototypes document and a source code program to verify that the header file works properly.

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. Scope refers to a brand of mouth wash.
2. User defined specific task functions are usually placed into a user defined library.
3. Local and global data storage is associated with the concept of scope.
4. Creating a header file for user defined specific task functions is a difficult task.
5. The stack is part of the computer's memory used for storage of data.

Answers:

1. false – Although Scope is a brand of mouth wash; we are looking for the computer related definition.
2. true

3. true
4. false – It may seem difficult at first, but with a little practice it is really quite easy.
5. true

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 07 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_07 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_07_Testing_Shell.cpp

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Navigate to your sub-folder: Chapter_07. Compile and run the Lab 07 Testing Shell source code. Note: This program uses an include file that points to the "udst_monitor.h" file as explained in Connexions Chapter 7 materials.

- Following same process as shown in the Connexions module "Using a Header File for User Defined Specific Task Functions" that is within the Chapter 7 materials; make the following files: **udst_us_to_metric.h** and **udst_us_to_metric_prototypes.txt** and **Lab_07_Verify_Header.cpp**
- Copy the header file to your user library, then build (compile and run) your verify header program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 07a – Instructions

Create the pseudocode to solve the following specific task function:

Problem: An interior designer always needs to calculate the area of a room to determine the amount of floor covering needed (usually carpet). The rooms are rectangular with the dimensions measured in feet (with decimal fractions). The function however needs to return square yards. Hint: There are 3 lineal feet to a yard.

Problem 07b – Instructions

Create test data for the following specific task function:

Problem: An interior designer always needs to calculate the area of a room to determine the amount of floor covering needed (usually carpet). The rooms are rectangular with the dimensions measured in feet (with decimal fractions). The function however needs to return square yards. Hint: There are 3 lineal feet to a yard.

REFERENCES

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

PART VII

STANDARD LIBRARIES

- Standard Libraries
- Practice

Standard Libraries

KENNETH LEROY BUSBEE

Overview of Standard Libraries

Many common or standard functions, whose definitions have 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 function organized into several libraries For example, within C++ many math functions exist and have been coded (and placed into libraries). These functions were written by programmers and tested to insure that they work properly. In most cases the functions were reviewed by several people to double and triple check to insure 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.

A 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 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. The prototype is:

```
int abs(int 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. The function is of data type int, meaning that the function will return an integer value. It is obvious that the integer value returned is the answer to the question, "What is the absolute value of the integer that is being passed into the function". This function is passed only one value; an int number. 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).

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on some

of the links and select “Save Target As” in order to download some of the files. Following the methods of your compiler/IDE, compile and run the program(s). Study the source code and/or other file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Standard_Libraries.cpp

Download from Connexions: Demo_Standard_Libraries_Listing.txt

KEY TERMS

abs
A function within the cmath standard library in C++ which stands for absolute.
confidence
The reliance that Standard Library functions work properly.

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

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Given a testing shell program already coded and tested, add another specific task function, and test it, then create a user defined specific task header file, a user defined specific task prototypes document and a source code program to verify that the header file works properly.

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. The standard library is a set of specific task functions that have been added to the programming language for universal use.
2. Programmers should not have confidence that standard library functions work properly.
3. It would be easier to write programs without using specific task functions.

Answers:

1. true
2. false
3. false

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 08 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_08 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_08_Testing_Shell.cpp

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Navigate to your sub-folder: Chapter_08. Compile and run the Lab 08 Testing Shell source code. Note: This program uses an include file that points to the "udst_monitor.h" file as explained in Connexions Chapter 7 materials.
- You need to add another function to this testing shell titled:

area_triangle that is to calculate the area of a triangle. Define the function, prototype it, and within the function main add an area for calling the function (similar to the existing functions with test data). Be confident that it is working properly.

- Following same process as shown in the Connexions module “Using a Header File for User Defined Specific Task Functions” that is within the Chapter 7 materials; make the following
  files: **udst_geo_area.h** and **udst_geo_area_prototypes.txt** a nd **Lab_08_Verify_Header.cpp**
- Copy the header file to your user library, then build (compile and run) your verify header program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 08a – Instructions

Write the C++ code to do the following specific task function.
pseudocode

```
Function area_regular_hexagon
  Pass In: side
  Calculate: side times side times 3 times the square root of 0
  Pass Out: the calculation
Endfunction
```

REFERENCES

- cnx.org: Programming Fundamentals – A Modular

PART VIII

CHARACTER DATA, SIZEOF, TYPEDEF, SEQUENCE

- Character Data Type
- Sizeof Operator
- Typedef – An Alias
- Sequence Operator
- Practice

Character Data Type

KENNETH LEROY BUSBEE

Overview of the Character Data Type

The **character** data type basically represents individual or single characters. Characters comprise a variety of symbols such as the alphabet (both upper and lower case) the numeral digits (0 to 9), punctuation, etc. All computers store character data in a one byte field as an integer value. Because a byte consists of 8 bits, this one byte field has 28 or 256 possibilities using the positive values of 0 to 255.

Most microcomputers 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.

The character data type attributes include:

C++ Reserved Word	char
Represent	Single characters
Size	1 byte
Normal Signage	Unsigned (positive values only)
Domain (Values Allowed)	Values from 0 to 127 as shown in the standard ASCII Character Set, plus values 128 to 255 from the Extended ASCII Character Set
C++ syntax rule	Single quote marks – Example: 'A'

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your

compiler/IDE, compile and run the program(s). Study the soruce code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Character_Data_Type.cpp

KEY TERMS

ASCII
American Standard Code for Information Interchange

character
A data type representing single text characters like the alphabet, numeral digits, punctuation, etc.
single quote marks
Used to create character type data within the C++ programming language.

REFERENCES

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

Sizeof Operator

KENNETH LEROY BUSBEE

Overview

Every data item, constants and variables, not only have a data type, but the data type determines how many bytes the item will use in the memory of the computer. The size of each data type varies with the complier being used and the computer. This effect is known as being machine dependent. Additionally, there have been some size changes with upgrades to the language. In "C" the int data type was allocated 2 bytes of memory storage on an Intel compatible central processing unit (cpu) machine. In "C++" an int is allocated 4 bytes.

There is an operator named "sizeof (...)" that is a unary operator, that is it has only one operand. The operand is to the right of the operator and is placed within the parentheses if it is a data type. The operand may be any data type (including those created by typedef). If the operand is an identifier name it does not need to go inside a set of parentheses. It works for both variable and memory constant identifier names. This operator is unique in that it performs its calculation at compile time for global scoped items and at run time for local scoped items. Examples:

```
cout << "The size of an integer is: " << sizeof (int);
```

The compiler would determine the byte size of an integer on the specific machine and in essence replaces the sizeof operator with a value. Integers are usually 4 bytes long, thus the line of code would be changed to:

```
cout << "The size of an integer is: " << 4;
```

If you place an identifier name that represents a data storage area (variable or memory constant), it looks at the definition for the identifier name. NOTE: the parentheses are not needed and often not included for an identifier name.
sizeof with a Variable

```
double money;     // variable set up with initialization
    then later on in the program
cout << "The size of money is: " << sizeof money;
```

The compiler would determine the byte size of money by looking at the definition where it indicates that the data type is double. The double data type on the specific machine (usually 8 bytes) would replace the code and it would become:

```
cout << "The size of money is: " << 8;
```

KEY TERMS

sizeof
An operator that tells you how many bytes a data type occupies in storage.

REFERENCES

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

Typedef - An Alias

KENNETH LEROY BUSBEE

General Discussion

The typedef statement allows the programmer to create an alias, or synonym, for an existing data type. This can be useful in documenting a program. The C++ programming language syntax is:

```
  typedef <the real data type> <the alias identifier name>;
```

Let's say a programmer is using a double data type to store the amount of money that is being used for various purposes in a program. He might define the variables as follows:
Regular Definition of Variables

```
double  income;
double  rent;
double  vacation;
```

However, he might use the typedef statement and define the variables as follows:
Using typedef when Defining Variables

```
typedef double cash;
    the typedef must be defined before its use
cash  income;
cash  rent;
cash  vacation;
```

The typedef statement is not used very often by beginning

programmers. It usually creates more confusion than needed, thus stick to using the normal data types at first.

KEY TERMS

typedef
Allows the programmer to create an alias, or synonym, for an existing data type.

REFERENCES

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

Sequence Operator

KENNETH LEROY BUSBEE

General Discussion

The **sequence** (or comma) operator is used to separate items. It has several uses, four of which are listed then demonstrated:

1. To separate identifier names when declaring variables or constants
2. To separate several parameters being passed into a function
3. To separate several initialization items or update items in a for loop
4. Separate values during the initialization of an array

This first example is often seen in textbooks, but this method of declaring variables is not preferred. It is difficult to quickly read the identifier names.

```
int pig, dog, cat, rat;
```

The following vertical method of declaring variables or constants is preferred.

Preferred Vertical Method of Defining Variables

```
int  pig;
int  dog;
int  cat;
int  rat;
```

The data types and identifier names (known as parameters) are separated from each other. This example is a function prototype.

```
double area_trapezoid(double base, double height,
double top);
```

In the syntax of a for loop you have three parts each separated by a semi-colon. The first is the initialization area which could have more than one initialization. The last is the update area which could have more than one update. Mutiple initializations or updates use the comma to separate them. This example is only the first line of a for loop.

```
for(x = 1, y = 5; x < 15; x++, y++)
```

The variable ages is an array of integers. Initial values are assigned using block markers with the values separated from each other using a comma.

```
int ages[] = {2,4,6,29,32};
```

KEY TERMS

sequence
An operator used to separate multiple occurrences of an item.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Given appropriate documents produced by a System Analyst, create planning documents (pseudocode and test data), then a source code program that accomplishes the goals of the program.

REVIEW QUESTIONS

Answer the following statements as either true or 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.

Answers:

1. false
2. true

3. false
4. false
5. true

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 09 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_09 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_09_Narrative_Description.txt

Download from Connexions: Lab_09_Aerial_View_Center_Pivot_Irrigation.jpg

Download from Connexions: Lab_09_Hierarchy_Chart.jpg

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Review the Connexions module "Systems Development Life Cycle" within the Chapter 1 materials. Think of yourself as a programmer assigned to a project during the Implementation phase with your professor as the System Analyst.
- Navigate to your sub-folder: Chapter_09. Review the first two items provided by the system analyst which he produced during the Design phase of the Systems Development Life Cycle. These two documents historically would have been printed and be placed into a program documentation folder. The items you produce in creating the program would be added to the folder. However, shifting to our paperless view of the world, today these items might be created and stored electronically in electronic folders (which is basically what we are doing by using our sub-folder titled: Chapter_09). The third item, the hierarchy chart, would normally be produced by the programmer. However, given your inexperience, the system analyst has created it for you. Make sure you understand what the program is to do. Any questions ask the system analyst (aka your professor).

The narrative description for this lab assignment describes how farmers in the mid-west part of the United States irrigate a piece of land using a circular irrigation system. This practice also known to as center pivot irrigation is not unique to the United States. Google "map Qatar", click on the map and switch to the "Satellite" view, zoom in and notice that there are several spots in this small middle eastern country where this type of

irrigation is being used. “These systems are found and used in all parts of the world…” which supports the appropriateness of this programming problem to all students.

- Design the program and create your test data by building a **Lab_09_Pseudocode.txt** file and a **Lab_09_Test_Data.txt** file. WARNING: Don’t touch the compiler/IDE. Don’t start by creating the source code file. Creating the source code then producing the planning documentation afterwards is a bad habit that beginning programmers often acquire. NOTE: In your pseudocode document you do not need to create any pseudocode for the Standard Library or User Library functions. Just indicate that you call them from the Program Control functions. If needed, review the Connexions module “Pseudocode Examples for Functions” within the Chapter 6 materials. HINT: Copying the pseudocode and test data files from the Chapter 06 folder might be a good way to start building these items.
- After you have successfully planned the document and created your test data; create the source code file naming it: **Lab_09.cpp** HINTS: Using a previous source code file as your starting file makes sense. The file in the Chapter 06 folder might be a good start. You might want to copy some of the include information from the Verify Header code in Chapter 08 into your Lab 09 source code file.
- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 09a – Instructions

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?

REFERENCES

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

PART IX

INTRODUCTION TO STRUCTURED PROGRAMMING

- Structured Programming
- Pseudocode Examples for Control Structures
- Flowcharting
- Practice

Structured Programming

KENNETH LEROY BUSBEE

Introduction

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 four main categories of control structures:

- **Sequence** – Very boring. Simply do one instruction then the next and the next. Just do them in a given sequence or in 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.

- **Branching** – A control 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 whilestructure (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. "[1]

Introduction to Selection Control Structures

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 multiway 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.

We have mentioned that the **if then else** control structure belongs to the selection category and is a two-way selection.
if then else control structure

```
if (age > 17)
  {
  cout << "You can vote.";
  }
else
  {
  cout << "You can't vote.";
  }
```

Introduction to Iteration Control Structures

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, counting loops, and test after loops. A question using Boolean concepts usually controls how long the loop will execute.

We have mentioned that the **while** control structure belongs to the iteration category and is a test before loop.
while control structure

```
counter = 0;
while (counter < 5)
  {
  cout << "\nI love computers!";
  counter ++;
  }
```

KEY TERMS

branching
A control 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 you select between two or more choices.

sequence
A control structure where you do the items in the sequence listed.
structured programming
A method of planning programs that avoids the branching category of control structures.

Footnotes

- 1 Structured programming from Wikipedia

REFERENCES

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

Pseudocode Examples for Control Structures

KENNETH LEROY BUSBEE

Overview

No standard for pseudocode syntax exists. However, there are some commonly followed conventions to help make pseudocode written by one programmer easily understood by another programmer. Most of these conventions follow two concepts:

- Use indentation to show the action part of a control structure
- Use an ending phrase word to end a control structure

The sequence control structure simply lists the lines of pseudocode. The concern is not with the sequence category but with selection and two of the iteration control structures. The following are commonly used ending phrase-words:

Control Structure	Ending Phrase Word
If then Else	Endif
Case	Endcase
While	Endwhile
For	Endfor

The **Do While** and **Repeat Until** iteration control structures don't need an ending phrase-word. We simply use the first word,

then the action part, followed by the second word with the test expression. Here are some 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
```

pseudocode: Case

```
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."
Endcase
```

Iteration (Repetition) Control Structures

pseudocode: While

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

pseudocode: For

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

```
Endfor
```

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
```

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Given pseudocode, write the C++ code for a program that uses if then else and while control structures.

REVIEW QUESTIONS

Answer the following statements as either true or 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

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 10 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_10 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_10_Pseudocode.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Create a source code file from the Lab_10_Pseudocode.txt file. Name it: Lab_10.cpp
- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions

from your professor/instructor for submitting it for grading.

Problems

Problem 10a – Instructions

List the four categories of control structures and provide a brief description of each category.

REFERENCES

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

PART X

TWO WAY SELECTION

- If Then Else
- Boolean Data Type
- Relational Operators
- Compound Statement
- Practice

If Then Else

KENNETH LEROY BUSBEE

Introduction to Two Way Selection

Traditional Two Way Selection

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's false
  do this
```

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

```
if expression is true
  then do this
else because it's 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
```

Two Way Selection within C++

The syntax for the if then else control structure within the C++ programming language is:

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

Note: The test expression is within the parentheses, but this is not a function call. The parentheses are part of the control structure. Additionally, there is no semicolon after the parenthesis following the expression.

KEY TERMS

if then else
A two way selection control structure.
mutually exclusive
Items that do not overlap. Example: true and false.

REFERENCES

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

Boolean Data Type

KENNETH LEROY BUSBEE

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. The rules within the C++ programming language are:

C++ Reserved Word	bool
Represent	Logical concepts of true and false
Size	Usually 1 byte
Normal Signage	Unsigned
Domain (Values Allowed)	0 meaning false, and 1 meaning true
C++ syntax rule	true and false are reserved words that can be used as values in expressions
C++ concept/ rule	Any value from any data type can be demoted into a Boolean data type with zero representing false and all non-zero values representing true.

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 expressionthat 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 C++ every data type can be used as a Boolean expression because the value of any data type within C++ can be demoted into a Boolean value.

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.

Within the C++ programming language the Boolean data type is one of the standard or basic data types and is a member of the integer family.

KEY TERMS

Boolean
A data type representing the concepts of true and false.
test expression
An expression used to control programming structures.

REFERENCES

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

Relational Operators

KENNETH LEROY BUSBEE

Overview of the Relational Operators

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.

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

Operator symbols and/or names vary with different programming languages. The C++ programming language operators with their meanings are:

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

Evaluate the following Boolean expressions:

1. 9 < 25
2. 9 < 3
3. 9 > 14
4. 9 <= 17
5. 9 >= 25
6. 9 == 13
7. 9 != 13
8. 9 !< 25

The answers to Boolean expressions within the C++ programming language are a value of either 1 for true or 0 for false.

Be careful. In math you are familiar with using this symbol = to mean equal and ≠ to mean not equal. In the C++ programming language the ≠ is not used and the = symbol means assignment.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file. Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Relational_Operators.cpp

REFERENCES

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

Compound Statement

KENNETH LEROY BUSBEE

The Need for a Compound Statement

For illustration we will use the syntax for the if then else control structure within the C++ programming language. However this problem generally exists for all control structures within any language that requires the use of compound statements. The syntax is:

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

Within the C++ programming language there can be **only one statement listed as the action part of a control structure**. Often, we will want to do more than one statement. This problem is overcome by creating a **compound statement**. The brace symbols – the opening { and the closing } – are used to create a compound statement. For example:

```
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 C++ programming industry encourages the use of indentation (to see the action parts clearly) and the use of compound statements (braces), even if there is only one action. 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 for C++ programmers and programmers in any other language that require the use of compound statements with the control structures.

Other Uses of a Compound Statement

"A compound statement is a unit of code consisting of zero or more statements. It is also known as a **block**. The compound statement allows a group of statements to become one single entry. You used a compound statement in your first program when you formed the body of the function main. All C++ functions contain a compound statement known as the function body.

A compound statement consists of an opening brace, optional declarations, definitions, and statements, followed by a closing brace. Although all three are optional, one should be present."[1]

KEY TERMS

block
Another name for a compound statement.

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

Footnotes

- 1 Behrouz A. Forouzan and Richard F. Gilberg, Computer Science A Structured Approach using C++ Second Edition (United States of America: Thompson – Brooks/Cole, 2004) 100.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Given pseudocode, write the C++ code for a program that uses the if then else control structure.

REVIEW QUESTIONS

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

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.

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 11 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_11 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select “Save Target As” in order to download the file.

Download from Connexions: Lab_11_Pseudocode.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Create a source code file from the Lab_11_Pseudocode.txt file. Name it: Lab_11.cpp

- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 11a – Instructions

Create a table with the six relational operators and their meanings.

Problem 11b – Instructions

Explain why we are using the "if then else" to manipulate the input data in the example below.
C++ source code

```
cout << "\nEnter one side of the rectangle --------->: " ;
cin >> side1;
cout << "\nEnter the other side of the rectangle --->: " ;
cin >> side2;

if (side1 > side2)
  {
  length = side1;
  width = side2;
  }
else
  {
  length = side2;
  width = side1;
  }
```

REFERENCES

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

PART XI

MULTIWAY SELECTION

- Nested If Then Else
- Logical Operators
- Case Control Structure
- Branching Control Structures
- Practice

Nested If Then Else

KENNETH LEROY BUSBEE

Introduction to Mulitway Selection

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.

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. In some situations there is no default action. Consider:

```
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
```

The last if then else control structure has no “else”. It’s implied “else do nothing”. Without the default the multiway selection could be written as a series of “if then without the else” structures. Consider:

```
if age equal to 18
  you can now vote
if age equal to 39
  you are middle aged
if age equal to 65
  you can consider retirement
```

We have shown two ways to accomplish multiway selection. The choice of using nested if then else control structures or a series of if then control structures is decided on the existence of a default action (you must use nested if then else) or programmer preference if there is not a default action (you may use nested if then else or a series of if then control structures).

if then else Syntax within C++

The syntax for the if then else control structure within the C++ programming language is:
C++ source code: Layout of an if then else

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

```
else
  {
  statement;
  }
```

The test expression is within the parentheses, but this is not a function call. The parentheses are part of the control structure. Additionally, there is no semicolon after the parenthesis following the expression.

C++ Example

Multiway selection is often needed to cover all possibilities. Assume that the user has been prompted for the ages of two people with the answers stored in variables named age1 and age2. Consider:
C++ source code

```
if(age1 > age2)
  {
  cout << "\n\nThe first person is older.";
  }
else
  {
  cout << "\n\nThe second person is older.";
  }
```

What if the two persons are the same age? The program incorrectly says the second person is older. To solve this we must handle all three possibilities. Consider this mulitway selection example:
C++ source code

```
if(age1 == age2)
```

```
  {
  cout << "\n\nThey are the same age.";
  }
else
  {
  if(age1 > age2)
    {
    cout << "\n\nThe first person is older.";
    }
  else
    {
    cout << "\n\nThe second person is older.";
    }
  }
```

KEY TERMS

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

nested control structures
Placing one control structure inside of another.

REFERENCES

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

Logical Operators

KENNETH LEROY BUSBEE

Overview of the Logical Operators

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. The answers to Boolean expressions within the C++ programming language are a value of either 1 for true or 0 for false. 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. The C++ programming language operators with their meanings are:

C++ Operator	Meaning	Comment	Typing
&&	Logical and		two ampersands
||	Logical or		two vertical dashes or piping symbols
!	Logical not	unary	the exclamation point

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
```

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

```
1 && 1
```

or

```
true && true
```

The final evaluation of the expression is: 1 meaning 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)
```

Truth Tables

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

Logical and (&&)

x	y	x && y
false	false	false
false	true	false
true	false	false
true	true	true

Logical or (||)		
x	**y**	**x ||y**
false	false	false
false	true	true
true	false	true
true	true	true

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

Examples

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

Everyday 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 view point 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. Evaluate the following Logical Boolean expressions:

1. 25 < 7 || 15 > 36
2. 15 > 36 || 3 < 7
3. 14 > 7 && 5 <= 5
4. 4 > 3 && 17 <= 7
5. ! false
6. ! (13 != 7)
7. 9 != 7 && ! 0
8. 5 > && 7

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select “Save Target As” in order to download the file.

Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Logical_Operators.cpp

KEY TERMS

logical operator
An operator used to create complex Boolean expressions.
truth tables
A common way to show logical relationships.

REFERENCES

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

Case Control Structure

KENNETH LEROY BUSBEE

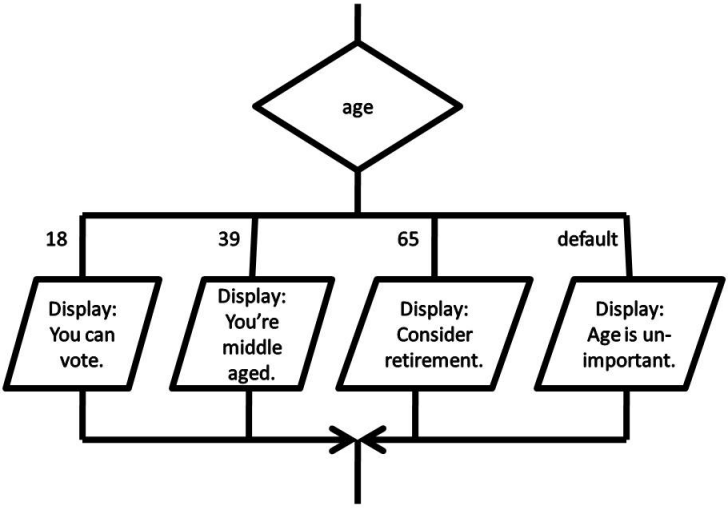
Traditional Case Control Structure

Multiway Selection using the Case Structure

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 un-important
```

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



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 where in 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.

C++ Code to Accomplish Multiway Selection

Using the same example as above, here is the C++ code to accomplish the case control structure.

C++ source code – case structure with integers

```
switch (age)
  {
  case 18: cout << "\nYou can vote.";
           break;
  case 39: cout << "\nYou're middle aged.";
           break;
  case 65: cout << "\nConsider retirement.";
           break;
  default: cout << "\nAge is un-important.";
  }
```

The first thing you should note is that the C++ programming language does not formally have a case control structure. It does have a switch control structure but it acts differently than the traditional case control structure. We use a break (which is a branching control structure) with the switch to make it act like the traditional case structure. This is one of the few allowable ways to use the switch with break within the C++ programming language to simulate the traditional case structure. All other uses of the switch or break are to be avoided if you are to stay within the bounds of good structured programming techniques.

The value in the variable age is compared to the first "case" (note: **case** is one of the C++ reserved words) 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 cout is executed which displays "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.

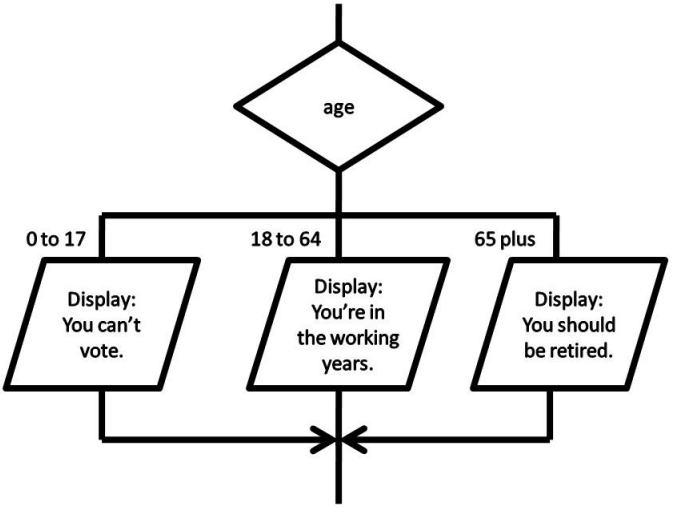
Most programming languages, including C++, require the listed values for the case control structure be of the integer family of data types. This basically means either an integer or

character data type. Consider this example that uses character data type (choice is a character variable):
C++ source code – case structure with characters

```
switch (choice)
  {
  case 'A': cout << "\nYou are an A student.";
            break;
  case 'B': cout << "\nYou are a B student.";
            break;
  case 'C': cout << "\nYou are a C student.";
            break;
  default:  cout << "\nMaybe you should study harder.";
  }
```

Limitations of the Case Control Structure

Most programming languages, including C++, 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."
Endcase
```

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 text book 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 and it uses the C++ programming language to build our initial programming skills. It is not a created with the intent to cover the C++ programming language in detail, despite the fact that at times we have to cover C++ language mechanics.

KEY TERMS

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

REFERENCES

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

Branching Control Structures

KENNETH LEROY BUSBEE

Discussion

The branching control structures allow the flow of execution to jump to a different part of the program. The common branching control structures that are used with other control structures are: break, continue and goto. These are rarely used in modular structured programming with one exception. That exception is in relation to creating the case within the selection category of control structures. Within C++ the break is used with the switch to create a structure that acts like the traditional case structure. There is one other branching control structure that is often not viewed as branching control structure. It is: return; which is used with functions. Thus, there are two commonly used branching control reserved words used in C++; break and return. Additionally, we will add to our list of branching items a pre-defined function commonly used in the C++ programming language of: exit; that is part of the C standard library (cstdlib). Some definitions:

Definitions

branching control structures
Allow the flow of execution to jump to a different part of the program.
break
A branching control structure that terminates the existing

structure.

continue

A branching control structure that causes a loop to stop its current iteration and begin the next one.

goto

A branching control structure that causes the logic to jump to a different place in the program.

return

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

exit

A pre-defined function used to prematurely stop a program and jump to the operating system.

We will discuss each item indicating which ones are allowed or not allowed within good structured programming practices.

Examples

break

The break is used in one of two ways; with the switch (a C++ programming structure) to make it act like a case structure (it's more common name within most programming languages) or as part of a looping process to break out of the loop. The first usage is allowed in good structured programming and the second is not allowed in good structured programming.

C++ source code

```
switch (age)
  {
  case 18: cout << "\nYou can vote.";
           break;
```

```
  case 39: cout << "\nYou are middle aged.";
           break;
  case 65: cout << "\nYou are at retirement age.";
           break;
  default: cout << "\nYour current age is not important.";
  }
```

The following is an unauthorized use of break in a loop and it gives the appearance that the loop will execute 8 times, but the break statement causes it to stop during the fifth iteration.
C++ source code

```
counter = 0;
while(counter < 8)
  {
  cout << counter << endl;
  if (counter == 4)
    {
    break;
    }
  counter++;
  }
```

continue

The continue structure is not allowed in good structured programming. 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.
C++ source code

```
for(counter = 0; counter < 8; counter++)
  {
  if (counter == 4)
```

```
    {
    continue;
    }
  cout << counter << endl;
  }
```

goto

The goto structure is not allowed in good structured programming. It is with a certain amount of hesitancy that we even show it. Many textbooks do not cover the goto. Within the C++ programming language you create a label with an identifier name followed by a colon. You use the command word goto followed by the label. A label can be used before it is declared.
C++ source code

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

return

The return is allowed in good structured programming, but only at the end of a function. A function should not pre-maturely end by having the logic of the function have it terminate by jumping back to the function that called it.
C++ source code

```
//*****************************************************
// get data
```

```
//*******************************************************

void get_data(void)
  {
  // Input - Test Data - 5678.9, 5432.1
  cout << "\nEnter the length of the property in feet --->: ";
  cin >> property_length;
  cout << "\nEnter the width of the property in feet ---->: ";
  cin >> property_width;
  return;
  }
```

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 pre-maturely stop the execution of the program. Within the C++ programming language 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. In our example the value returned to the operating system is the value of the constant named: EXIT_FAILURE.
C++ source code

```
inData.open(filename);  //Open input file
if (!inData)            //Test to see if file was opened
  {
  cout << "\n\nError opening file: " << filename << "\n\n";
  pause();              //Pause - user reads message
```

```
exit(EXIT_FAILURE);   //Allows a pre-mature jump to OS
}
```

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Identify which selection control structures are two-way selection and which are multiway selection.
3. Understand, define and/or explain case, switch and nested if then else.
4. Be able to write pseudo code or flowcharting for the case control structure.
5. Be able to write C++ source code for a case structure using equality and listed values (switch with break to act like a case structure).
6. Be able to write C++ source code for a case structure using ranges of values or floating-point values (nested if then else to act like a case structure).
7. When feasible, be able to convert C++ source code from switch acting like a case to nested if then else and vice versa.

REVIEW QUESTIONS

Evaluate the following Logical Boolean expressions:

1. 25 > 39 || 15 > 36
2. 19 > 26 || 13 < 17
3. 14 < 7 && 6 <= 6
4. 4 > 3 && 17 >= 7

5. ! true
6. ! (13 == 7)
7. 9 != 7 && ! 1
8. 6 < && 8

Answers:

1. 0
2. 1
3. 0
4. 1
5. 0
6. 1
7. 0
8. Error, there needs to be an operand between the operators < and &&.

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 12 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_12 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_12a.cpp

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Compile and run the Lab_12a.cpp source code file. Understand how it works.
- Copy the source code file Lab_12a.cpp naming it: Lab_12b.cpp
- Convert the nested if then else to a switch with breaks.
- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 12a – Instructions

Flowchart the following pseudocode:
pseudocode

```
Case of shoe_size
  4 to 6  Display "Small."
  7 to 9  Display "Medium."
  10 +    Display "Large."
```

```
Endcase
```

Problem 12b – Instructions

The “Flip-Flops” is a unique shoe store that only sells flip-flops. Adult shoe sizes less than 4 are handled in the children’s department, thus we don’t need to concern ourselves with sizes less than 4. Half shoe sizes are to be rounded down, thus the prompt to the user that happens before this case structure will have addressed that issue. The variable shoe_size will be an integer value between 4 and 1,000,000,000 (one billion).

Write C++ source code for the following pseudocode:
pseudocode

```
Case of shoe_size
  4 to 6  Display "Small."
  7 to 9  Display "Medium."
  10 +    Display "Large."
Endcase
```

Problem 12c – Instructions

Write C++ source code for the following pseudocode:
pseudocode

```
If age equal to 24
  Display a message "You’re the same age as Melinda."
Else
  If age equal to 27
    Display a message "You’re the same age as Ruth."
  Else
    If age equal to 34
      Display a message "You’re the same age as Ben."
    Else
```

```
      Display a message "You're age is un-important."
    Endif
  Endif
Endif
```

REFERENCES

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

PART XII

TEST AFTER LOOPS

- Do While Loop
- Flag Concept
- Assignment vs Equality within C++
- Repeat Until Loop
- Practice

Do While Loop

KENNETH LEROY BUSBEE

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 the: do while.

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 every language that I know 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.

The do while Structure within C++

Syntax

The syntax for the do while control structure within the C++ programming language is:

```
do
  {
  statement;
  statement;
  statement;
  statement;    // This statement updates the flag;
  }
```

```
while (expression);
```

The test expression is within the parentheses, but this is not a function call. The parentheses are part of the control structure. Additionally, there is a semicolon after the parenthesis following the expression.

An Example

C++ source code: do while loop

```
do
  {
  cout << "\nWhat is your age? ";
  cin >> age_user;
  cout << "\nWhat is your friend's age? ";
  cin >> age_friend;
  cout >> "\nTogether your ages add up to: ";
  cout >> (age_user + age_friend);
  cout << "\nDo you want to do it again? y or n ";
  cin >> 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 normal industry practice within the C++ community.

Infinite Loops

At this point it's worth mentioning that good programming always provides for a method to insure 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:
C++ source code: infinite loop

```
loop_response = 'y';
do
  {
  cout << "\nWhat is your age? ";
  cin >> age_user;
  cout << "\nWhat is your friend's age? ";
  cin >> age_friend;
  cout >> "\nTogether your ages add up to: ";
  cout >> (age_user + age_friend);
  }
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:
C++ source code: infinite loop

```
do
```

```
  {
  cout << "\nWhat is your age? ";
  cin >> age_user;
  cout << "\nWhat is your friend's age? ";
  cin >> age_friend;
  cout >> "\nTogether your ages add up to: ";
  cout >> (age_user + age_friend);
  cout << "\nDo you want to do it again? y or n ";
  cin >> 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 within the Boolean concepts of the C++ programming language, 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 available in C++.
infinite loop
No method of exit, thus a bad thing.

test item

An attribute of iteration control structures.

update item
An attribute of iteration control structures.

REFERENCES

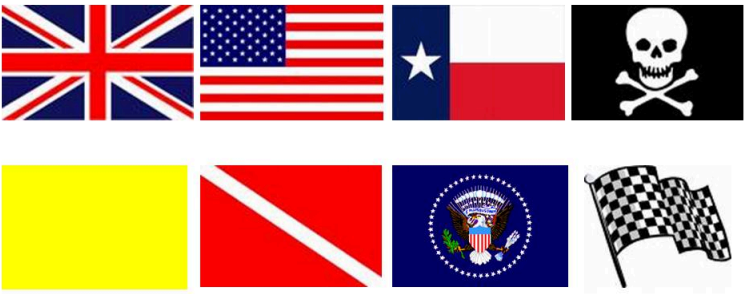
- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Flag Concept

KENNETH LEROY BUSBEE

Concept 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 cross bones. 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.



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 flag pole. 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 normally 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 controling 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 his 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 or constant used to store information that will normally be used to control the program.

REFERENCES

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

Assignment vs Equality within C++

KENNETH LEROY BUSBEE

General 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 expressionthat 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 C++ every data type can be used as a Boolean expression, because every data type can be demoted into a Boolean value by using the rule/concept that zero represents false and all non-zero values represent true.

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

```
if (pig = 'y')
  {
  cout << "\nPigs are good";
  }
else
```

```
  {
  cout << "\nPigs 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. Let's consider:

```
do
  {
  cout << "\nPigs are good";
  cout << "\nDo it again, answer y or n: ";
  cin >> do_it_again
  }
while (do_it_again = 'y');
```

The loop's test expression will always be true, because the expression is an assignment (not the relational operator of ==). It assigns the 'y' to the variable do_it_again, then looks at the value in do_it_again and determines that it is not zero; therefore the expression is true. And it will always be true and you have just created an infinite loop. As a reminder, infinite loops are not a good thing.

These examples are to remind you that you must be careful in creating your test expressions so that they are indeed a question usually involving the relational operators.

Don't get caught using assignment for equality.

REFERENCES

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

Repeat Until Loop

KENNETH LEROY BUSBEE

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 the: repeat until.

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 every language that I know 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.

The repeat until Structure within C++

Well, it just does not exist. Most programming languages have either the do while or the repeat until control structures, but not both.

KEY TERMS

repeat until
A test after iteration control structure that is not available in C++.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Identify which selection control structures are test after iteration.
3. Be able to write pseudo code or flowcharting for the do while control structure.
4. Be able to write C++ source code for a do while control structure.

REVIEW QUESTIONS

Answer the following statements as either true or 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.

Answers:

1. false
2. true

3. false
4. false
5. false

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 13 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_13 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_13_Pseudocode.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Create a source code file from the Lab_13_Pseudocode.txt

file. Name it: Lab_13.cpp
- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 13a – Instructions

Flowchart the following pseudocode:
pseudocode

```
Do
  Display "I like cheese cake!"
  Display "Do it again? y or n ---> "
  Get answer from keyboard
While answer is 'y'
```

REFERENCES

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

PART XIII

TEST BEFORE LOOPS

- Increment and Decrement Operators
- While Loop
- Practice

Increment and Decrement Operators

KENNETH LEROY BUSBEE

General Discussion

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

```
increment the counter
```

The concept is:

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

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

Operator symbols and/or names vary with different programming languages. The C++ programming language operators with their meanings are:

C++ Operator	Meaning
++	increment, **two plus signs**
—	decrement, **two minus signs**

C++ Code Examples

Basic Concept

Within the C++ programming language the increment and decrement are often used in this simple generic way. The operator of increment is represented by two plus signs in a row. Examples:

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

As C++ statements, the four examples all do the same thing. They add 1 to the value of whatever is stored in counter. The decrement opereator 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). Within C++ 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; // variable set up with initialization
  then later on in the code
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 within the C++ programming language.

Prefix Increment

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

```
int oldest = 44; // variable set up with initialization
  then later on in the code
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) where as 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. C++ however, expands this not only to all of the integer family but also to the floating-point family (float and double). Incrementing 3.87 will change the value to 4.87. Decrementing 'C' will change the value to 'B'. Remember the ASCII character values are really one byte unsigned integers (domain from 0 to 255).

Exercises

Evaluate the following items using increment or decrement:

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

KEY TERMS

decrement
Subtracting one from the value of a variable.

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

REFERENCES

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

While Loop

KENNETH LEROY BUSBEE

Introduction to Test Before Loops

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

Understanding Iteration in General – while

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

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

In almost all 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 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).

The while Structure within C++

Syntax

The syntax for the while control structure within the C++ programming language is:

```
statement;         // This statement initializes the flag;
while (expression)
  {
  statement;
  statement;
  statement;
  statement;     // This statement updates the flag;
  }
```

The test expression is within the parentheses, but this is not a function call. The parentheses are part of the control structure. Additionally, there is not a semicolon after the parenthesis following the expression.

An Example

C++ source code: while

```
loop_response = 'y';
while (loop_response == 'y')
  {
```

```
cout << "\nWhat is your age? ";
cin >> age_user;
cout << "\nWhat is your friend's age? ";
cin >> age_friend;
cout >> "\nTogether your ages add up to: ";
cout >> (age_user + age_friend);
cout << "\nDo you want to do it again? y or n ";
cin >> loop_response;
}
```

The four attributes of a test before loop are present. The initialization of the flag. The test is the equality relational comparison of the value in the flag variable to the lower case character of y. 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.

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 normal industry practice within the C++ community.

Infinite Loops

At this point it's worth mentioning that good programming always provides for a method to insure 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:
C++ source code: infinite loop

```
loop_response = 'y';
while (loop_response == 'y')
  {
  cout << "\nWhat is your age? ";
  cin >> age_user;
  cout << "\nWhat is your friend's age? ";
  cin >> age_friend;
  cout >> "\nTogether your ages add up to: ";
  cout >> (age_user + age_friend);
  }
```

The programmer assigned a value to the flag before the loop which is correct. However, he 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:
C++ source code: infinite loop

```
loop_response = 'y';
while (loop_response = 'y')
  {
  cout << "\nWhat is your age? ";
  cin >> age_user;
  cout << "\nWhat is your friend's age? ";
  cin >> age_friend;
  cout >> "\nTogether your ages add up to: ";
  cout >> (age_user + age_friend);
  cout << "\nDo you want to do it again? y or n ";
  cin >> 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 within the Boolean concepts of the C++ programming language, the answer to the test expression is true. Viola, you have an infinite loop.

C++ source code: infinite loop

```
loop_response = 'y';
while (loop_response == 'y');
  {
  cout << "\nWhat is your age? ";
  cin >> age_user;
  cout << "\nWhat is your friend's age? ";
  cin >> age_friend;
  cout >> "\nTogether your ages add up to: ";
  cout >> (age_user + age_friend);
  cout << "\nDo you want to do it again? y or n ";
  cin >> loop_response;
  }
```

The undesirable semi-colon on the end of while line causes the action of the while loop to be the “nothingness” between the closing parenthesis and the semi-colon. The program will infinitely loop because there is no action (that is no action and no update). If this is the first item in your program it will appear to start but there will be no output.

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:

C++ source code: while loop that is counting

```
counter = 0;
while (counter < 5)
  {
  cout << "\nI love ice cream!";
  counter++;
  }
```

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 my 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:
C++ source code: infinite loop

```
counter = 0;
while (counter < 5)
  {
  cout << "\nI 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.
C++ source code: while as a counting loop

```
while (0 < age)
  {
  cout << "\nI love candy!";
  age--;
  }
```

Consider the following variation assuming that age and counter are both integer data type and that age has a value:
C++ source code: while as a counting loop

```
counter = 0;
while (counter < age)
```

```
{
cout << "\nI love corn chips!";
counter++;
}
```

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.

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 available in C++.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Identify which selection control structures are test before iteration.
3. Be able to write pseudo code or flowcharting for the while control structure.
4. Be able to write C++ source code for the while control structure.

REVIEW QUESTIONS

Evaluate the following items using increment or decrement:

1. True or false: `x = x - 1;` and `x -= 1;` and

   ```
   x--;
   ```

   and

   ```
   --x;
   ```

   all accomplish decrement.
2. Given: `int y = 26;` and `int z;` what values will y and z have after: `z = y++;`
3. Given: `double x = 4.44;` and `int y;` what values will x and y have after:

```
y = --x;
```

4. As an expression: `10 / ++(money * 4)` Is this ok? Why or why not?

Answers:

1. true
2. y is: 27 and z is: 26
3. x is: 3.44 and y is: 3 Note: truncation of 3.44 to 3 upon demotion to integer data type.
4. Not ok. Error, the item incremented must have Lvalue attributes, usually a variable. Because of the parentheses, it is an expression not a variable.

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 14 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_14 within the folder named: Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_14_Pseudocode.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Create a source code file from the Lab_14_Pseudocode.txt file. Name it: Lab_14.cpp
- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 14a – Instructions

Flowchart the following pseudocode:
pseudocode

```
Assign counter a value of zero
While counter is less than 5
  Display "I love cookies!"
  Increment counter
Endwhile
```

REFERENCES

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

PART XIV

COUNTING LOOPS

- For Loop
- Circular Nature of the Integer Data Type Family
- Formatting Output
- Nested For Loops
- Practice

For Loop

KENNETH LEROY BUSBEE

Introduction to Test Before Loops

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

Understanding Iteration in General – for

In most 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.

The for Structure within C++

Syntax

The syntax of the for loop control structure within the C++ programming language is:

```
for (initializations; expression; updates)
  {
  statement;
  statement;
  statement;
  }
```

The initializations, test expression and updates are within the parentheses (each separated by a semi-colon), but this is not a function call. The parentheses are part of the control structure. Additionally, there is not a semicolon after the parenthesis following the expression.

An Example

C++ source code: for

```
for (counter = 0; counter < 5; counter++)
  {
  cout << "\nI 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 normal industry practice within the C++ community.

Infinite Loops

At this point it's worth mentioning that good programming always provides for a method to insure 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:
C++ source code: infinite loop

```
for (counter = 0; counter < 5;)
  {
  cout << "\nI love ice cream!";
  }
```

The programmer assigned a value to the flag during the initialization step which is correct. However, he 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).

Multiple Items in the Initialization and Update

The following shows the use of the sequence operator to separate the multiple initializations and multiple updates. This is not available in most languages, thus is more unique to the C++ programming language.
C++ source code: for with multiple initializations and updates

```
for (x = 0, y = 10; x < 10; x++, y--)
  {
  cout << x * y << endl;
  }
```

Counting Loop Conversion – a while into a for

Below is a color coded the conversion of a while loop that displays a message exactly three times (which is a counting loop) into a for loop using C++ programming language syntax. The four loop attributes are color highlighted as follows:

```
blue is the initialize
orange is the test
green is the action
red is the update

counter = 0;
while (counter < 3)
 {
 cout << "\nPlease be careful driving in Houston.";
 counter++;
 }

for (counter = 0; counter < 3; counter++)
 {
 cout << "\ nPlease be careful driving in Houston.";
 }
```

Miscellaneous Information about the for Structure

Many languages (Pascal, FORTRAN, and other) have a for loop structure that is used exclusively for counting. The for loop in the C++ programming language is much more versatile and can be used (and generally is used) in place of the while loop structure. In reality a counting loop is just a particular use of a while loop.

The name for comes from mathematics' method of writing an iteration (or repetition). In math we would say: "For the variable i

starts at a given value and repeats an action increasing the value of i until i is executed for the stopping value". Usually written in math as:

for i = 1 to 5 do some action

Note: here the = means equals not assignment. Another way to say it is that i varies from 1 to 5.

KEY TERMS

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

REFERENCES

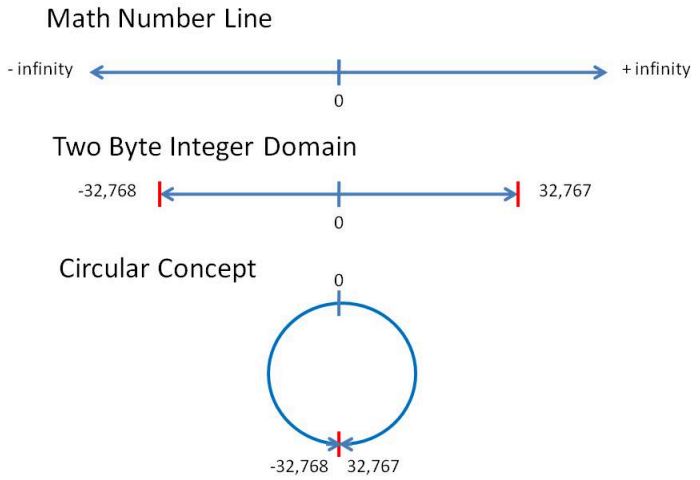
- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Circular Nature of the Integer Data Type Family

KENNETH LEROY BUSBEE

General Discussion

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

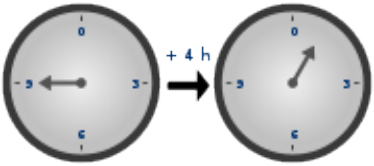


This circular nature of the integer family works for both integer and character data types. In theory, it should work for the Boolean data type as well; but in most programming languages it does not for various technical reasons.

"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. Usual 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).
C++ source code

```
for (int x = 0; x < 10; x--)
  {
```

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

The above code accidently 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.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Circular_Nature_Integer.cpp

KEY TERMS

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

REFERENCES

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

Formatting Output

KENNETH LEROY BUSBEE

General Discussion

Formatting of output is handled in different ways in the various languages used today. Many programming languages have different formatting instructions for the standard output device which is usually the monitor (going to a DOS black screen output box) versus using the monitor as a Graphical User Interface (GUI). File storage output is often handled similarly to the standard output device. All of this makes formatting of output very machine, output device and language dependent.

When teaching programming fundamentals, many professors prefer to use the standard output device. For the C++ programming language this means going to the monitor using a DOS black screen output box.

C++ Considerations using Standard Output (cout)

Text Wrapping and Vertical Spacing

There are two items used to keep output from filling up a line and **wrapping** on to the next line. They are:

- Using the escape code sequence of \n within your strings (text between as set of double quote marks).
- Using the item from the iostream named: endl; which is short for end line.

Thus the programmer is responsible for making text show

reasonably on the screen. Both of the above also allow for adequate vertical spacing when needed in your output.

Handling Floating-point Data Type

It is nice to have your output displayed so humans can read it (most humans are not use to scientific notation). There are three lines often inserted near the start of your code (first items in the function main) that can be used to direct the formatting of floating-point data. They are:

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(n);
```

They do the following for the rest of your program:

- fixed – Do not use scientific notation but show floating-point values like integer values (numeral digits of 0 to 9 – no exponent notation).
- showpoint – Always show a decimal point for floating-point values even if there is no fractional part.
- precision – Always show this number of digits (change n to a number like 2) to the right of the decimal point.

Setting the Width for Numbers

Setting the width for integer family and floating-point family data types must be done for the output of each value. Assume in the following example that age is an integer data type and money is a floating-point data type.

```
cout << setw(4) << age << endl;
cout << setw(8) << money << endl;
```

Note that each value had to have its own setw(n) where n is an integer number telling it how many positions to use for the

output. The iomanip header file (immediately shown) will need to be included in your program.

```
#include<iomanip> // needed for the setw
```

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Formatting_Output.cpp

KEY TERMS

formatting
Modifying the way the output is displayed.
wrapping
When output is not vertically spaced properly.

REFERENCES

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

Nested For Loops

KENNETH LEROY BUSBEE

General 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 12 by 12 multiplication table:

```
        1 |   2 |   3 |   4 |   5 |   6 |   7 |   8 |   9 |  10
    ------------------------------------------------------------
 1 !   1 |   2 |   3 |   4 |   5 |   6 |   7 |   8 |   9 |  10
 2 !   2 |   4 |   6 |   8 |  10 |  12 |  14 |  16 |  18 |  20
 3 !   3 |   6 |   9 |  12 |  15 |  18 |  21 |  24 |  27 |  30
 4 !   4 |   8 |  12 |  16 |  20 |  24 |  28 |  32 |  36 |  40
 5 !   5 |  10 |  15 |  20 |  25 |  30 |  35 |  40 |  45 |  50
 6 !   6 |  12 |  18 |  24 |  30 |  36 |  42 |  48 |  54 |  60
 7 !   7 |  14 |  21 |  28 |  35 |  42 |  49 |  56 |  63 |  70
 8 !   8 |  16 |  24 |  32 |  40 |  48 |  56 |  64 |  72 |  80
 9 !   9 |  18 |  27 |  36 |  45 |  54 |  63 |  72 |  81 |  90
10 !  10 |  20 |  30 |  40 |  50 |  60 |  70 |  80 |  90 | 100
11 !  11 |  22 |  33 |  44 |  55 |  66 |  77 |  88 |  99 | 110
12 !  12 |  24 |  36 |  48 |  60 |  72 |  84 |  96 | 108 | 120
```

We might also see that the answers could be designed as a collection of cells (each cell being exactly six spaces wide). The C++ source code to produce the above is:
C++ source code: nested for loops – multiplication table

```
cout << "      ";
for(across=1; across <13; across++)
  {
  cout << setw(4) << across << " |";
  }
cout << endl;

cout << "      ";
for(across=1; across <13; across++)
  {
  cout << "------";
  }
cout << endl;

for(down=1; down <13; down++)
  {
  cout << setw(4) << down << " !";
  for(across=1; across <13; across++)
    {
    cout << setw(4) << down*across << " |";
    }
  cout << endl;
  }
```

```
cout << "      ";
for(across=1; across<13; across++)
  {
  cout << setw(4) << across << " |";
  }
cout << endl;

cout << "      ";
for(across=1; across<13; across++)
  {
  cout << "------";
  }
cout << endl;

for(down=1; down<13; down++)
  {
  cout << setw(4) << down << " !";
  for(across=1; across<13; across++)
    {
    cout << setw(4) << down*across << " |";
    }
  cout << endl;
  }
```

Colorized Code – multiplication table

```
     1 |   2 |   3 |   4 |   5 |   6 |   7 |   8 |   9 |  10 |  11 |  12 |
-------------------------------------------------------------------------
 1 !   1 |   2 |   3 |   4 |   5 |   6 |   7 |   8 |   9 |  10 |  11 |  12 |
 2 !   2 |   4 |   6 |   8 |  10 |  12 |  14 |  16 |  18 |  20 |  22 |  24 |
 3 !   3 |   6 |   9 |  12 |  15 |  18 |  21 |  24 |  27 |  30 |  33 |  36 |
 4 !   4 |   8 |  12 |  16 |  20 |  24 |  28 |  32 |  36 |  40 |  44 |  48 |
 5 !   5 |  10 |  15 |  20 |  25 |  30 |  35 |  40 |  45 |  50 |  55 |  60 |
 6 !   6 |  12 |  18 |  24 |  30 |  36 |  42 |  48 |  54 |  60 |  66 |  72 |
 7 !   7 |  14 |  21 |  28 |  35 |  42 |  49 |  56 |  63 |  70 |  77 |  84 |
 8 !   8 |  16 |  24 |  32 |  40 |  48 |  56 |  64 |  72 |  80 |  88 |  96 |
 9 !   9 |  18 |  27 |  36 |  45 |  54 |  63 |  72 |  81 |  90 |  99 | 108 |
10 !  10 |  20 |  30 |  40 |  50 |  60 |  70 |  80 |  90 | 100 | 110 | 120 |
11 !  11 |  22 |  33 |  44 |  55 |  66 |  77 |  88 |  99 | 110 | 121 | 132 |
12 !  12 |  24 |  36 |  48 |  60 |  72 |  84 |  96 | 108 | 120 | 132 | 144 |
```

Colorized Output – multiplication table

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Nested_For_Loops.cpp

KEY TERMS

complex logic
Often solved with nested control structures.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Identify which selection control structures are commonly used a counting loops.
3. Be able to write pseudo code or flowcharting for the for control structure.
4. Be able to write C++ source code for a for control structure.
5. When feasible, be able to convert C++ source code from while loop acting like a counting loop to a for loop and and vice versa.

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. Only for loops can be counting loops.
2. The integer data type has modular arithmetic attributes.
3. The escape code of \n is part of formatting output.
4. Nested for loops is not allowed in the C++ programming language.
5. Counting loops use all four of the loop attributes.

Answers:

1. false
2. true

3. true
4. false
5. true

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 15 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_15 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_15a.cpp

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Compile and run the Lab_15a.cpp source code file.

Understand how it works.

- Copy the source code file Lab_15a.cpp naming it: Lab_15b.cpp
- Convert the code that is counting (all four attributes) to a for loop.
- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 15a – Instructions

Using proper C++ syntax, convert the following for loop to a while loop.
C++ source code

```
for (x = 0; x < 10; x++)
  {
  cout << "Having fun!";
  }
```

REFERENCES

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

PART XV

STRING CLASS, UNARY POSITIVE AND NEGATIVE

- String Class within C++
- Unary Positive and Negative Operators
- Practice

String Class within C++

KENNETH LEROY BUSBEE

General Discussion

In most programming languages a string is typically a string of characters (string them along in a series). The rules for handling strings vary from language to language. Technically, there is no string data type in the C++ programming language. However, the concept of a string data type makes it easy to handle stings of character data. Associated with object oriented programming the string class has been added to C++ as a standard part of the programming language.

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 first most students encounter. An array is a sequenced collection of elements of the same data type with a single identifier name. This definition perfectly describes our string data type concept. The simplest array is called a one-dimensional array; also know as a list because we usually list the members or elements vertically. However, strings are viewed as a one-dimensional array that visualize as listed horizontally. Strings are an array of character data.

In the "C" programming language all strings were handled as an array of characters that end in an ASCII null character (the value 0 or the first character in the ASCII character code set). This changed with the implementation of the string class within C++ where strings are stored as a length controlled array with a maximum length of 255 characters. This string class implementation also allowed programmers to use the reserved

word string as if it were a data type. Commonly used operators and some alternatives for the string class are summarized in the following table:

C++ Operator	Operator Name	String Class Implementation
=	assignment	Same as for standard data types
<, >, <=, >=, ==, !=	six relational	Same as for standard data types
+	addition	Concatenation or Append
sizeof	Usage how many bytes a data type occupies	Implemented using a class member function named length. Format: identifier_name.length() NOTE: The period between the identifier name and the function name is the class member operator.
. **the period**	class member	Used in conjunction with class functions

Most other operators are not allowed and basically do not make sense for a string data type. The above items are demonstrated in the following program.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_String_Class.cpp

KEY TERMS

array
A sequenced collection of elements of the same data type with a single identifier name.
class member
An operator used to invoke functions associated with a class.

concatenation
Combining two strings into one string.

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

REFERENCES

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

Unary Positive and Negative Operators

KENNETH LEROY BUSBEE

General Discussion

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

```
+5 + -2
```

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

```
-2 - +5
```

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

```
7 - -2
```

First constants that do not have a unary minis 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.

C++ Code Examples

The above examples work within the C++ programming language. What happens if we put a unary positive or unary negative in front of a variable or a named constant?

Negation – Unary Negative

The concept of negation is to take a value and change its sign, that is: flip it. If it 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

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Identify which operators are allowed with the string class.
3. Understand the unary positive and unary negative operators.

REVIEW QUESTIONS

Evaluate the following items involving unary positive and unary negative:

1. +13 – -2
2. -10 + 14
3. 4 + – 3
4. +8 – * +5

Answers:

1. 15
2. 4
3. 1
4. Error, no operand between subtraction and multiplication.

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 16 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_16 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_16_Pseudocode.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Create a source code file from the Lab_16_Pseudocode.txt file. Name it: Lab_16.cpp
- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions

from your professor/instructor for submitting it for grading.

Problems

Problem 16a – Instructions

Describe the normal C++ operators allowed with the string data type.

Problem 16b – Instructions

Describe why unary positive is worthless.

Problem 16c – Instructions

Describe how unary negative works.

REFERENCES

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

PART XVI

CONDITIONAL OPERATOR AND RECURSION

- Conditional Operator
- Recursion vs Iteration
- Practice

Conditional Operator

KENNETH LEROY BUSBEE

Overview

The conditional operator is unique in that it has three operands separated by two unconnected operator symbols. All other C++ operators are either unary (one operator and one operand) or binary (one operator and two operands). On the "Abbreviated Precedence Chart for C++ Operators" the conditional operator has the word "trinary" in the comments column. This prefix "tri" means three, thus three operands.

C++ Operator	Meaning	Comments
? :	conditional	trinary – three operands with two operators

As an operator it produces a value for the expression. An easy way to explain the conditional operator is to convert an "if then else" control structure to an expression using the conditional operator.
if then else

```
if (age > 17)
  {
  cout << "You can vote.";
  }
else
  {
  cout << "You can't vote.";
  }
```

conditional = option 1

```
age > 17 ? cout << "You can vote." : cout << "You can't vote.";
```

conditional = option 2

```
cout << (age > 17 ? "You can vote." : "You can't vote.");
```

The use of parenthesizes is needed because of the precedence of operators. The conditional expression is of lower precedence than the insertion (writing) operator.

The first operand is a **test expression** similar to those that control program flow in control structures. This type of expression is also known as a **Boolean expression** because they create a Boolean answer of true or false. If the test is true the second operand becomes the value of the expression. If false, the third operand becomes the value of the expression. The operators of the question mark and colon separate the three operands.
general format

```
test expression ? expression true : expression false
```

KEY TERMS

conditional
A trinary C++ operator that acts like an if then else control structure.

REFERENCES

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

Recursion vs Iteration

KENNETH LEROY BUSBEE

Repetitive Algorithms

"In general, there are two approaches to writing repetitive algorithms. One uses loops; the other uses recursion. **Recursion** is a repetitive process in which a function calls itself. Both approaches provide repetition, and either can be converted to the other's approach."[1] Iteration is one of the categories of control structures. It allows for the processing of some action zero to many times. Iteration is also known as looping and repetition. The math term "to iterate" means to perform the statement parts of the loop. Many problems/tasks require the use of repetitive algorithms. With most programming languages this can be done with either:

1. looping control structures, specifically the for loop (an iterative approach)
2. recursive calling of a function

Using repetitive algorithms as the solution method occurs in many mathematical oriented problems. These in include factorial, Fibonacci numbers, and the Towers of Hanoi problem. Solutions to these problems are often only presented in terms of using the recursive method. However, "... you should understand the two major limitations of recursion. First, recursive solutions may involve extensive overhead because they use function calls. Second, each time you make a call you use up some of your memory allocation. If the recursion is deep – that is, if there is a large number of recursive calls – then you may run out of

memory. Both the factorial and Fibonacci numbers solutions are better developed iteratively."[2]

Understanding how recursion or the iterative approaches work will be left to others. They are usually covered in detail as part of studying data structures. Our goal in covering them is to:

1. Provide you with a definition of recursion
2. Introduce the alternate solution approach of iteration

The following demonstration program shows both solutions for 8! (eight factorial).

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Factorial.cpp

KEY TERMS

factorial
A math problem that often is solved using recursion.

recursion
A repetitive process in which a function calls itself.

Footnotes

- 1 Behrouz A. Forouzan and Richard F. Gilberg, Computer Science A Structured Approach using C++ Second Edition (United States of America: Thompson – Brooks/Cole, 2004) 265.
- 2 Behrouz A. Forouzan and Richard F. Gilberg, Computer Science A Structured Approach using C++ Second Edition (United States of America: Thompson – Brooks/Cole, 2004) 272.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Understand the conditional operator and how it works.
3. Understand recursion as a problem solving technique.
4. When feasible, be able to convert C++ source code from a conditional expression to an if then else and vice versa.

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. The conditional expression acts like a case structure.
2. The conditional operator is a two part operator with three operands.
3. Recursion is one method of implementing a repetitive algorithm.
4. Recursion is always preferred over an iterative approach to a repetitive problem.
5. Factorial is usually demonstrated with an iterative approach.

Answers:

1. false
2. true
3. true

4. false
5. false

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 17 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_17 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_17a.cpp

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Compile and run the Lab_17a.cpp source code file.
  Understand how it works.

- Copy the source code file Lab_17a.cpp naming it: Lab_17b.cpp
- Convert the code that is using the if then else to a conditional expression.
- Convert the code that is using the conditional expression to an if then else.
- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 17a – Instructions

Using proper C++ syntax, convert the following if then else to a conditional expression.
if then else

```
if (x == y)
  {
  z = 14;
  }
else
  {
  z++;
  }
```

Problem 17b – Instructions

Using proper C++ syntax, convert the following conditional expression to an if then else.
conditional

```
answer = y < z ? 47 : 92;
```

REFERENCES

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

PART XVII

INTRODUCTION TO ARRAYS

- Array Data Type
- Array Index Operator
- Displaying Array Members
- Practice

Array Data Type

KENNETH LEROY BUSBEE

Overview

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). This module will only cover the single dimension array. Most single dimension arrays are visualized vertically and are often called a list.

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. Within C++ the string data type is a length-controlled array and is a pre-defined data class.

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. Additionally, the **sizeof** operator is often used to calculate the number of members in an array.

Defining an Array in C++

Example:

```
int ages[5] = {49,48,26,19,16};
```

This is the **defining of storage space**. The square brackets (left [and right]) 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 done it this way:

```
int ages[] = {49,48,26,19,16};
```

By leaving out the five and having initial values assigned, the compiler will know to create the array with five storage spaces because there are five values listed. This method is preferred because we can simply add members to or remove members from the array by changing the items inside of the braces. We could have also done this:

```
int ages[5];
```

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 (actually there would be values there but we don't know what they would be and thus think of the values as garbage). We could assign values later in our program by doing this:

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

The members of the array go from 0 to 4; **NOT** 1 to 5. This is

explained in more detail in another Connexions module that covers accessing array members and is listed in the supplemental links provided. See: Array Index Operator.

KEY TERMS

dimension
An axis of an array.
list
A single dimension array.
table
A two dimension array.

REFERENCES

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

Array Index Operator

KENNETH LEROY BUSBEE

Array Index Operator in C++

Example:

```
int ages[5] = {49,48,26,19,16};
int my_age;
my_age = ages[2]
```

This second usage of the square brackets is as the **array notation of dereference** or more commonly called the **index operator**. As an operator it either provides the value held by the member of the array (Rvalue) or changes 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 variable named my_age. 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. The array members (or elements) are referenced starting at zero. The more common way for people to reference a list is by starting with one. Many programming languages reference array members starting at one, however for some languages (and C++ is one of them) you will need to **change your thinking**. Consider:

Position	C++	Miss America	Other Contests
zero offsets from the front	ages [0]	Winner	1st Place
one offsets 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 3rdPlace. 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.

The C++ operator name is called the array index or simply the index operator and it uses the square brackets as the operator symbols.

KEY TERMS

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

REFERENCES

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

Displaying Array Members

KENNETH LEROY BUSBEE

Accessing Array Members in C++

accessing the members of an array

```
int ages[] = {49,48,26,19,16};
int counter;

for (counter = 0, counter < 5, counter++)
  {
  cout << ages[counter] << endl;
  }
```

This second usage of the square brackets is as the **array notation of dereference** or more commonly called the **index operator**. As an operator it provides the value held by the member of the array. For example, during one of the iterations of the for loop the index (which is an integer data type) will have the value of 3. The expression ages[counter] would in essence be: ages[3]. The dereference operator of [3] means to go the 3rd offset from the front of the ages array and get the value stored there. In this case the value would be 19. The array members (or elements) are referenced starting at zero. The more common way for people to reference a list is by starting with one. Many programming languages reference array members starting at one, however for some languages (and C++ is one of them) you will need to **change your thinking**. Consider:

Position	C++	Miss America	Other Contests
zero offsets from the front	ages [0]	Winner	1st Place
one offsets 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 3rdPlace. We would be talking about the same position in the array of the five finalists.

Rather than using the for loop to display the members of the array, we could have written five lines of code as follows:

```
cout << ages[0] << endl;
cout << ages[1] << endl;
cout << ages[2] << endl;
cout << ages[3] << endl;
cout << ages[4] << endl;
```

Using the Sizeof Operator with Arrays in C++

using the sizeof operator

```
int ages[] = {49,48,26,19,16};
int counter;

for (counter = 0, counter < sizeof ages / sizeof ages[0], counte
  {
```

```
cout << ages[counter] << endl;
}
```

Within the control of the for loop for the displaying of the grades, note that we calculated the number of the members in the array by using the sizeof operator. The expression is:

```
sizeof ages / sizeof ages[0]
```

When you ask for the sizeof an array identifier name the answer is how many total bytes long is the array (or in other words – how many bytes of storage does this array need to store its values). This will depend on the data type of the array and the number of elements. When you ask for the sizeof one of its members, it tells you how many bytes one member needs. By dividing the total number of bytes by the size of one member, we get the answer we want: the number of members in the array. 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.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Arrays.cpp

KEY TERMS

flexible coding
Using the sizeof operator to calculate the number of members in an array.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Within C++ source code be able to define a single dimension array.
3. Within C++ source code be able to access array members using the index operator.
4. Within C++ source code be able to calculate the number of members in an array using the sizeof operator.

REVIEW QUESTIONS

Answer the following statements as either true or 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.

Answers:

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

5. false

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 18 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_18 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_18_Narrative_Description.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Create a source code file following the directions in the Lab_18_Narrative_Description.txt file. Name it: Lab_18.cpp
- Build (compile and run) your program.

- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 18a – Instructions

Briefly explain what an array is and list the two common operators used with arrays.

REFERENCES

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

PART XVIII

FILE I/O AND ARRAY FUNCTIONS

- File Input and Output
- Arrays and Functions
- Loading an Array from a File
- Math Statistics with Arrays
- Practice

File Input and Output

KENNETH LEROY BUSBEE

Overview of File I/O in C++

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 know an 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 (this 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 C++ to indicate a return character when used with in a literal string with the cout.

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, Pascal using .pas, Cobol using .cob, etc. Thus, most compiler/Integrated Development Environment software packages (such as the **Bloodshed Dev-C++ 5 compiler/IDE**) can be used to create ASCII text files.

Filename – The name and its extension. Most operating systems have restrictions on which characters can be used in filenames. Example for MS-DOS and Windows: 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.

Filespec – The location of a file along with its filename. It 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 for MS-DOS and Windows: 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. Within C++ this is accomplished by including the header file: <fstream> File Input/Output is handled in C++ by using a pre-defined class of data objects, similar to the way string data type is handled. This class of objects has both data type names and functions built to specifically accomplish opening and closing a file.

Within your program you create a local storage variable with the data type of fstream like this:

```
fstream inData;
```

This variable will be used to store the device token that the operating system assigns to the file being opened. Thus, opening a file uses a class member function call like this:

```
inData.open("C:\\myfiles\\cosc_1436\\Lab_05.txt",
ios::in);
```

The two parameters passed to the function are the filespec and the method that you want to use the file (in this example as input). The function provides a returning value of a **device token** from the operating system and it is stored in the variable named inData.

It is considered good programming practice to determine if the file was opened properly. The device token should be a non zero value. It the operating system gives you the value of zero

it was not able to open the file. The reason it 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. We often test the device token by using an if then control structure with the action consisting of stopping the program if it is true that you got the zero. The first line of the if then control structure looks like this:

```
if (!inData)
```

Don't be misled by the not operator. This reads "if it is true that the token stored in inData is zero". If inData is zero, noting zero is 1 or true.

Read – Moving data from a device that has been opened into a memory location defined in your program. When reading text files that have integer or floating-point constants, the operating systems converts the text symbols to a binary number. The operator used is the extraction or read operator. An example of reading is:

```
inData >> next_number
```

This expression is similar to reading from the standard input device (aka the keyboard):

```
cin >> next_number
```

The "cin" is a predefined device token associated with the Standard Input and Output devices. For our file reading example you might say, "Go to the device identified by the token stored in the inData variable and read in the next value storing it in the next_number variable within my program".

Write – Moving data from a memory location defined in your program to a device that has been opened. When writing integer or floating-point data types, the operating system converts the binary number into the proper text symbols. The operator used is the insertion or write operator. An example of writing is:

```
outData << "Total is: " << total << endl;
```

This expression is similar to writing to the standard output device (aka the monitor):

```
cout << "Total is: " << total << endl;
```

The “cout” is a predefined device token associated with the Standard Input and Output devices. For our file writing example you might say, “Go to the device identified by the token stored in the outData variable and write the items listed (the string constant then the value stored in my program variable named total then the endl or new line or the return character)”.

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 insure that all of the data is physically stored in the output file. Some examples of closing files:

```
inData.close();
outData.close();
```

You need to study this module in conjunction with the demo file provided.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials. You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Demo_File_IO.cpp

Download from Connexions: Demo_File_IO_Input.txt

After you run the program use a text editor to examine the **Demo_File_IO_Output.txt** file created by the program. You should see the output as: `Total is: 33.3`

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.

text file

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

write

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

REFERENCES

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

Arrays and Functions

KENNETH LEROY BUSBEE

Overview of Array Functions

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 of functions also pass another piece of data needed for that particular functions task.

This module covers the displaying the array members on the monitor via calling an **array function** dedicated to that task. You need to study this module in conjunction with the demo file provided.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials.

Download from Connexions: Demo_Array_Display_Function.cpp

KEY TERMS

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

REFERENCES

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

Loading an Array from a File

KENNETH LEROY BUSBEE

Conceptual Overview

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?

The solution involves some simple steps:

1. We can read the file once to get the element count. Thus, we will know exactly how many members (elements) we will need.
2. We can then create an array using dynamic memory allocation by defining the array within a function so that it has local scope. Local scope variables are created during the execution of the program and use the stack as the storage location instead of the data area. If you define the array outside of a function (global scope also known as static memory allocation) it stores it in the data area and must know how much storage space to allocate to the array when you **write the source code**. Since we don't know how many elements will be on the input file when we write the source code defining an array with global scope will not work. But, we can determine exactly how many members we need for the array by having our program count them (step 1) so that we can then define the array with local scope to the precise size needed.

3. We can then load the array by reading the file a second time and storing the values read into the array just created.

This method is demonstrated in the demo file provided, thus you need to study this material in conjunction with the demo program.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials. You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Demo_Loading_Array_from_File.cpp

Download from Connexions: Demo_Farm_Acres_Input.txt

KEY TERMS

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

REFERENCES

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

Math Statistics with Arrays

KENNETH LEROY BUSBEE

Overview

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 of 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**. The **array function** is often called "sum" and it does some parameter passing. It passes into the function the common two items of the array: its name along with the number of elements; but it also returns a value representing sum or total of the values within the array. You need to study this module in conjunction with the demo file provided.

Other mathematical functions often associated with statistics such as: average, count, minimum, maximum, standard deviation, etc. are often developed for processing arrays.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials. You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Demo_Sum_ Array_Function.cpp
Download from Connexions: Demo_Farm_Acres_Input.txt

KEY TERMS

sum
Latin for summa or a total.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Within C++ source code be able to understand basic file input and file output.
3. Understand why we test to see if a file was opened properly.
4. Understand why we close a file when we are done with it.
5. Within C++ source code be able to understand functions for arrays, specifically counting the number of elements in a file so you can define an array, load that array with those elements, display the array and sum the array.
6. Within C++ source code be able to create functions for arrays, specifically a function for averaging.

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. Text files are hard to create.
2. A filespec refers to a very small (like a spec dust) file.
3. 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.
4. Programmers should not worry about closing a file.
5. Where you define an item, that is global or local scope, is rarely important.

Answers:

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

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 19 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_19 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_19_Narrative_Description.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Create a source code file following the directions in the Lab_19_Narrative_Description.txt file. Name it: Lab_19.cpp
- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 19a – Instructions

For what purpose do we use the sizeof operator with an array.

Problem 19b – Instructions

Why would we open a file and count its elements and then close the file.

REFERENCES

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

PART XIX

MORE ARRAY FUNCTIONS

- Finding a Specific Member of an Array
- Sorting an Array
- Practice

Finding a Specific Member of an Array

KENNETH LEROY BUSBEE

Overview

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 algorithm. Two specific searches can be made for the maximum (largest) values in the array or the minimum (smallest) value in the array. Maximum and minimum are also know as max and min.

There are two basic ways of searching for a specific value:

1. Linear search
2. Binary search

"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 search 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."[1]

Binary search is not cover in this module. Linear search and searching for the maximum is demonstrated in the demo file provided, thus you need to study this material in conjunction with the demo program.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials. You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Demo_Finding Array Member.cpp

Download from Connexions: Demo_Farm_Acres_Input.txt

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.

Footnotes

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

REFERENCES

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

Sorting an Array

KENNETH LEROY BUSBEE

Overview

Sorting is the process through which data are arranged according to their values. There are several sorting algorithms or methods that can be used to sort data. Some include:

1. Bubble
2. Selection
3. Insertion

We will not be covering the selection or insertion sort methods in this module.

"The bubble sort is an easy way to arrange data in ascending or descending order. If an array is sorted in ascending order, it means the values in the array are stored from lowest to highest. If values are sorted in descending order, they are stored from highest to lowest. Bubble sort works by comparing each element with its neighbor and swapping them it they are not in the desired order."[1]

There are several different methods of bubble sorting and some methods are more efficient than others. Most use a pair of nested loops or iteration control structures. One method sets a flag that indicates that the array is sorted, then does a pass and if any elements are exchanged (switched); it sets the flag to indicate that the array is not sorted. It is executed until it makes a pass and nothing is exchanged.

This bubble sort sets a flag that indicates that the array is sorted (that is it does not need more sorting), then does a pass and if any elements are exchanged (switched); it sets the flag to indicate that the array is not sorted (that is it needs more sorting). The outer do while loop is executed until the inner for loop makes a pass and nothing is exchanged.

Here is some color highlighted C++ code from **Demo_Sort_Array_Function.cpp**

```
do
   {
   moresortneeded = false;
   for(int i = 0; i < array_size - 1; i++)
     {
     if(things[i] > things[i+1])
       {
       temp = things[i];
       things[i] = things[i+1];
       things[i+1] = temp;
       moresortneeded = true;
       }
     }
   }
while(moresortneeded);
```

The bubble sort gets its name from the lighter bubbles that move or "bubble up" to the top of a glass of soda pop. We move the smaller elements of the array to the top as the larger elements move to the bottom of the array. This can be viewed from a different perspective. Using an Italian salad dressing with oil, water and herbs; once shaken you can either:

1. envision the lighter oil rising to the top; **OR**
2. envision the heaver water and herbs sinking to the bottom

Either way is correct and this version of the code simply demonstrates the sinking to the bottom the heaver or larger elements of the array.

Bubble sorting is demonstrated in the demo file provided,

thus you need to study this material in conjunction with the demo program.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials. You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Demo_Sort_Array_Function.cpp
Download from Connexions: Demo_Farm_Acres_Input.txt

KEY TERMS

bubble sort
A method of swapping array members until they are in the

desired sequence.

sorting
Arranging data according to their values.

Footnotes

- 1 Tony Gaddis, Judy Walters and Godfrey Muganda, Starting Out with C++ Early Objects Sixth Edition (United States of America: Pearson – Addison Wesley, 2008) 569.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Within C++ source code be able to understand functions for arrays, specifically searching a array's values to see if a given value exists, finding the maximum value in an array and sorting an array.
3. Within C++ source code be able to create functions for arrays, specifically a function for finding the smallest value in an array.
4. Within C++ source code be able to modifying existing code to process different types of arrays.

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. Linear searches require complex algorithms.
2. Functions are often created for searching for the max and min values in an array.
3. The bubble sort is an easy way to arrange data an array.
4. There is only one method of bubble sorting.
5. Sorting an array is frequently done.

Answers:

1. false

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

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 20 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_20 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select “Save Target As” in order to download the file.

Download from Connexions: Lab_20_Narrative_Description.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Create a source code file following the directions in the Lab_20_Narrative_Description.txt file. Name it: Lab_20.cpp
- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 20a – Instructions

Give a short explanation of bubble sorting.

REFERENCES

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

PART XX

MORE ON TYPEDEF

- Versatile Code with Typedef
- Practice

Versatile Code with Typedef

KENNETH LEROY BUSBEE

Overview

Everyone seeks of ways to be more efficient in what they do. A farmer uses a tractor instead of a horse. A construction worker uses an air powered nail gun instead of a hammer. Programmers are no different than others, in that they are constantly improving their ability to produce correctly working programs. Some aspect of this is the use of modular/structured programming, proper documentation and following industry rules for a specific programming language. One example of efficient coding is letting the computer count the number of elements in an array. If we define an array:

```
int ages[] = {33,32,10,3};
```

We can use the following expression to calculate the number of members in the array:

```
sizeof ages / sizeof ages[0]
```

This type of **flexible coding** allows us to change the members of the array by adding or subtracting a values, like this:

```
int ages[] = {57,33,32,3,1};
```

Thus, we don't have to modify our code that uses the expression that calculates the number of member in the array.

One use of the **typedef** is to allow us to write code that can be quickly changed to handle different data types. There are several integer and floating-point data types that all store number values with different domains. If we write our code using some typedef statement, then our code becomes **versatile**. By

changing only our typedef commands, our code can be used to process data of a different data type. This is demonstrated within the demo file provided, thus you need to study this material in conjunction with the demo program.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials. You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Demo_Versatile_Array_Functions.cpp

Download from Connexions: Demo_Farm_Acres_Input.txt

Download from Connexions: Demo_Deposit_Checks_Input.txt

KEY TERMS

flexible coding
Using the sizeof operator to calculate the number of members in an array.
typedef
Allows the programmer to create an alias, or synonym, for an existing data type.
versatile
Easily modifying code to handle another data type.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Understand how typedef is used within C++ source code be able make the code versatile that is easy to change for different data types.

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. Most programmers rarely worry about efficiency.
2. Modular/structured programming helps improve efficiency.
3. Flexible coding helps improve efficiency.
4. Who cares about indentation and alignment within source code. It's a waste of time.
5. Versatile code is a concept that is easy to understand.

Answers:

1. false – Efficiency of code execution, no; efficiency of code production and maintenance, yes.
2. true
3. true
4. false
5. maybe true and maybe false – It does require some effort to catch on to how it works.

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 18 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_21 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_21_Narrative_Description.txt
Download from Connexions: Lab_21_Input.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Create a source code file following the directions in the Lab_21_Narrative_Description.txt file. Name it: Lab_21.cpp
- Build (compile and run) your program.
- After you have successfully written this program, if you are

taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 21a – Instructions

Explain the difference between flexible coding and versatile coding.

REFERENCES

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

PART XXI

POINTERS

- Address Operator
- Parameter Passing by Reference
- Pointer Data Type
- Indirection Operator
- Practice

Address Operator

KENNETH LEROY BUSBEE

Address Operator in C++

"Every variable is assigned a memory location whose address can be retrieved using the address operator &. The address of a memory location is called a pointer. Every variable in an executing program is allocated a section of memory large enough to hold a value of that variable's type."[1] Thus, whether the variables are global scope and use the data area for storage or local scopeand use the stack for storage; you can ask the question at what address in the memory does this variable exist. Given an integer variable named age:

```
int age = 47;
```

We can use the **address operator** [which is the ampersand or &] to determine where it exists (or its address) in the memory by:

```
&age
```

This expression is a **pointer** data type. The concept of an address and a pointer are one in the same. A pointer points to the location in memory because the value of a pointer is the address were the data item resides in the memory.

The address operator is commonly used in two ways:

1. To do parameter passing by reference
2. To establish the value of pointers

Both of these items are covered in the supplemental links to this module.

You can print out the value of the address with the following code:

```
cout << &age;
```

This will by default print the value in hexadecimal. Some people prefer an integer value and to print it as an integer you will need to cast the address into a long data type:

```
cout << long(&age);
```

One additional tidbit, an array's name is by definition a pointer to the arrays first element. Thus:

```
int iqs[] = {122, 105, 131, 97};
```

establishes "iqs" as a pointer to the array.

KEY TERMS

address operator
The ampersand or &.
pointer
A variable that holds an address as its value.

Footnotes

- 1 Tony Gaddis, Judy Walters and Godfrey Muganda, Starting Out with C++ Early Objects Sixth Edition (United States of America: Pearson – Addison Wesley, 2008) 597.

REFERENCES

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

Parameter Passing by Reference

KENNETH LEROY BUSBEE

Overview

When we pass parameters to functions we usually pass by value; that is the calling function provides several values to the called function as needed. The called function takes these values which have local scope and stores them on the stack using them as needed for whatever processing the functions accomplishes. This is the preferred method when calling user defined specific task functions. The called function passes back a single value as the return item if needed. This has the advantage of a closed communications model with everything being neatly passed in as values and any needed item returned back as a parameter.

By necessity there are two exceptions to this closed communications model:

1. When we need more than one item of information returned by the function
2. When a copy of an argument cannot reasonably or correctly be made (example: file stream objects).

These exceptions are handled by parameter passing by reference instead of passing a value. The item passed is called a **reference variable** and it represents a concept of an alias for the variable. Any change made to the reference variable is actually performed on the variable that it represents. The symbol

of the ampersand is used to designate the reference variable (and it is associated with the address operator).
parameter passing by reference

```
// prototype
void process_values(int qty_dimes, int qty_quarters, double &val

// variable definitions
int     dimes = 45;
int     quarters = 33;
double  value_dimes;
double  value_quarters;

// somewhere in the function main
process_values(dimes, quarters, value_dimes, value_quarters);

// definition of the function
void process_values(int qty_dimes, int qty_quarters, double &val
  {
  value_dimes = dimes * 0.10;
  value_quarters = quarters * 0.25;
  }
```

The ampersand must appear in both the prototype and the function definition but it does not appear in the function call.

The above example shows the basic mechanics of parameter passing by reference. You should study the demonstration program in conjunction with this module.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials. You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Demo_Parameter_Passing.cpp

KEY TERMS

reference variable
Used with parameter passing by reference.

REFERENCES

- cnx.org: Programming Fundamentals – A Modular

Pointer Data Type

KENNETH LEROY BUSBEE

Pointer Data Type in C++

A **pointer** variable is a variable that holds the address of a memory location. "Every variable is assigned a memory location whose address can be retrieved using the address operator &. The address of a memory location is called a pointer."[1] The **pointer data type** allows us to designate a variable to hold an address or a pointer. The concept of an address and a pointer are one in the same. A pointer points to the location in memory because the value of a pointer is the address were the data item resides in the memory. Given an integer variable named age:

```
int age = 47;
```

We can create a pointer variable and establish its value which would be the done using the address operator [which is the ampersand or &] by:

```
int * int_pointer = &age;
```

The asterisk is used to designate that the variable int_pointer is an integer pointer [int *]. This means that whenever we use the variable int_pointer that the compiler will know that it is a pointer that points to an integer.

In order to use pointers you will need to understand the indirection operator which is covered a supplemental link.

KEY TERMS

pointer
A variable that holds an address as its value.

Footnotes

- 1 Tony Gaddis, Judy Walters and Godfrey Muganda, Starting Out with C++ Early Objects Sixth Edition (United States of America: Pearson – Addison Wesley, 2008) 597.

REFERENCES

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

Indirection Operator

KENNETH LEROY BUSBEE

Indirection Operator in C++

When we pass parameters to functions we usually pass by value; that is the calling function provides several values to the called function as needed. The called function takes these values which have local scope and stores them on the stack using them as needed for whatever processing the functions accomplishes. This is the preferred method when calling user defined specific task functions. The called function passes back a single value as the return item if needed. This has the advantage of a closed communications model with everything being neatly passed in as values and any needed item returned back as a parameter.

By necessity there are two exceptions to this closed communications model:

1. When we need more than one item of information returned by the function
2. When a copy of an argument cannot reasonably or correctly be made (example: file stream objects).

These exceptions could be handled by parameter passing by reference instead of passing a value. Although different syntax than parameter passing when using a reference variable; using a pointer variable and the **indirection operator** can accomplish the same effect. The indirection operator is the asterisk or the character that we also use for multiplication. The concept of indirection is also known as **dereferencing**, meaning that we are

not interested in the pointer but want the item to which the address is referring or referencing.
parameter passing with pointers

```
// prototype
void process_values(int qty_dimes, int qty_quarters, double * pt

// variable definitions
int      dimes = 45;
int      quarters = 33;
double   value_dimes;
double   value_quarters;
double * ptr_value_dimes = &value_dimes;
double * ptr_value_quarters = &value_quarters;

// somewhere in the function main
process_values(dimes, quarters, ptr_value_dimes, ptr_value_quart

// definition of the function
void process_values(int qty_dimes, int qty_quarters, double * pt
  {
  * ptr_value_dimes = dimes * 0.10;
  * ptr_value_quarters = quarters * 0.25;
  }
```

The asterisk and must appear in both the prototype and the function definition when defining the pointer variables but it does not appear in the function call when the pointers are passed into the function.

The above example shows the basic mechanics of the indirection operator.

The use of pointers with indirection is often preferred for processing arrays. The **array index operator** is also known as

the **array method of dereferencing**. The following couts are equivalent:

```
int ages[] = {47, 45, 18, 11, 9};
cout << ages[3];
cout << *(ages + 3);
```

The both say, "The name of an array is a pointer; take the pointer and calculate a new address that points to the 3rd offset by adding the correct number of bytes onto the pointer (integer data type is normally 4 bytes long – 3 offsets times 4 bytes is 12 bytes); then dereference that pointer (since this is an Rvalue context – fetch me the value that you are pointing at) and send it to the standard output device."

You should study the demonstration programs in conjunction with this module.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device

in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials. You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Demo_Pointer_Passing.cpp

Download from Connexions: Demo_Array_Pointer_Processing.cpp

KEY TERMS

dereferencing
The concept of using the item to which a pointer or address is pointing at.

indirection operator
The asterisk used for dereferencing a pointer.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Acquire a general understanding of the pointer data type, the address and indirection operators, the concept of dereferencing.
3. Given pseudocode, write the C++ code for a program that uses reference variables.

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. The address operator is the @ symbol.
2. Passing by reference should be used when there is only one item to be modified.
3. Variables of pointer data type are defined using an asterisk.
4. Using pointers with the indirection operator can be used instead of passing variables by reference.
5. There are two kinds of dereferencing – one with the indirection operator and the other with the index operator.

Answers:

1. false
2. false
3. true

4. true
5. true

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 22 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_22 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_22_Pseudocode.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Create a source code file from the Lab_22_Pseudocode.txt file. Name it: Lab_22.cpp

- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 22a – Instructions

Give a general explanation of the pointer data type and the use of addresses and dereferencing. Include both the indirection operator and the index operator in your discussion.

REFERENCES

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

PART XXII

MORE ARRAYS & COMPILER DIRECTIVES

- Multidimensional Arrays
- Conditional Compilation
- Practice

Multidimensional Arrays

KENNETH LEROY BUSBEE

Overview

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

We refer to the individual values as members (or elements) of the array. 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 an **array member** is the **index** operator. It is important to learn how to define an array and initialize its members. The index operator is a set of square brackets with an integer value between the brackets that represents the **offset** from the front 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.

Because of the complexity for multidimensional arrays, the demonstration program shows a two dimension array and you should study it in conjunction with this module.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials. You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Demo_Multidimension_Arrays.cpp

KEY TERMS

array member
An element or value in an array.

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

list
A single dimension array.
offset
The method of referencing array members by starting at zero.

table
A two dimension array.

REFERENCES

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

Conditional Compilation

KENNETH LEROY BUSBEE

Overview

As you proceed in your programming career, the problems/tasks that need solving become more complex. The documentation of the algorithm done in pseudo code (or some other method) will still need to be converted into a programming solution. Inevitably, when writing that source code mistakes will be introduced. When learning the syntax of a new programming language, programmers sometimes automatically think in their old language syntax, and make mistakes that are sometimes hard to detect.

The concept of using a flag to either activate or have remain dormant certain lines of code designed solely to help with the debugging of a program has existed since almost the beginning of modern computer programming (1950's). One of the debugging tools available within C++ is **conditional compilation**. For our flag, we would use a defined constant like:

```
#define DEBUG 1
```

Then using another compiler directive pair, the #if and #endif, we can have the compiler during the pre-processor either include or not include one or more lines of code.

```
#if DEBUG
  cout << "\n***** DEBUG Code ** Hi mom!";
#endif
```

Of course saying “Hi mom!” is not very useful for debugging your code. However, you can use test data with conditional compilation. A series of input data values and a series of output predictors can be placed in the program. Then you can turn on the debug feature or turn them off with your debugging flag.

You should study the demonstration program in conjunction with this module.

Demonstration Program in C++

Creating a Folder or Sub-Folder for Source Code Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Demo_Programs

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Demo Program

Download and store the following file(s) to your storage device in the appropriate folder(s). Following the methods of your compiler/IDE, compile and run the program(s). Study the source code file(s) in conjunction with other learning materials. You may need to right click on the link and select “Save Target As” in order to download the file.

Download from Connexions: Demo_Conditional_Compliation.cpp

KEY TERMS

conditional compilation
A compiler directive that includes or excludes lines of code based on a Boolean expression.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Have an exposure to multidimensional arrays.
3. Understand conditional compilation as a testing technique.
4. When supplied with test data, add conditional compilation lines to an existing C++ source code.

REVIEW QUESTIONS

Answer the following statements as either true or false:

1. Very few arrays need more than one axis.
2. Multidimensional arrays use multiple square brackets, one set per axis.
3. Using a flag to activate debugging lines of code has been around since the 1950s.
4. Within C++ we can use the conditional compilation compiler directives to implement debugging line of code.

Answers:

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

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 23 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_23 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_23a.cpp

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Compile and run the Lab_23a.cpp source code file. Understand how it works.
- Copy the source code file Lab_23a.cpp naming it: Lab_23b.cpp
- Add conditional compilation statements similar to the

demonstration program used in the Conditional Compilation Connexions module. Specifically use: 157 pennies, 92 nickels, 23 dimes and 31 quarters as your test data.
- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 23a – Instructions

Give three examples in the real world where data might be structured into a multidimensional array. One example (and you can't count it) is a theatre ticket which might have a section, row and seat number on it.

REFERENCES

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

PART XXIII

OOP & HPC

- Object Oriented Programming
- Understanding High Performance Computing
- Practice

Object Oriented Programming

KENNETH LEROY BUSBEE

Discussion

"In procedural programming, the programmer constructs procedures (or functions, as they are called in C++). The procedures are collections of programming statements that perform a specific task. The procedures each contain their own variables and commonly share variables with other procedures. Procedural programming is centered on the procedure or function."[1] For decades (1950s to through the 1980s) most programming was taught as **procedural programming**. Coupled with the imposition of using standardized control structures in the late 1960s, we have what is typically called modular structured programming.

Another, equally valid approach to programming is **object-oriented programming** or OOP. It was introduced in the mid 1980s and was widely accepted as a programming approach by the early 1990s. The first languages to introduce OOP to the masses were C++ and Java. Shortly after their introduction, there were American National Standards Institute (ANSI) standards established for those languages. Today, C++ and Java are widely used.

"The primary differences between the two approaches is their use of data. In a procedural program, the design centers around the rules or procedures for processing the data. The procedures, implemented as functions in C++, are the focus of the design. The data objects are passed to the functions as parameters.

The key question is how the functions will transform the data they receive for either storage or further processing. Procedural programming has been the mainstay of computer science since its beginning and is still heavily used today.

In an object-oriented program, abbreviated OOP, the design centers around objects that contain (encapsulate) the data and the necessary functions to process the data. In OOP, the objects own the functions that process the data."[2]

"Object-oriented programming ... is centered on the object. An object is a programming element that contains data and the procedures that operate on the data. The objects contain, within themselves, both the information and the ability to manipulate the information."[3]

To help complicate the picture, the C++ programming language can be used (and is used) to write either a procedural program (modular structured program) or an object-oriented program. Some items used by those writing procedural programs in C++ are in fact objects. Examples include:

1. Standard input and output items of: cout and cin; example: cout.setf(ios::fixed)
2. Strings; calculating the length with: identifier_name.length()
3. File input/output; example: inData.open(filespec, ios::in)

Objects are implemented with a "class" data type; which is a complex or derived data type. Implementation details will not be presented in the module.

Transition

Many students will learn modular structured programming before learning object-oriented programming. The common

way of teaching programming fundamentals is to cover them or divide them into three courses, usually covered in this order:

1. Modular structured
2. Object-oriented
3. Data structures

The following items learned in modular structured programming flow into the learning of object-oriented programming:

1. The standard and complex data types are the same
2. The operators are the same, thus data manipulation is the same
3. The control structures are the same
4. Concepts of documentation and making code readable are the same
5. The use of test data to verify logical thinking and program results is similar

KEY TERMS

object oriented
A programming approach that encapsulates data with functions.

procedural programming
Aka modular structured programming.

Footnotes

- 1 Tony Gaddis, Judy Walters and Godfrey Muganda, Starting Out with C++ Early Objects Sixth Edition (United States of

America: Pearson – Addison Wesley, 2008) 22.

- 2 Behrouz A. Forouzan and Richard F. Gilberg, Computer Science A Structured Approach using C++ Second Edition (United States of America: Thompson – Brooks/Cole, 2004) 156.
- 3 Tony Gaddis, Judy Walters and Godfrey Muganda, Starting Out with C++ Early Objects Sixth Edition (United States of America: Pearson – Addison Wesley, 2008) 22.

REFERENCES

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

Understanding High Performance Computing

KENNETH LEROY BUSBEE

Preface – November 13, 2009

This module was created as an entry for the **2008-'09 Open Education Cup: High Performance Computing** competition. The competition was supervised by Dr. Jan Erik Odegard, Executive Director of the Ken Kennedy Institute for Information Technology at Rice University. It was submitted to the "Parallel Algorithms and Applications" category and specifically designed as an introduction to the subject targeting intermediate grade school students to collegiate undergraduates who have little knowledge of High Performance Computing (HPC).

This module received the **"Best Module"** award for the "Parallel Algorithms and Applications" category which included a US $500 prize.

Those who reviewed the entries for the competition made some suggestions for improvement and most have been incorporated into this revised edition of the module. As always; my thanks to them and all others who make suggestions for improving educational materials.

Kenneth Leroy Busbee

Introduction to High Performance Computing

Grouping multiple computers or multiple computer processors to accomplish a task quicker is referred to as **High Performance Computing** (HPC). We want to explain how this is accomplished using parallel programming algorithms or concepts.

The Shift from a Single Processor to Parallel

We are going to start our explanation by giving two simple examples.

After eating all you can, you toss your chicken leg bone out of the car window (shame on you for trashing up the highway), but in short order an ant finds your tossed chicken bone. One single ant could bite off the left over on the bone and transport it to the colony, one bite at a time; but, it might take him 1 whole day (24 hours) of work. But, what if he gets help? He signals some buddies and being a small colony of ants they allocate a total of 10 ants to do the task. Ten times the workers take one tenth the time. The ten ants do the task in 2 hours and 24 minutes.

I toss another bone out the window. An ant finds it and the colony allocates 50 ants to do the task of picking the bone clean. In less than 30 minutes (28.8 to be exact) the 50 ants working in parallel complete the task.

One painter might take 8 hours to paint the exterior of an average sized house. But, if he can put a crew of 10 painters working simultaneously (or in other words in parallel) it takes only 48 munities. What about a crew of 50 painters assuming that they can do work and not get in the way of each other; well how about less than 10 minutes (9.6 to be exact).

Now let's make sure we understand that the same amount of

work was done in the examples given. The work was only completed in a shorter amount of time because we put more workers on the task. Not all tasks can be divided up in this way, but when it can be divided between multiple workers, we can take advantage of the workers doing their sub part of the task in parallel. Let's look at another example.

I want to drive from Houston, Texas to Dallas, Texas; a distance of about 250 miles. For easy calculations let's say I can travel 50 miles in one hour. It would take me 5 hours. Well, I could divide the task between 5 cars and have each car travel 50 miles and arrive in Dallas in 1 hour. Right?

Well, wrong. The task of driving from Houston to Dallas cannot be divided into tasks that can be done in parallel. The task can only be done by one person driving in a line from Houston to Dallas in 5 hours. I used the word "line" because it helps connect us to the word: **linear**. A linear task cannot be broken-up into smaller tasks to be done in parallel by multiple workers. Within the computer world, the word associated with linear concept is **sequential processing**. I must drive one mile at a time in sequence to get to Dallas.

Our natural tendency is to share the work that is to work in parallel whenever it is possible. As a group we can accomplish many tasks that can be done in parallel in less time.

The Birth of Computers – A "Parallel" to Central Processing Unit (CPU) Story

"ENIAC, short for Electronic Numerical Integrator And Computer, was the first general-purpose electronic computer (July 1946). It was the first Turing-complete, digital computer capable of being reprogrammed to solve a full range of computing problems. ENIAC had twenty ten-digit signed

accumulators which used ten's complement representation and could perform 5,000 simple addition or subtraction operations between any of them and a source (e.g., another accumulator, or a constant transmitter) every second. It was possible to connect several accumulators to run simultaneously, so the peak speed of operation was potentially much higher due to parallel operation." (ENIAC from Wikipedia)

Often not understood by many today, the first computer used base 10 arithmetic in the electronics and was a **parallel processing**machine by using several accumulators to improve the speed. However, this did not last for long. During its construction:

"The First Draft of a Report (commonly shortened to First Draft) on the EDVAC – Electronic Discrete Variable Automatic Computer was an incomplete 101 page document written by John von Neumann and distributed on June 30, 1945 by Herman Goldstine, security officer on the classified ENIAC project. It contains the first published description of the logical design of a computer using the stored-program concept, which has come to be known as the von Neumann architecture." (First Draft of a Report on the EDVAC from Wikipedia)

"The von Neumann architecture is a design model for a stored-program digital computer that uses a [central] processing [unit] and a single separate storage structure to hold both instructions and data. It is named after the mathematician and early computer scientist John von Neumann. Such computers implement a universal Turing machine and have a sequential architecture." (Von Neumann architecture from Wikipedia)

Von Neumann also proposed using a binary (base 2) numbering system for the electronics. One of the characteristics of the von Neumann architecture was the trade off of multiple processors using base 10 electronics to a single central processor using base 2 (or digital) electronics. To compare to our ant

example, the idea was to use one real fast ant versus 10 slow ants. If one real fast ant can do 1,000 tasks in an hour; it would be more powerful (be able to do more tasks) than 10 ants doing 10 tasks an hour or the equivalent of 100 tasks per hour.

The rest is history – most commercially built computers for about the first forty years (1951 to 1991) followed the von Neumann architecture. The electronic engineers keep building more reliable and faster electronics. From vacuum tube, to transistor, to integrated circuit to what we call today "chip" technology. This transformation made computers break down less frequently (they were more reliable), physically smaller, needing less electric power and faster. Personal computers were introduced in the late 1970's and within ten years became more commonly available and used.

One short coming was that most programming efforts were towards improving the linear (or sequential) way of thinking or solving a problem. After all, the computer electronic engineers would be making a faster computer next year. Everyone understood that the computer had only one **central processing unit** (CPU). Right?

The Need for Power

Well, wrong. Computer scientists and electronic engineers had been **experimenting** with multi-processor computers with parallel programming since 1946. But it's not until the 1980's that we see the first parallel processing computers (built by Cray and other computer companies) being sold as commercial built computers. It's time for another example.

The circus traveling by train from one city to the next has an elephant that dies. They decide to toss the elephant off the train (shame on them for trashing up the country side), but in short

order a “super” ant (faster than most regular ants) finds the elephant. This project is much larger than your tossed chicken bone. One single “super” ant could do the task (bite off a piece of the elephant and transport it to the colony, one bite at a time); but, it might take one whole year. After all this requires a lot more work than a chicken bone. But, what if he gets help? He signals some buddies and being a large colony of “super” ants they allocate a total of 2,190 ants to do the task. Wow, they devour the elephant in six hours.

This elephant example is exactly where the computer scientists had arrived. The electronic engineers were going to continue to make improvements in the speed of a single central processing unit computer, but not soon enough to satisfy the “need for power” to be able to solve tasks requiring **immense computing power**. Some of the new tasks that would require immense computer power included the human genome project, searching for oil and gas by creating 3 dimensional images of geological formations and the study of gravitational forces in the universe; just to mention a few. The solution: parallel processing to the rescue. Basically the only way to get this immense computer power was to implement parallel processing techniques. During the late 1970’s and early 1980’s scientists saw the need to explore the parallel processing paradigm more fully and thus the birth of High Performance Computing. Various national and international conferences started during the 1980’s to be able to further the cause of High Performance Computing. For example in November of 2008 the “SC08” supercomputing conference celebrated their 20thanniversary.

The predicting of the weather is a good example for the need of High Performance Computing. Using the fastest central processing unit computer it might take a year to predict tomorrow’s weather. The information would be correct but 365

days late. Using parallel processing techniques and a powerful "high performance computer", we might be able to predict tomorrow's weather in 6 hours. Not only correct, but in time to be useful.

Measuring Computer Power

Most people are familiar with the giga hertz (billions of instructions per second) measure to describe how fast a single CPU's processor is running. Most microcomputers of today are running around 3 GHz or 3 billion instructions a second. Although 3 billion sounds fast, many of these instructions are simple operations.

Supercomputing uses a measurement involving floating point arithmetic calculations as the benchmark for comparing computer power. "In computing, **FLOPS** (or **flops** or **flop**/**s**) is an acronym meaning **FL**oating point **O**perations **P**er **S**econd." and again "On May 25, 2008, an American military supercomputer built by IBM, named 'Roadrunner', reached the computing milestone of one petaflop by processing more than 1.026 quadrillion calculations per second." (FLOPS from Wikipedia)
For those of us not familiar:
Getting a Sense of Power

```
3 billion or 3 GHz is:                  3,000,000,000
1 quadrillion or 1 pedaflop is: 1,000,000,000,000,000
```

You also should realize that your personal computer is not doing 3 gigafolp worth of calculations, but something slower when using the FLOPS measurement.

High Performance Computing Made Personal

It took several years (about 30) to get computers to a personal

level (1951 to 1981). It took about twenty years (late 1980's to present 2009) to get multi-processor computers to the personal level. Currently available to the general public are computers with "duo core" and "quad core" processors. In the near future, micro computers will have 8 to 16 core processors. People ask, "Why would I need that much computer power?" There are dozens of applications, but I can think of a least one item that almost everyone wants: high quality voice recognition. That's right! I want to talk to my computer. Toss your mouse, toss your keyboard, no more touch pad – talk to it.

Again, one short coming is that most programming efforts have been towards teaching and learning the sequential processing way of thinking or solving a problem. Educators will now need to teach and programmers will now need to develop skills in programming using parallel concepts and algorithms.

Summary

We have bounced you back and forth between sequential and parallel concepts. We covered our natural tendency to do work in parallel. But with the birth of computers the parallel concepts were set to the side and the computer industry implemented a faster single processor approach (sequential). We explained the limitations of sequential processing and the need for computing power. Thus, the birth of High Performance Computing. Parallel processing computers are migrating into our homes. With that migration, there is a great need to educate the existing generation and develop the next generation of scientists and programmers to be able to take advantage of High Performance Computing.

Learner Appropriate Activities

High Performance Computing is impacting how we do everything. Learning, working, even our relaxation and entertainment are impacted by HPC. To help more people understand HPC, I have listed appropriate activities based on where a learner is in relation to their programming skills.

Computer Literacy but No Programming Skills

We have provided two computer programs that help students see the impact of parallel processing. The first is a "Linear to Parallel Calculator" where the student enters how long it would take one person to complete a task, asks how many people will work as a group on the task, then calculates how long it will take the group to complete the task. The second is a "Parallel Speed Demonstration Program" that simulates parallel processing. It displays to the monitor the first 60 factorial numbers in 60 seconds, then shows as if 10 processors are doing it in 6 seconds, then as if 100 processors are doing it in less than 1 second. Both are compiled and ready for use on an Intel CPU machine (compiled for use on Windows OS).

Download the executable file from Connexions: Linear to Parallel Calculator

Download the executable file from Connexions: Parallel Speed Demonstration Program

An interesting activity would be to join a group that is using thousands of personal microcomputers via Internet connections for parallel processing. Several distributed processing projects are listed in the "FLOPS" article on Widipedia. One such group is the "Great Internet Mersenne Prime Search – GIMPS".

A link to the GIMPS web site is: http://www.mersenne.org/

Another activity is to "Google" some keywords. Be careful –

"Googling" can be confusing and often can be difficult to focus on the precise subject that you want.

- high performance computing
- computational science
- supercomputing
- distributed processing

Learning Programming Fundamentals

Students learning to program that are currently taking courses in Modular/Structured programming and/or Object Oriented programming might want to review the source code files for the demonstration programs listed above. These programs do not do parallel programming, but the student could modify or improve them to better explain parallel programming concepts.

You may need to right click on the link and select "Save Target As" in order to download these source code files.

Download the source code file from Connexions: Linear to Parallel Calculator

Download the source code file from Connexions: Parallel Speed Demonstration Program

Another appropriate activity is to "Google" some of the key words listed above. With your fundamental understanding of programming, you will understand more of the materials than those with no programming experience. You should get a sense that parallel programming is becoming a more important part of a computer professional's work and career.

Review the "Top 500 Super Computers" at: http://www.top500.org/

Look at the source code listings provided in the next section, but remember, you cannot compile or run these on your normal computer.

Upper Division Under-Graduate College Students

The challenge is to try parallel computing, not just talk about it.

During the week of May 21st to May 26th in 2006, this author attended a workshop on Parallel and Distributed Computing. The workshop was given by the National Computational Science Institute and introduced **parallel programming** using multiple computers (a group of micro computers grouped or clustered into a super-micro computer). The conference emphasized several important points related to the computer industry:

1. During the past few years super-micro computers have become more powerful and more available.
2. Desk top computers are starting to be built with multiple processors (or cores) and we will have multiple (10 to 30) core processors within a few years.
3. Use of super-micro computing power is wide spread and growing in all areas: scientific research, engineering applications, 3D animation for computer games and education, etc.
4. There is a shortage of educators, scientific researchers, and computer professionals that know how to manage and utilize this developing resource. Computer professionals needed include: Technicians that know how to create and maintain a super-micro computer; and **Programmers that know how to create computer applications that use parallel programming concepts**.

This last item was emphasized to those of you beginning a career in computer programming that as you progress in your education, you should be aware of the changing nature of computer programming as a profession. Within a few years **all professional programmers will have to be familiar with parallel programming**.

During the conference this author wrote a program that sorts an array of 150,000 integers using two different approaches. The first way was without parallel processing. When it was compiled and executed using a single machine, it took 120.324 seconds to run (2 minutes). The second way was to redesign the program so parts of it could be run on several processors at the same time. When it was compiled and executed using 11 machines within a cluster of micro-computers, it took 20.974 seconds to run. That's approximately 6 times faster. Thus, **parallel programming will become a necessity to be able to utilize the multi-processor hardware of the near future.**

A distributed computing environment was set up in a normal computer lab using a Linix operating system stored on a CD. After booting several computers with the CD, the computers can communicate with each other with the support of "Message Passing Interface" or MPI commands. This model known as the Bootable Cluster CD (BCCD) is available from:

Bootable Cluster CD – University of Northern Iowa at: http://www.bccd.net/

The source code files used during the above workshop were modified to a version 8, thus an 8 is in the filename. The non-parallel processing "super" code was named: nonps8.cpp with the parallel processing "super" code named: ps8.cpp (Note: The parallel processing code contains some comments that describe that part of the code being run by a machine identified as the "SERVER_NODE" with a part of the code being run by the 10 other machines (the Clients). The client machines communicate critical information to the server node using "Message Passing Interface" or MPI commands.)

You may need to right click on the link and select "Save Target As" in order to download these source code files.

Download the source code file from Connexions: nonps8.cpp

Download the source code file from Connexions: ps8.cpp

Two notable resources with super computer information were provided by presenters during the workshop:

Oklahoma University – Supercomputing Center for Education & Research at: http://www.oscer.ou.edu/education.php

Contra Costa College – High Performance Computing at: http://contracosta.edu/hpc/resources/presentations/

You can also "Google" the topic's key words and spend several days reading and experimenting with High Performance Computing.

Consider reviewing the "Educator Resources" links provided in the next section.

Educator Resources

There are many sites that provide materials and assistance to those teaching the many aspects of High Performance Computing. A few of them are:

Shodor – A National Resource for Computational Science Education at: http://www.shodor.org/home/

CSERD – Computational Science Education Reference Desk at: http://www.shodor.org/refdesk/

National Computational Science Institute at: http://www.computationalscience.org/

Association of Computing Machinery at: http://www.acm.org/

Super Computing – Education at: http://sc09.sc-education.org/about/index.php

Simple Definitions

high performance computing

Grouping multiple computers or multiple computer processors to accomplish a task in less time.

sequential processing
Using only one processor and completing the tasks in a sequential order.
parallel processing
Dividing a task into parts that can utilize more than one processor.
central processing unit
The electronic circuitry that actually executes computer instructions.
parallel programming
Involves developing programs that utilize parallel processing algorithms that take advantage of multiple processors.

REFERENCES

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

Practice

KENNETH LEROY BUSBEE

LEARNING OBJECTIVES

1. Understand key terms and definitions as listed in the modules associated with this chapter.
2. Gain an exposure to object-oriented programming.
3. Gain an exposure to high performance computing.
4. Given general instructions, write the C++ code for a program that includes a general review of the textbook/collection/course.

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.
3. High Performance Computing is a new topic on the computer scene.
4. The concepts and examples of High Performance Computer are difficult to explain.
5. All programmers will need to know about parallel programming in the near future.

Answers:

1. false

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

ACTIVITIES

Creating a Folder or Sub-Folder for Chapter 24 Files

Depending on your compiler/IDE, you should decide where to download and store source code files for processing. Prudence dictates that you create these folders as needed prior to downloading source code files. A suggested sub-folder for the Bloodshed Dev-C++ 5 compiler/IDE might be named:

- Chapter_24 within the folder named:
  Cpp_Source_Code_Files

If you have not done so, please create the folder(s) and/or sub-folder(s) as appropriate.

Download the Lab File(s)

Download and store the following file(s) to your storage device in the appropriate folder(s). You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Lab_24_Narrative_Description.txt

Detailed Lab Instructions

Read and follow the directions below carefully, and perform the steps in the order listed.

- Create a source code file following the directions in the Lab_24_Narrative_Description.txt file. Name it: Lab_24.cpp
- Build (compile and run) your program.
- After you have successfully written this program, if you are taking this course for college credit, follow the instructions from your professor/instructor for submitting it for grading.

Problems

Problem 24a – Instructions

Describe the fundamental differences between procedural (modular structured) programming and object-oriented programming.

Problem 24b – Instructions

Explain why High Performance Computing is needed to predict tomorrow's weather.

REFERENCES

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

PART XXIV

REVIEW MATERIALS

- Review: Foundation Topics Group: 1-5
- Review: Modular Programming Group: 6-9
- Review: Structured Programming Group: 10-16
- Review: Intermediate Topics Group: 17-21
- Review: Advanced Topics Group: 22-24

Review: Foundation Topics Group: 1-5

KENNETH LEROY BUSBEE

Strategy Discussion

Exams vary depending on your instructor. Many will use the following:

1. Definitions
2. Self-grading questions including true/false, multiple choice, short answer, etc.
3. Problems

The materials in this textbook/collection have covered these items at the end of every chapter within the **Practice** module for that chapter. We suggest the following test preparation strategies:

1. If your professor is testing the definitions and expecting you to have them memorized, you should review the "Using the Flash Card Activity" within the "Study Habits that Build the Brain" module within the Appendix materials. Practice writing your definitions using the Flash Card Activity in the Memory Building Activities (MBAs) available within the **Practice** modules or in the Memory Building Activities within this **Review** module.
2. Do a quick review of any exercises within the Connexions modules or the **Practice** modules. Also review quizzes or exams that you have taken and pay special attention to

making sure you understand why you missed a question.

3. If your professor has indicated that they might include a few of the problems presented within the **Practice** modules, make sure you have formulated a good answer for each problem. If authorized, collaborate with other students to improve your answers to the problems. Spend a moderate amount of time reviewing each problem with its answer before the exam.

REFERENCES

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

Review: Modular Programming Group: 6-9

KENNETH LEROY BUSBEE

Strategy Discussion

Exams vary depending on your instructor. Many will use the following:

1. Definitions
2. Self-grading questions including true/false, multiple choice, short answer, etc.
3. Problems

The materials in this textbook/collection have covered these items at the end of every chapter within the **Practice** module for that chapter. We suggest the following test preparation strategies:

1. If your professor is testing the definitions and expecting you to have them memorized, you should review the "Using the Flash Card Activity" within the "Study Habits that Build the Brain" module within the Appendix materials. Practice writing your definitions using the Flash Card Activity in conjunction with the Memory Building Activities (MBAs) available within the **Practice** modules or in the Memory Building Activities within this **Review** module.
2. Do a quick review of any exercises within the Connexions

modules or the **Practice** modules. Also review quizzes or exams that you have taken and pay special attention to making sure you understand why you missed a question.

3. If your professor has indicated that they might include a few of the problems presented within the **Practice** modules, make sure you have formulated a good answer for each problem. If authorized, collaborate with other students to improve your answers to the problems. Spend a moderate amount of time reviewing each problem with its answer before the exam.

REFERENCES

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

Review: Structured Programming Group: 10-16

KENNETH LEROY BUSBEE

Strategy Discussion

Exams vary depending on your instructor. Many will use the following:

1. Definitions
2. Self-grading questions including true/false, multiple choice, short answer, etc.
3. Problems

The materials in this textbook/collection have covered these items at the end of every chapter within the **Practice** module for that chapter. We suggest the following test preparation strategies:

1. If your professor is testing the definitions and expecting you to have them memorized, you should review the "Using the Flash Card Activity" within the "Study Habits that Build the Brain" module within the Appendix materials. Practice writing your definitions using the Flash Card Activity in conjunction with the Memory Building Activities (MBAs) available within the **Practice** modules or in the Memory Building Activities within this **Review** module.
2. Do a quick review of any exercises within the Connexions

modules or the **Practice** moudles. Also review quizzes or exams that you have taken and pay special attention to making sure you understand why you missed a question.

3. If your professor has indicated that they might include a few of the problems presented within the **Practice** modules, make sure you have formulated a good answer for each problem. If authorized, collaborate with other students to improve your answers to the problems. Spend a moderate amount of time reviewing each problem with its answer before the exam.

REFERENCES

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

Review: Intermediate Topics Group: 17-21

KENNETH LEROY BUSBEE

Strategy Discussion

Exams vary depending on your instructor. Many will use the following:

1. Definitions
2. Self-grading questions including true/false, multiple choice, short answer, etc.
3. Problems

The materials in this textbook/collection have covered these items at the end of every chapter within the **Practice** module for that chapter. We suggest the following test preparation strategies:

1. If your professor is testing the definitions and expecting you to have them memorized, you should review the "Using the Flash Card Activity" within the "Study Habits that Build the Brain" module within the Appendix materials. Practice writing your definitions using the Flash Card Activity in conjunction with the Memory Building Activities (MBAs) available within the **Practice** modules or in the Memory Building Activities within this **Review** module.
2. Do a quick review of any exercises wtihin the Connexions modules or **Practice** modules. Also review quizzes or exams that you have taken and pay special attention to making

sure you understand why you missed a question.

3. If your professor has indicated that they might include a few of the problems presented within the **Practice** modules, make sure you have formulated a good answer for each problem. If authorized, collaborate with other students to improve your answers to the problems. Spend a moderate amount of time reviewing each problem with its answer before the exam.

REFERENCES

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

Review: Advanced Topics Group: 22-24

KENNETH LEROY BUSBEE

Strategy Discussion

Exams vary depending on your instructor. Many will use the following:

1. Definitions
2. Self-grading questions including true/false, multiple choice, short answer, etc.
3. Problems

The materials in this textbook/collection have covered these items at the end of every chapter within the **Practice** module for that chapter. We suggest the following test preparation strategies:

1. If your professor is testing the definitions and expecting you to have them memorized, you should review the "Using the Flash Card Activity" within the "Study Habits that Build the Brain" module within the Appendix materials. Practice writing your definitions using the Flash Card Activity in conjunction with the Memory Building Activities (MBAs) available within the **Practice** modules or in the Memory Building Activities within this **Review** module.
2. Do a quick review of any exerecises within the Connexions modules or the **Practice** modules. Also review quizzes or exams that you have taken and pay special attention to

making sure you understand why you missed a question.

3. If your professor has indicated that they might include a few of the problems presented within the **Practice** modules, make sure you have formulated a good answer for each problem. If authorized, collaborate with other students to improve your answers to the problems. Spend a moderate amount of time reviewing each problem with its answer before the exam.

REFERENCES

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

PART XXV

APPENDIX

- Abbreviated Precedence Chart for C++ Operators
- C++ Reserved Keywords
- ASCII Character Set
- Show Hide File Extensions
- Academic or Scholastic Dishonesty
- Successful Learning Skills
- Study Habits that Build the Brain

Abbreviated Precedence Chart for C++ Operators

KENNETH LEROY BUSBEE

An operator is a language-specific syntactical token (one or more symbols) that causes an action to be taken on one or more operands. The following item provides an abbreviated list of those C++ operators that are typically taught in a programming fundamentals course that teaches modular structured programming concepts.

The first column shows the precedence (the higher precedence is 1 or it goes first) and operators that have the same precedence also have the same associativity (the associativity is only listed once for the group of operators). Decrement is two minus signs, but some word processing software programs might have problems printing two minus signs and convert it to a double dash. Insertion (two < signs) and extraction (two > signs) might also have printing problems. These printing problems are noted in the comments with **emphasized** text.

PR	OPERATOR NAME	SYMBOL(S)	COMMENTS	ASSOICIATIVITY	CONNEXIONSMO
1	function call	()		Left to Right	m19145
1	index	[]	aka array index		m21316
2	class member	.	a period	Right to Left	m20796
2	postfix increment	++	unary		m20499
2	postfix decrement	—	unary, **two minus signs**		m20499
3	indirection	*	unary, aka dereference	Right to Left	m22152
3	address	&	unary		m22148
3	unary positive	+	unary, aka plus		m20501
3	unary negative	–	unary, aka minus		m20501
3	prefix increment	++	unary		m20499
3	prefix decrement	—	unary, **two minus signs**		m20499
3	cast	(type)	unary		m18744
3	sizeof	sizeof (type)	unary		m18736
3	logical NOT	!	unary		m19847
4	multiply	*		Left to Right	m18706
4	divide	/			m18706
4	modulus	%	remainder		m18706
5	add	+		Left to Right	m18706
5	subtract	–			m18706
6	insertion	<<	writing, **two less than signs**	Left to Right	m18835

6	extraction	>>	reading, **two greater than signs**		m18835
7	less than	<		Left to Right	m19549
7	greater than	>			m19549
7	less than or equal to	<=			m19549
7	greater than or equal to	>=			m19549
8	equality	==	equal to	Left to Right	m19549
8	inequality	!=	not equal to		m19549
9	logical AND	&&		Left to Right	m19847
10	logical OR	||		Left to Right	m19847
11	conditional	? :	trinary	Left to Right	m20811
12	assignment	=		Right to Left	m18725
12	addition assignment	+=			m18743
12	subtraction assignment	-=			m18743
12	multiplication assignment	*=			m18743
12	division assignment	/=			m18743
12	modulus assignment	%=			m18743
13	sequence or comma	,		Left to Right	m18690

REFERENCES

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

C++ Reserved Keywords

KENNETH LEROY BUSBEE

All programming languages have “reserved words”. There are usually less than 50 of these reserved words in any given programming language. They are reserved because they have been pre-assigned a specific meaning within that programming language, thus the compiler recognizes those words to mean a specific thing or action. Within C++ the reserved words are also known as “keywords”.

Programmers use identifier names for a variety of items, to include: functions, variables, named constants, alias names, etc. But, they can’t use as identifier names the words that are “reserved to the language”.

For the C++ language all “reserved keywords” are typed in lower case. The list that follows includes the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) lists of reserved words for the C++ programming language. The ISO reserved words may not be implemented in the compiler that you are using, however they may be adopted in future releases of C++ compilers. Wisdom dictates to avoid using them at this point so that there will not be a problem compiling your source code in future releases of compilers. There has been no distinction made in the ANSI or ISO reserved word lists. A search of the Internet for C++ reserved words will reveal several different lists. Some are more unique to a specific compiler. Some will be incomplete because the list has been enlarged. The talbe that follows should work for any beginning programming course using C++. The reserved keywords are:

and	double	not_eq	throw
and_eq	dynamic_cast	operator	true
asm	else	or	try
auto	enum	or_eq	typedef
bitand	explicit	private	typeid
bitor	extern	protected	typename
bool	false	public	union
break	float	register	unsigned
case	fro	reinterpret-cast	using
catch	friend	return	virtual
char	goto	short	void
class	if	signed	volatile
compl	inline	sizeof	wchar_t
const	int	static	while
const-cast	long	static_cast	xor
continue	mutable	struct	xor_eq
default	namespace	switch	
delete	new	template	
do	not	this	

REFERENCES

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

ASCII Character Set

KENNETH LEROY BUSBEE

ASCII stands for American Standard Code for Information Interchange (pronounced “ask-key”). Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as ‘a’ or ‘@’ or an action of some sort. ASCII was developed a long time ago and now the non-printing characters are rarely used for their original purpose. The first 32 values (0 to 31) and the last value (127) are the non-printing characters.

Several software products can be used to create an ASCII text file.

- Notepad within Windows OS and it uses by default the .txt extension.
- Microsoft Word by saving the file as ‘text only’ and it uses by default the .txt extension.
- Integrated Development Environment (IDE) compliers for most programming languages usually save source code as ASCII text files but they will use an extension that describes the content of the text file. Example: C++ usually uses .cpp as the extension.

The following web links provide more information and tables listing the ASCII Character Set:

- http://asciiset.com/
- http://www.asciitable.com/
- http://en.wikipedia.org/wiki/ASCII

REFERENCES

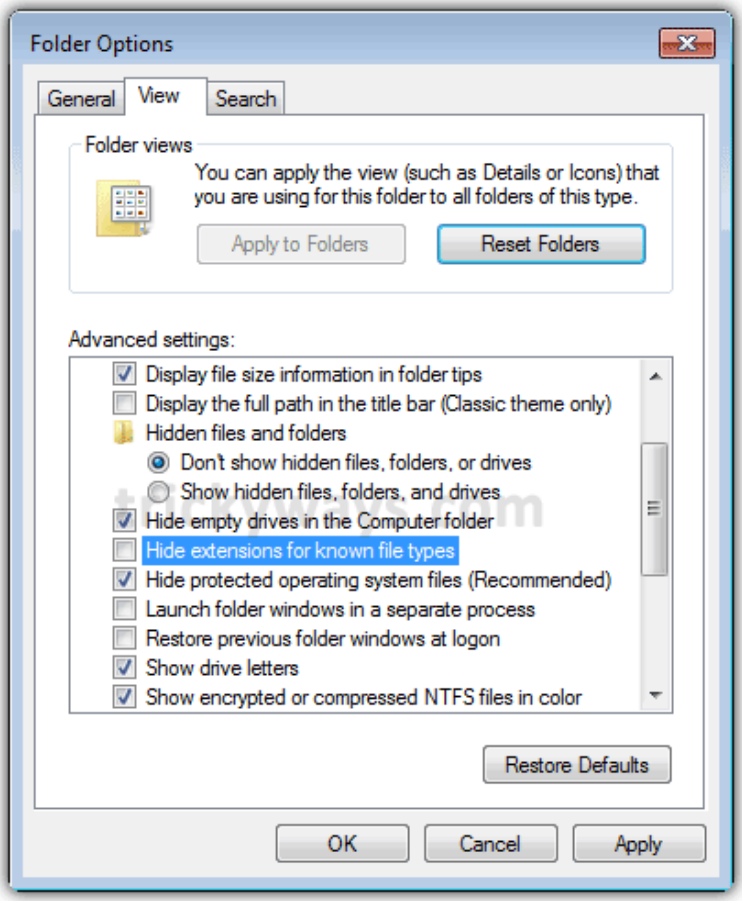
- cnx.org: Programming Fundamentals – A Modular Structured Approach using C++

Show Hide File Extensions

KENNETH LEROY BUSBEE

By default, file extensions for known file types are hidden in Windows operating systems. However, you can change this setting so that file extensions are shown for all file types. Being able to see file extensions can be very helpful for students taking computer courses because those course instructions often refer to file extensions.

All Windows operating systems navigate you to the “Folder Options” menu, then have you select the “View” tab. Indeed the box is identical in Windows XP, Windows Vista and Windows 7.



The check in the box acts like a toggle switch. With a check present, it will hide known file types. Without the check present, it will show all file types. Click on the box to make the check appear [hide file extensions] or disappear [show file extensions] and then select “OK”.

Instructions for navigating to the “Folder Options” for various Windows operating systems along with an Internet link for additional help are provided below.

Windows XP

With the Windows Explorer open, slect the “Tools” tab and then “Folder Options”.

Link for additional help: http://www.fileinfo.net/help/windows-show-extensions.html or http://dotwhat.net/page/displayextensions/

Windows Vista

Select the “Start” button, then “Control Panel”, then “Appearance and Personalization” and then “Folder Options”.

Link for additional help: http://windows.microsoft.com/en-us/windows-vista/Show-or-hide-file-name-extensions

Windows 7

Select the “Start” button, then “Control Panel” and then “Folder Options”.

Link for additional help: http://maximumpcguides.com/windows-7/hide-file-extensions/

REFERENCES

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

Academic or Scholastic Dishonesty

KENNETH LEROY BUSBEE

Introduction

The relationship between faculty and students has always been one of open and honest communication. The faculty member carries the responsibility of presenting course materials via reading assignments, lectures, labs, etc. The student is to learn and understand these materials. Additionally, the faculty members employ various methods to assess the student's mastery of the course materials. Frequently this is done via quizzes, tests, writing assignments, the completion of lab materials, etc. Academic dishonesty (sometimes called "Scholastic Dishonesty") is the violation of that trust.

Cheating on quizzes and tests as well as plagiarism is usually well understood by students before arriving at the collegiate level of education. Most colleges include adequate explanation in their student handbook explaining well what constitutes cheating on exams and plagiarism. Academic dishonesty often carries some stiff penalties. Usually, the student receives the grade of "F" from the professor in the course in which he is enrolled. The student might be expelled from all of their classes for which they are currently enrolled ("F" in all of your classes) and expelled from the institution (may not register for classes in the future). Sounds harsh, but it is a violation of the **bond of trust** between the student and the educational institution.

Collusion

Another category of academic dishonesty is collusion which is the unauthorized collaboration with another person in preparing written work (including lab assignments) offered for credit (counting towards your grade calculation). To better understand collusion, students need to realize that as part of the learning and evaluation of that learning, many professors use group projects; a directed or authorized collaboration. Often students are encouraged to form study groups to help discuss the course materials thus improving the learning process. These authorized and sometimes directed activities are not collusion.

The following discussion is to help the student understand collusion (unauthorized collaboration) with specific reference to courses that use computers. This is not an all inclusive list, but will cover the common situations that faculty have encountered over the years. Unless your specific professor informs you differently, you are to assume that the following items discussed are collusion.

Type it Yourself

Lab assignments are to be your own personal typing efforts. That is you are to type them or make the modifications yourself to the files (documents, spreadsheets, databases, programming source code, etc.) If your course is a programming subject, you are to run the source code file on your compiler, making corrections as need to complete the lab assignment. If the directions for an assignment include starting a new file then don't use an existing file and modify it to complete the assignment. **Unless specifically authorized by your professor, students should not complete computerized course work as a team or group and then share the final completed product.**

Students have said that they worked as a team or group and that all participated and all learned the materials. Don't try this excuse because professors don't buy it. Here is the problem: Part of the learning process is in you doing it yourself. Example: I ask two students to make me some pancakes for breakfast; I expect two individually prepared plates of pancakes (one from each of them) for my breakfast. The professor really does not want to eat two plates of pancakes (or 50 to 100 plates of pancakes, depending on how many students they are teaching), but part of your directed learning activity for the course is to demonstrate that you can make pancakes (not watch someone else make pancakes or participate as a group to make pancakes).

Control Access to Your Files

Controlling the files you create (or are directed to modify) means that others will not have access to copy your work. In other words, don't share your files.

Students have said that they shared the file so they the other student could see how the completed assignment should look. Don't try this excuse because professors don't buy it. Here is the problem: When you share the file you share your typing efforts (or your original work and your efforts to create that original work). Back to our pancake example: "I only gave the other student a plate of completed pancakes, so he could see what the end product should be." All the other student does is add some blue berries and whip cream. If a student makes minor modifications to your work (changes the spots where his name is at) and turns it in as his work – you will be included in the charge of academic dishonesty. **Unless specifically authorized by your professor, don't share any files that you create or modify with another student – ever, not now and not in the future.**

Here are two suggestions for controlling access to your files:

When using a course delivery software product or learning system, such as BlackBorad Vista, **don't give another person your password**. With the password, they will have access to your submitted assignments including the files that you created.

Don't leave your files on a machine where others may have access to them. If multiple students are using or have access to the same machine (often happens with students living in the same household – husband/wife, siblings or roommates) or in an on-campus course where many students will have access to the machine – **store your files on a flash drive**. Physically control who gets access to your flash drive.

Ask for a Clarification of the Collaboration

If you have any question about an activity that might be construed as unauthorized collaboration, ask your professor. They will provide clarification and direction to you about the activity.

Students have said that they did not understand or think that it was unauthorized collaboration. Don't try this excuse because professors don't buy it. Here is the problem: We can't, and won't list every minor way in which students can collude. **The burden is for you to ask for any clarification for the specific course from your professor.** Don't assume that what another instructor allowed in another course will be allowed by this professor in this course.

Detecting Academic Dishonesty

Professors weren't born yesterday. The faculty members of most institutions have individually years and collectively thousands of years at understanding academic dishonesty. Cheating on

tests, plagiarism and collusion are not new to us. We share our expertise with each other at detecting academic dishonesty. Additionally, the years of technical computer experience of professors who teach using computers in lab settings is often astounding.

Students have said that they did not think they could be detected or that academic dishonesty could not be proved. Don't try this approach because professors believe that they are slightly smarter. Actually, we know that we are a lot smarter. It amazes us that student don't realize that professors are a formidable force. **Don't gamble that you can beat us at the "Academic Dishonesty Game".** Please don't take this as a challenge and use it as an excuse to see if you can be academically dishonest and not get caught. We are warning you, not challenging you.

Serious Consequences

The consequences will vary from instructor to instructor and from institution to institution. They range from a simple slap on the hand (don't do it again) to complete explusion from the institution (expelled from all of your courses). Because the **bond of trust** is broken, many instructors will simply expel you from the course you are taking. As an example: Within the BCIS1405 course at Houston Community College, we expelled 8 students (along with giving them the grade of "F") from Distance Educations sections during the Spring 2008 term for Academic Dishonesty.

Be ready for what ever the consequences your instructor will deliver if you are dishonest.

Summary

- The ethics of academic honesty; there is a bond of trust that whatever the student does in relationship to the evaluation process are their own work and efforts.
- Collusion is the unauthorized collaboration of students on work submitted for evaluation.
- First directive: Type if yourself
- Second directive: Don't share your files
- Seek clarification from your professor if you have any doubt that the collaborative activity might be considered collusion.
- Professors are very capable at detecting academic dishonesty.
- There are usually consequences to your dishonest behavior.

REFERENCES

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

Successful Learning Skills

KENNETH LEROY BUSBEE

Realize the Time Commitment

College computer courses often are listed in the catalog of courses with both lecture and lab hours. But unlike the natural and biological sciences (chemistry, physics and biology) that must meet in a specific lab room designed for those courses, students can usually complete their lab portions at a variety of locations (the college's computer lab, home, work, public library, friend's house, etc.).

The normal rule of thumb is 1 to 1.5 hours out of class studying for every hour in class and for computer courses this normally means both the lecture and lab hours. Students with learning disabilities or those whose primary language is not English will want to plan for more study time and should use a larger ratio. Thus, you should calculate the weekly hours of commitment needed for a course depending on your circumstances. Example:

If a student is taking a 4 credit hour computer course that the college catalog says contains a combination of 6 hours (adding your lecture and lab hours) during a regular 16 week semester; the weekly classroom and study time for that course would be 12 to 15 hours a week.

But many students take courses at a faster pace by either taking a course between semesters in a very concentrated mode, starting a course after the regular start of a semester or during the summer. To calculate the weekly study time needed

you will need to calculate the total regular semester instructional time and divide by the number of weeks in the faster pace delivery. Example:

Our 4 credit hour course is to be taken during a summer term that has 9 weeks of instruction time. The total regular semester time would be 15 times the normal semester commitment (180 to 225 hours). Dividing it by 9 would mean 20 to 25 hours per week.

Understand Your Capacity to Concentrate

You cannot expect to spend long periods of time working on computer course materials. After 3 to 4 hours of working on course materials, your ability to learn drops significantly (and for most to near zero). This problem is compounded by the nature of the material which is cumulative in nature. This means that you must understand item a before you try to learn item b. All of the math and sciences courses of study are of this nature.

Plan Regular Study Times

The combination of the time commitment and your ability to concentrate leads to the conclusion that you cannot cram your study time into a week-end of concentrated study. You must break up your study time into 3 to 4 hour study periods doing only one study period per day. You must establish a regular routine for each week. Students taking a regular semester course on-campus will count their class (lecture and lab) time and plan 2 to 3 additional study periods.

If taking a course via distance education, students need to plan for all of the course time, thus during a regular semester term, our 4 credit hour course example would require 3 to 4 study periods with 3 to 4 hours for each study period per week.

If taking the course at **faster pace** (9 week summer term) you will need to schedule more study times. This may mean a **3 to 4 hour study period daily for 6 days a week** (with only one day off as a day of rest).

You need to stay on top of a course to successfully complete it. Pacing yourself with multiple study times allows for effective learning. Students who procrastinate until close to an exam and then try cramming through course materials are rarely "A" students.

Learning Requires Variety and Repetition

Variety comes in many forms and includes lecture, lab assignments, studying textbooks, multi-media materials, quizzes, writing a research papers, learning activities such as group discussions, crossword puzzles, flash cards, etc. This variety actually helps our brain to understand and build memory. In addition to variety, repetition (exposure over multiple study periods) is essential for our brains to be able to learn and recall the course materials. Again, this understanding and recall are essential to courses that require cumulative learning (you must understand item a before you can learn item b).

Textbooks and professors break-up course materials into chapters or learning modules often with learning objectives first and review items at the end of each unit. Each chapter or module might have any of the above mentioned items. But doing things and study are different. You can't just show up to class and listen, you can't just read stuff, you need to study. Study requires a variety of activities. Ask yourself:

- Do you understand each learning objective?
- Can you explain or formulate an answer for each learning

objective?
- If you did not understand the reading materials, did you re-read it?
- Do the review items (especially questions).
- Take lecture notes.
- Do the lecture notes or handouts give you a better understanding than the textbook?
- Often the problems or lab assignments are to be studied in conjunction with and reinforce the study materials. Have you tried to do and understand the problems or lab assignments?
- Are there any learning activities available and if yes, did you do them.
- Did you consider using 3×5 cards to study definitions and vocabulary?
- Did you review the learning objectives before taking any quizzes?
- If the quizzes are computerized, did you study your quiz results?
- After reviewing quiz results and re-study, did you retake the quiz again if available?

All of this requires time and effort on your part as the student in any course (distance education or on-campus). You need several study periods a week to learn the materials in any course. The purpose of a quiz is for you to self assess your understanding of the materials. If your learning is not complete, **change or modify your learning habits**.

Interact with the Other Students

In a normal classroom students interact with each other. They often form study groups with other students and meet regularly

to help each other study materials. These interactions in most cases are essential to the learning process. If your only interaction is by private conversation or private email with the instructor, you are not fully participating in the course. For distance education students, most learning systems (such as Blackboard Vista) provide several tools to create this interaction. They typically include announcements, discussion list, email and chat tools.

Don't Procrastinate and Don't Get Behind

What should you do if you get behind? Plan **regular study periods.** The lack of regular study periods is most likely the reason for why you got behind. Plan when you will do **extra study periods** in order to catch up.

Attend Class and Take Notes

Taking lecture notes and being able to review those note later when you are studying provides variety that is needed to learn material. Just writing the notes down more actively engages the brain, because you are listening and writing. But you need to arrange with at least two fellow classmates that you will all take notes and share notes with each other if absent. In addition to course materials, other administrative matters are discussed in class (such as the announcement of exam date change).

If you are taking a distance education course, you need to regularly enter the learning management system (such as Blackboard Vista) and review the announcements, discussion list postings and read (and answer if appropriate) email. Most distance education professors assume that anything he has communicated via these tools will have been read by the

student within 3 days. In short this means you are responsible for having read the items and completing any action requested.

REFERENCES

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

Study Habits that Build the Brain

KENNETH LEROY BUSBEE

Introduction

During the spring of 2008 the author, Kenneth Leroy Busbee, did some research with students taking a computer programming fundamentals course to determine if using 3×5 cards would improve student performance on exams. In short, it did! This was not a surprise, but it became obvious that most of us (faculty at all levels of education as well as students) have little understanding of how are brain builds understanding and long term memory.

Attached are several PowerPoint presentations that have been save in an Adobe PDF format. Please spend a few minutes reviewing the information provided. Hopefully it will help students to better learn the subjects they are studying.

Main Presentation

Link to: Study Habits that Build the Brain

Specific Topics

Link to: Reading the Textbook
Link to: Taking Lecture Notes
Link to: Using 3×5 Cards
Link to: Using the Flash Card Activity

REFERENCES

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