African Virtual University

Applied Computer Science: CSI 2103

STRUCTURED PROGRAMMING

Noela Jemutai Kipyegen

Foreword

The African Virtual University (AVU) is proud to participate in increasing access to education in African countries through the production of quality learning materials. We are also proud to contribute to global knowledge as our Open Educational Resources are mostly accessed from outside the African continent.

This module was developed as part of a diploma and degree program in Applied Computer Science, in collaboration with 18 African partner institutions from 16 countries. A total of 156 modules were developed or translated to ensure availability in English, French and Portuguese. These modules have also been made available as open education resources (OER) on oer.avu.org.

On behalf of the African Virtual University and our patron, our partner institutions, the African Development Bank, I invite you to use this module in your institution, for your own education, to share it as widely as possible and to participate actively in the AVU communities of practice of your interest. We are committed to be on the frontline of developing and sharing Open Educational Resources.

The African Virtual University (AVU) is a Pan African Intergovernmental Organization established by charter with the mandate of significantly increasing access to quality higher education and training through the innovative use of information communication technologies. A Charter, establishing the AVU as an Intergovernmental Organization, has been signed so far by nineteen (19) African Governments - Kenya, Senegal, Mauritania, Mali, Cote d'Ivoire, Tanzania, Mozambique, Democratic Republic of Congo, Benin, Ghana, Republic of Guinea, Burkina Faso, Niger, South Sudan, Sudan, The Gambia, Guinea-Bissau, Ethiopia and Cape Verde.

The following institutions participated in the Applied Computer Science Program: (1) Université d'Abomey Calavi in Benin; (2) Université de Ougagadougou in Burkina Faso; (3) Université Lumière de Bujumbura in Burundi; (4) Université de Douala in Cameroon; (5) Université de Nouakchott in Mauritania; (6) Université Gaston Berger in Senegal; (7) Université des Sciences, des Techniques et Technologies de Bamako in Mali (8) Ghana Institute of Management and Public Administration; (9) Kwame Nkrumah University of Science and Technology in Ghana; (10) Kenyatta University in Kenya; (11) Egerton University in Kenya; (12) Addis Ababa University in Ethiopia (13) University of Rwanda; (14) University of Dar es Salaam in Tanzania; (15) Universite Abdou Moumouni de Niamey in Niger; (16) Université Cheikh Anta Diop in Senegal; (17) Universidade Pedagógica in Mozambique; and (18) The University of the Gambia in The Gambia.

Bakary Diallo

The Rector

African Virtual University

Production Credits

Author

Noela Jemutai Kipyegen

Peer Reviewer

Victor Odumuyiwa

AVU - Academic Coordination

Dr. Marilena Cabral

Overall Coordinator Applied Computer Science Program

Prof Tim Mwololo Waema

Module Coordinator

Robert Oboko

Instructional Designers

Elizabeth Mbasu

Benta Ochola

Diana Tuel

Media Team

Sidney McGregor

Barry Savala

Edwin Kiprono

Kelvin Muriithi

Victor Oluoch Otieno

Michal Abigael Koyier

Mercy Tabi Ojwang

Josiah Mutsogu

Kefa Murimi

Gerisson Mulongo

Copyright Notice

Supported By

AVU Multinational Project II funded by the African Development Bank.

Table of Contents

Course Overview

Welcome to Structured Programming

This is your second programming course after principles of computer programming. Now we want to take programming even further, as we move closer and closer to becoming 'Gurus' in software development. To achieve this, we shall start by learning programming using a structural language. Structured programming aims at programs that can perform critical file operations. This will greatly help you to easily learn other programming paradigms like object oriented programming languages, such as Java. We shall therefore, look at general overview of structured programming, language's environment which is C, operators, data types, control structures, functions, arrays, pointers, C structures and file processing

Prerequisites

Principles of Programming

Materials

The materials required to complete this course are:

- Computers
- Internet connection
- Recommended reading material

Course Goals

Upon completion of this course the learner should be able to:

1. describe various data types in a computer program
2. implement a program using control structures, loops and functions
3. design a solution by applying data structures
4. explain the dynamics of computer memory with the use of pointers
5. write programs that can perform critical file operations

Units

•Unit 0: Pre-Assessment

This is a pre-assessment unit. It reminds and tests you on some of the concepts that you need and are assumed in this module. This module will assess basic computer programming skills including problem solving techniques, operators, and control structures.

•Unit 1: The overview of structured programming

This unit introduces structured programming by looking at the definition, history, and elements. It also introduces C language including history of C language, C language environment, data types, variables, constants and operators. It also incorporates a lab (practical).

•Unit 2: Control structure and functions

This unit looks at conditional structures which include if, if/else, nested, nested if else statements if and multi-section structures (switch statements) as well as jump statements. It also considers loops or repetition structures like while, do/while, for and nested for statement. Lastly the unit discusses functions and incorporates a lab (practical).

•Unit 3: Data Structures

This unit discusses three kinds of data structures such as arrays- one dimensional arrays and multidimensional including a two dimensional arrays, pointers, and strings. It also incorporates a lab (practical).

•Unit 4: Structures, unions and File processing

This unit looks at three items: C structures, C unions looking at the definition and how to apply in programming, and file processing-which looks at the definitions, creating, opening reading and closing a file. It also incorporates a lab (practical).

Assessment

Formative assessments, used to check learner progress, are included in each unit.

Summative assessments, such as final tests and assignments, are provided at the end of each module and cover knowledge and skills from the entire module.

Summative assessments are administered at the discretion of the institution offering the course. The suggested assessment plan is as follows:

1	Four unit assignment of varying complexity and weights for the four units	20%
3	Mid Term exam	10%
4	Final exam	70%

Schedule

Unit	Unit title	Activities	Estimated time
0	Pre-assessment		2 hours
1	The overview of structured programming	Activity 1.1: Introduction to structured programming	5 hours
		Activity 1.2: History and structure of C language	5 hours
		Activity 1.3: Data types, variables, constants and operators	8 hours
		Lab 1.1: compiler Installation	2 hours
		Lab 1.2: Practicing use of variables, operators, input and output functions	8 hours
2	Control structure and functions	Activity 2.1: Conditional structures	6 hours
		Activity 2.2: Loops/Repetition	6 hours
		Activity 2.3: Functions	5 hours
		Lab2.1: Conditional structures/loops/ functions	11 hours
3	Data Structures	Activity 3.1:Arrays	5 hours
		Activity 3.2: Strings	6 hours
		Activity 3.3: Pointers	8 hours
		Lab 3.1: Arrays/Strings/Pointers	18 hours
4	Structure/Unions and File processing	Activity 4.1: Structures	5 hours
		Activity 4.2: Union	8 hours
		Activity 4.3: File processing	9 hours
5	Final Exams	Final assessment	3 Hours
		TOTAL	120 Hours

Readings and Other Resources

The readings and other resources in this course are:

Unit 1

Required readings and other resources:

- Deitel, P. and Deitel, H. (2016). C How to program with an introduction to C++. 8/e. Pearson Education.

(Pages and chapters to be referenced on this book are indicated in the content)

- Chhabra, K. J. (2010). C programming concepts with problems and
- solutions. Published by Tata McGraw Hill Education Private Limited. New Delhi pp1-86.
- McGrath, M. (2014). C programming in easy steps. 4th ed. Easy Steps Limited in UK. page 7-64
- https://scs.senecac.on.ca/~btp100/pages/content/sarra.html
- http://www.le.ac.uk/users/rjm1/cotter/page_62.htm
- http://beginnersbook.com/2014/01/2d-arrays-in-c-example/

Unit 2

Required readings and other resources:

- Deitel, P. and Deitel, H. (2016). C How to program with an introduction to C++. 8/e. Pearson Education.

(Pages and chapters to be referenced on this book are indicated in the content)

- Chhabra, K. J. (2010). C programming concepts with problems and
- solutions. Published by Tata McGraw Hill Education Private Limited. New Delhi. pp93-171
- https://scs.senecac.on.ca/~btp100/pages/content/sarra.html
- http://www.le.ac.uk/users/rjm1/cotter/page_62.htm
- http://beginnersbook.com/2014/01/2d-arrays-in-c-example/

Unit 3

Required readings and other resources:

- Deitel, P. and Deitel, H. (2016). C How to program with an introduction to C++. 8/e. Pearson Education.

(Pages and chapters to be referenced on this book are indicated in the content)

- Chhabra, K. J. (2010). C programming concepts with problems and solutions. Published by Tata McGraw Hill Education Private Limited.
- New Delhi. pp179-261
- https://scs.senecac.on.ca/~btp100/pages/content/sarra.html
- http://www.le.ac.uk/users/rjm1/cotter/page_62.htm
- http://beginnersbook.com/2014/01/2d-arrays-in-c-example/

Unit 4

Required readings and other resources:

- Deitel, P. and Deitel, H. (2016). C How to program with an introduction to C++. 8/e. Pearson Education.

(Pages and chapters to be referenced on this book are indicated in the content)

- Chhabra, K. J. (2010). C programming concepts with problems and
- solutions. Published by Tata McGraw Hill Education Private Limited.
- New Delhi. pp289-315
- https://scs.senecac.on.ca/~btp100/pages/content/sarra.html
- http://www.le.ac.uk/users/rjm1/cotter/page_62.htm
- http://beginnersbook.com/2014/01/2d-arrays-in-c-example/

Unit 0. Pre-Assessment

Unit Introduction

This initial test assess learners on basic computer programming skills based on previous module. It is also meant to refresh students' memory on important elements of the computer. This will allow learners to quickly grasp other subsequent units of this course.

Unit Objectives

Upon completion of this unit you should be able to:

- explain what programming is and how it is used in problem solving
- analyze problems
- design problem's solutions using proper methodology

Key Terms

Programming: is the process of writing a sequence of instructions to be executed by a computer to solve a problem.

Programming language: is a set of instructions used for writing computer programs

Programmer: people who write programs using different programming language to communicate instructions to the computer.

Problem: A perceived gap between the existing state and a desired state, or a deviation from a norm, standard, or status quo (http://www.businessdictionary.com/definition/problem.html#ixzz40yqd6LBD)

Software analysis: is the process of determining user expectations for a new or modified product (http://searchsoftwarequality.techtarget.com/definition/requirements-analysis). It entails refining requirements to eliminate errors, inconsistencies and ambiguity.

Software design: Defines how a software system should be produced in order to make it functional, reliable, reasonably easy to modify, understand and maintain.

Algorithm: is a well-defined procedure that allows a computer to solve a problem. Another way to describe an algorithm is a sequence of unambiguous instructions (http://study.com/academy/lesson/what-is-a-computer-algorithm-design-examples-optimization.html).

Flowchart: is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows.

Unit Assessment

Check your understanding!

1. Define the following terms: (12marks, 2 each)

i. A variable

ii. Computer programming

iii. An algorithm

iv. Unary operator

v. Programming language

vi. Programming paradigm

2. List two advantages and two disadvantages for each of the following; (4marks)

i. Low level language

ii. High level languages

3. The details look different in different languages, but a few basic instructions appear in just about every language. Name five common instructions(5 marks)

4. A programmer has to use a programming language when developing a program but the choice of the language is subject to many considerations. List any four factors that influence the considerations of a language. (4 marks)

5. An algorithm must be efficient. Briefly elaborate (2mark)

6. Design a flowchart for the statement below (4marks)

```
for(b=1; b<=2; ++b)
    printf("%d", b);
```

7. Read the statement following then answer questions below it.

Mwalimu tasked you to provide a solution to a problem. The problem is; print "large" if x is greater than 10, otherwise print small.

i. Design a solution to this problem using an algorithm (3marks)

ii. Use a flowchart to represent the algorithm (4marks)

8. You were tasked to develop a software for the following problem statement **Give the sum of the integers from 20 to 150.**

i. Analyse the requirements based on the problem statement, what does analysis entail? (5 marks)

ii. Design solution using a pseudocode and flowchart, what does design entail (10 marks)

9. Certain rules apply when inventing an identifier. Give any two rules (2 marks)

10. Differentiate between compilers and interpreters (2 marks)

11. Write a single C statement for i to xi below that: (11marks-1 each)

i. Declare variable x to an integer

ii. Declare variable y to an character

iii. Input integer variable x using c statement

iv. Input integer variable y using c statement

v. Initialize integer i to 1

vi. Initialize integer variable p to the value in x

vii. Initialize x to the value in i

viii. Multiply variable v by x and assign the result to v

ix. Increment variable j by 1

x. Test p to see if it is less than or equal to x

xi. Output integer variable p using c statement

12. List any four types of operators (4 marks)

Instructions

Answer all the questions. If you score:

Less than 60% marks then you will be required to refer to the reference material for revision.

60-69% you need to do moderate revision

Above 70% minimal revision and move on to unit one.

Grading Scheme

The score for each question was indicated. Since this was part of your revision, apply the instructions.

Answers:

1. Definitions

   i. A variable is a memory space for storing values/placeholders

   ii. Computer programming is the act of writing instructions to be executed by computer to solve a particular problem.

   iii. An algorithm is a finite set steps that are followed to complete a task or an ordered set of unambiguous executable steps, defining a terminating process.

   iv. Unary operator is an operator that operates on one operand

   v. Programming language is a set of instructions used for writing computer programs

   vi. Programming paradigm Is a programming style or defines how a particular language is structured.

2. i. Low level language- advantages: short execution time (fast), no interpretation, disadvantages: prone to errors, machine dependent, difficult to program

   ii. High level languages- advantages: easy to program, understand, learn, debug and document disadvantages: must be interpreted, long execution time

3. - input: Get data from the keyboard, a file, or some other device.

   - output: Display data on the screen or send data to a file or other device.

   - arithmetic: Perform basic arithmetical operations like addition and multiplication.

   - conditional execution: Check for certain conditions and execute the appropriate sequence of statements.

   - repetition: Perform some action repeatedly, usually with some variation.

4. - company policy,

   - suitability to task,

- availability of third-party packages, or

- individual preference.

5. To save on space and time

6. flowchart

7. i. Start

Read x

If x is greater than 10

Output “large”

Else print “small”

end

ii. flowchart

8. i. analysis involves description of input/output and the operations to be performed by the system

ii. algorithm

start

Accept an integer for n

initialize sum to 0

for each integer i in the range 1 to n

assign sum+i to sum

return the value of sum

end

or

start

Accept an integer for n

return the value of n*(n+1)/2

end

flowchart

9.Do not use language keywords

start with a letter or an underscore

Do not start with numeric value

10. A compiler program translates the instructions of a high level language to a machine level language. While an interpreter is system software that is fairly similar to a compiler. It is also used to convert a high level language program to a machine level language program. The difference is that, it analyses every line of source program and if any error, reports them instantaneously and stops further translation. Interpreters occupy less space than a compiler. The interpreter is 5-25 time slower that the compiler. The disadvantage with interpreter is that, whenever a high level program is executed, it has to translate each and every time but a compiler translates just once and produces an object file that can be executed from then onwards.

11. i). int x;

ii).char y;

iii). scanf("%d", &x);

iv). scanf("%c", & y);

v). i=1

vi). p=x;

vii). x = i;

viii). v *= x ;

ix). ++ j or j+= 1 or j=J+1;

x). if (p<=x)

xi). printf("%d", p);

12. - Arithmetic operators

Assignment operators

Logical operators

Increment/decrement operators

Relational operators

Unit 1. The overview of structured programming

Unit Introduction

"During 1950s and 1960s few people actually thought about the question of developing a general method for organizing a program. Instead, programs were written much the way you might, off the top of your head, give someone instructions on how to change a tire or, worse, build a house. When a program did not work, it was simply "patched" (fixed), error by error, until the programmer felt that the he had found them all. Usually, however, errors continued to appear even after the program was released" (http://users.csc.calpoly.edu). This method was so unstructured. The unstructured methodology was applied in first and second generation computers. When the third generation emerged in late 1960s there was a problem of "hit and miss". This was as a result of the hardware being more powerful than the software. The software industry was unable to improve their software the same way hardware industry did. To utilize the power of the computer, programs had to contain thousands of instructions, written by many different developers. By this time there was no structured way of organizing programs programs that handle tasks of high magnitude. This resulted to software crisis. The crisis was characterized by late software delivery, over budget and unresponsiveness (or not meeting the desired functionalities). Therefore, this unit looks at the history and general overview of structured programming, structured language (in this case C) and its environment.

Unit Objectives

Upon completion of this unit you should be able to:

- Explain the meaning of structured programming
- Describe the history and structure of C language
- Apply C language variable, data types and operator.

Key Terms

Structured programming: Structured programming is a method of writing a computer program that uses (1) top-down analysis for problem solving, (2) modularization for program structure and organization, and (3) structured code for the individual modules (http://users.csc.calpoly.edu/~jdalbey/308/Resources/StructuredProgramming).

Operators: An operator is a symbol that tells the compiler to perform certain mathematical or logical manipulations (http://www.studytonight.com/c/operators-in-c.php).

Variables: is a storage location in the memory

code: also known as source code is any collection of computer instructions (possibly with comments) written using some human-readable computer language, usually as text. (https://en.wikipedia.org/wiki/Source_code).

Identifier: is the name of a variable

Learning Activities

Activity 1.1 - Introduction to structured programming

Introduction

As we mentioned earlier, programming began in 1960s with unstructured methodologies until late 1964 when structured programming was mentioned. This activity will define structured programming and give the history and rationale of structured programming

History and rationale of Structured programming

(adapted from http://users.csc.calpoly.edu/~jdalbey/308/Resources/StructuredProgramming).

The history of structured programming began in 1964 at an international colloquium held in Israel. There Corrado Bohm and Guiseppe Jacopini presented a paper (in Italian] that proved mathematically that only three "control structures" were necessary to write any program. This theorem, and control structures in general will be discussed in subsequent units. Suffice it to say here that the work of Bohm and Jacopini made the GOTO statement unnecessary in computer programming. During the mid-1960s virtually no programmer could even conceive of a program written without GOTO statements. Therefore, it is no surprise that even after an English translation of their paper in the trade journal Communications of the ACM in 1986, the theorem of Bohm and Jacopini and its implications were almost entirely ignored in the United States. The turning point, however. occurred in 1968 when Edsger Dijkstra of the Netherlands published a letter to the editor in the Communications of the ACM. This letter was given the appropriate title, "Go To Statement Considered Harmful." For over twenty years Dijkstra crusaded for a better way of programming—a systematic way to organize programs, called structured programming. It can be used with profit for any program but pays enormous dividends on very large programs. Edward Yourdon is another important name in the history of structured programming. By giving seminars on the subject in the mid-1970s, he was the individual most responsible for popularizing the method in the United States. However, its widespread acceptance would probably not have occurred were it not for the tremendous success of the now famous New York Times project, which was completed in 1972.

This was the first major project using structured programming and its first great success story. This project, developed for the New York Times by a programming team at IBM under the direction of Harlan Mills, was a system to automate the newspaper clipping file. Using a list of index terms, users could browse through abstracts of all the paper's articles and then retrieve the full-length articles of their choice from microfiche for display on a terminal screen.

The task took 22 months, included about 83 000 lines of code, and involved approximately 11 man-years of effort. The file processing system passed a week of acceptance testing without error and ran for 20 months until the first error was detected. in the first 13 months. only one program error resulted in system failure. The system-control programmers achieved about 10 000 lines of source code and one error per man~year.

In the entire system, only 21 errors were found during five weeks of acceptance testing, and only 25 additional errors were discovered during the first year of the system's operation. Moreover, it was delivered under budget and ahead of schedule." Contrast these statistics with those mentioned previously for the development of IBM's 0S/360 operating system and the overall average output per programmer per day (read http://users.csc.calpoly.edu/~jdalbey/308/Resources/StructuredProgramming).

These results shocked the programming community. Software developers began to pay attention to what Dijkstra had been saying and writing. By the mid- to late 1970s. structured programing was being used for everything from home computers

to multi million-dollar defense projects. IBM certainly became a believer. Mills, IBM's chief programmer on the New York Times project, says "I was surprised myself at how big an impact that project made. It was as if the world were waiting for something like that to happen. We [now] use it [structured programming] at IBM across the whole company. Hardly anyone can survive without it."

Indeed, structured programming has been called a "revolution in programming"' and "one of the most important advances in computer software of the past two decades." This is not to say, however, that old habits die easily. Many who learned to program prior to Dijkstra's work spurn the concept even today. Jim Horning, a computer scientist at Xerox's Palo Alto Research Center explains, "There are some people in this laboratory who read everything he [Dijkstra] writes and are extremely grateful for it, and there are others who would not be willing to have him come visit us. He tends to polarize people." Nevertheless, both academia and industry are turning more and more to the philosophy and techniques of structured programming. "Today it is safe to say that virtually all practitioners [of the art of programming] at least acknowledge the merits of the discipline [of structured programming], and most practice it exclusively.

What is structured programming?

Now after a long history of structured programming, we ask ourselves, what exactly is structured programming? Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of subroutines, block structures and conditional statements which minimizes the use of simple tests and jumps such as the goto statement which could lead to "spaghetti code" which is difficult both to follow and to maintain. Structured programming is a method of writing

a computer program that uses (1) top-down analysis for problem solving, (2) modularization for program structure and organization, and (3) structured code for the individual modules (http://users.csc.calpoly.edu/~jdalbey/308/Resources/StructuredProgramming).

An unstructured program is a procedural program where the statements are executed in sequence as written. But this type of programming uses the goto statement. A goto statement allows control to be passed to any other place in the program. When a goto statement is executed, the sequence continues from the target of the goto. Thus to understand how a program works, you have to pretend to execute it. This means that it is often difficult to understand the logic of such a program. Some program compilers cross-index where a goto connects to, making it practical to rapidly navigate the source code. However, it was a common practice in some programming languages to use a variable in association with where the goto goes, making automated indexing impractical. There are similar problems in some structured programming languages, such as how foreign language views are implemented, to permit many people to view the same computer data, in their human language. This contrasts with the idea of using some form of abstraction to understand how a program works - as in structured programming. For this reason, it was suggested by Dijkstra that the goto statement should be banned as explained in the history of structured programming (https://en.wikibooks.org/wiki/Computer_Programming/Structured_programming).

Elements of structured programming

Control structures

Following the structured program theorem, all programs are seen as composed of control structures:

- "Sequence"; ordered statements or subroutines executed in sequence.
- "Selection"; one or a number of statements is executed depending on the state of the program. This is usually expressed with keywords such as if..then..else..endif.
- "Iteration"; a statement or block is executed until the program reaches a certain state, or operations have been applied to every element of a collection. This is usually expressed with keywords such as while, repeat, for or do..until. Often it is recommended that each loop should only have one entry point (and in the original structural programming, also only one exit point, and a few languages enforce this).
- "Recursion"; a statement is executed by repeatedly calling itself until termination conditions are met. While similar in practice to iterative loops, recursive loops may be more computationally efficient, and are implemented differently as a cascading stack.

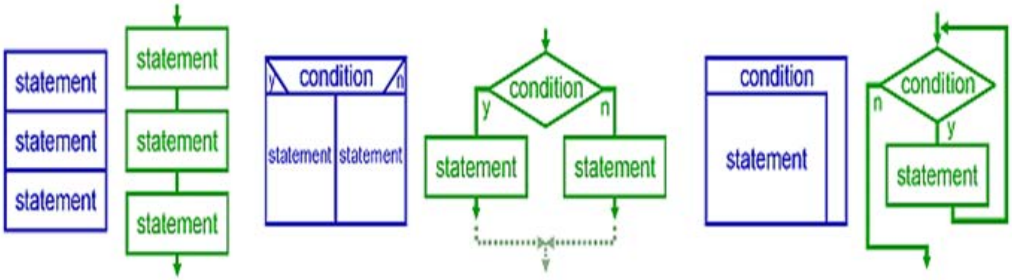


Figure 1.1 Graphical representations of the three basic patterns Source:

https://en.wikipedia.org/wiki/Structured_programming

Subroutines

Subroutines are callable units such as procedures, functions, methods, or subprograms are used to allow a sequence to be referred to by a single statement.

Blocks

Blocks are used to enable groups of statements to be treated as if they were one statement. Block-structured languages have a syntax for enclosing structures in some formal way, such as an if-statement bracketed by if..fi as in ALGOL 68, or a code section bracketed by BEGIN.. END, as in PL/I, whitespace indentation as in Python or the curly braces {...} of C and many later languages.Read http://users.csc.calpoly.edu/~jdalbey/308/Resources/StructuredProgramming for more information on elements of structured programming.

Advantages of structured programming

1. Programs are more easily and more quickly written.
2. Programs have greater reliability.
3. Programs require less time to debug and test.
4. Programs are easier to maintain.

Read http://users.csc.calpoly.edu/~jdalbey/308/Resources/StructuredProgramming for more information on the advantages of structured programming. Also textbook by Deitel P. and Deitel H. (2016). C How to program. 8/e. Pearson Education. Page 69-88.

Conclusion

Structured programming was necessitated by crisis in software industry where software did not meet the functionalities, it was delivered late and exceeded the budget. Structured programming tries to minimize the use of goto statement by use of three control structures elements; control structures, subroutines and blocks. It has various advantages unlike its predecessor (unstructured programming). With structured programming, programs are more easily and more quickly written, programs have greater reliability, programs require less time to debug and test, and programs are easier to maintain.

Assessment

1. Structured programming was defined as method of writing a computer program that uses three elements (1) top-down analysis for problem solving, (2) modularization for program structure and organization, and (3) structured code for the individual modules. Discuss these elements (15 marks)
2. Differentiate between structured and unstructured programming (4marks)
3. List any five languages that fall under structured programming paradigm (5 marks)

Activity 1.2 - History and structure of C language

Introduction

C language falls under structured programming paradigm and has various benefits when used as a choice programming language. Therefore, this course will use C language to reinforce the concepts of structured programming languages. Also, this unit will introduce C language, history, structure, variable and data types.

History of C language

The field of computing as we know it today started in 1947 with three scientists at Bell Telephone Laboratories—William Shockley, Walter Brattain, and John Bardeen—and their groundbreaking invention: the transistor. In 1956, the first fully transistor-based computer, the TX-0, was completed at MIT. The first integrated circuit was created in 1958 by Jack Kilby at Texas Instruments, but the first high-level programming language existed even before then.

"The Fortran project" was originally developed in 1954 by IBM. A shortening of "The IBM Mathematical Formula Translating System", the project had the purpose of creating and fostering development of a procedural, imperative programming language that was especially suited to numeric computation and scientific computing. It was a breakthrough in terms of productivity and programming ease (compared to assembly language) and speed (Fortran programs ran nearly as fast as, and in some cases, just as fast as, programs written in assembly). Furthermore, Fortran was written at a high-enough level (and thus was machine independent enough) to become the first widely adopted programming language. The Algorithmic Language (Algol 58) was derived from Fortran in 1958 and evolved into Algol 60 in 1960. The Combined Programming Language (CPL) was then created out of Algol 60 in 1963. In 1967, it evolved into Basic CPL, which was itself, the base for B in 1969. Finally, B, the root of C, was created in 1971.

C was the direct successor of B, a stripped down version of BCPL, created by Ken Thompson at Bell Labs, that was also a compiled language - User's Reference to B, used in early internal versions of the UNIX operating system. As noted in Ritchie's C History : "The B compiler on the PDP-7 did not generate machine instructions, but instead 'threaded code', an interpretive scheme in which the compiler's output consists of a sequence of addresses of code fragments

that perform the elementary operations. The operations typically — in particular for B — act on a simple stack machine". Thompson and Dennis Ritchie, also working at Bell Labs, improved B and called the result NB. Further extensions to NB created its logical successor, C. Most of UNIX was rewritten in NB, and then C, which resulted in a more portable operating system.

The portability of UNIX was the main reason for the initial popularity of both UNIX and C. Rather than creating a new operating system for each new machine, system programmers could simply write the few system-dependent parts required for the machine, and then write a C compiler for the new system. Since most of the system utilities were thus written in C, it simply made sense to also write new utilities in C.

The American National Standards Institute began work on standardizing the C language in 1983, and completed the standard in 1989. The standard, ANSI X3.159-1989 "Programming Language C", served as the basis for all implementations of C compilers. The standards were later updated in 1990 and 1999, allowing for features that were either in common use, or were appearing in C++ (https://en.wikibooks.org/wiki/C_Programming/History).

Why use C language?

C has been used successfully for every type of programming problem imaginable from operating systems to spreadsheets to expert systems - and efficient compilers are available for machines ranging in power from the Apple Macintosh to the Cray supercomputers. The largest measure of C's success seems to be based on purely practical considerations:

1. the portability of the compiler;
2. the standard library concept;
3. a powerful and varied repertoire of operators;
4. an elegant syntax;
5. ready access to the hardware when needed;
6. and the ease with which applications can be optimised by hand-coding isolated procedures

C is often called a "Middle Level" programming language. This is not a reflection on its lack of programming power but more a reflection on its capability to access the system's low level functions. Most high-level languages (e.g. Fortran) provides everything the programmer might want to do already built into the language. A low level language (e.g. assembler) provides nothing other than access to the machines basic instruction set. A middle level language, such as C, probably doesn't supply all the constructs found in high-languages - but it provides you with all the building blocks that you will need to produce the results you want! (http://www2.le.ac.uk/projects/oer/oers/beyond-distance-research-alliance/introduction-to-c-programming)

Uses of C

C was initially used for system development work, in particular the programs that make-up the operating system.

Why use C? Mainly because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:

1. Operating systems
2. Language compilers
3. Assemblers
4. Text editors
5. Print spoolers
6. Network drivers
7. Modern programs
8. Databases
9. Language interpreters
10. Utilities

In recent years C has been used as a general-purpose language because of its popularity with programmers. It is not the world's easiest language to learn and you will certainly benefit if you are not learning C as your first programming language! C is 'trendy' - many well established programmers are switching to C for all sorts of reasons, but mainly because of the portability that writing standard C programs can offer (http://www2.le.ac.uk/projects/oer/oers/beyond-distance-research-alliance/introduction-to-c-programming).

Executing a C program

To execute a program in C language, you will need a C compiler. There are various variants of C compilers most of which are free like:

1. Code Blocks: http://www.codeblocks.org/downloads
2. Digital mars C/C++ compilers : http://www.digitalmars.com/
3. Dev-C++ compilers : http://www.bloodshed.net/dev/devcpp.html
4. minGw C compiler: https://sourceforge.net/projects/mingw/files/
5. Bloodshed Dev-C++: http://www.bloodshed.net/devcpp.html
6. Visual Studio Express: https://www.visualstudio.com/products/visual-studio-express-vs
7. Borland C++: http://www.mediafire.com/download/ad1e5uc8zg9ko9h/Borland+C%2B%2B+5.5+%2B+Sp1%2CSp2.exe
8. Turbo C: http://www.mediafire.com/download/ear7h0il04r6ib4/TurboC%2B%2B+for+Windows+7_v3.7.7.7major_release.zip

Among other C compilers.

We shall use Code Blocks for this module. Download the Compiler using the links then use its manual to understand how to use it. Dev Codeblocks looks as follows:

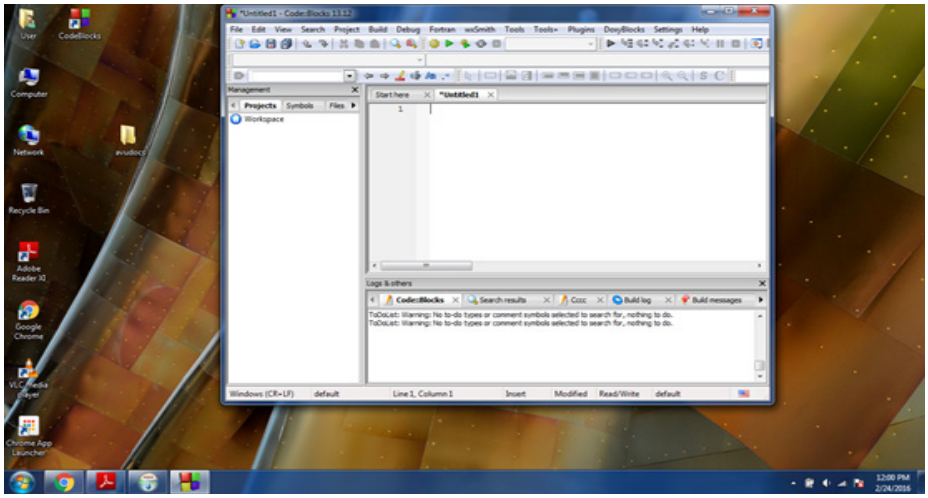


Figure 1.2 Code Blocks window

The Edit-Compile-Link-Execute Process

Developing a program using C compiler entails four steps:

1. editing (or writing) the program
2. compiling
3. linking
4. executing

Editing

You write a computer program with words and symbols that are understandable to human beings. This is the editing part of the development cycle. You type the program directly into a window on the screen and save the resulting text as a separate file. This is often referred to as the source file (you can read it with the TYPE command in DOS or the cat command in unix). The custom is that the text of a C program is stored in a file with the extension .c for C programming language

Compiling

You cannot directly execute the source file. To run on any computer system, the source file must be translated into binary numbers understandable to the computer's Central Processing Unit (for example, the 80*87 microprocessor). This process produces an intermediate object file - with the extension .obj, the .obj stands for Object.

Linking

The first question that comes to most people's minds is Why is linking necessary? The main reason is that many compiled languages come with library routines which can be added to your program. Theses routines are written by the manufacturer of the compiler to perform a variety of tasks, from input/output to complicated mathematical functions. In the case of C the standard input and output functions are contained in a library (stdio.h) so even the most basic program will require a library function. After linking files, executable file with the extension .exe is created.

Executable files

Thus the text editor produces .c source files, which go to the compiler, which produces .obj object files, which go to the linker, which produces .exe executable file. You can then run .exe files as you can other applications, simply by typing their names at the DOS prompt or run using windows menu (http://www2.le.ac.uk/projects/oer/oers/beyond-distance-research-alliance/introduction-to-c-programming)

You are required to read more on program execution from the textbook by Deitel P. and Deitel H. (2016). C How to program. 8/e. Pearson Education. Page 15-17.

Structure of C program

A simple C program looks as follows:

Explanation

a. Comment

The first line is a comment the code. Commenting involves placing Human Readable Descriptions inside of computer programs detailing what the Code is doing. Proper use of commenting can make code maintenance much easier, as well as helping make finding bugs faster. Further, commenting is very important when writing functions that other people will use. Remember, well documented code is as important as correctly working code. Commenting is the "art" of describing what your program is going to do in "high level" English statements. Commenting is best done before actually writing the code for your program.

Comments are specially marked lines of text in the program that are not evaluated. There are usually two syntactic ways to comment:

a. Single line comment and, as implied, only applies to a single line in the "source code" (the program).

For example:

```
//First c program

//Learning variable
```

Each new line must start with a double forward slash when using a single line comment. If you have three comments on separate lines, then it means you will have double forward slash preceding each of the three comments.

b. Block comment or multiple line comment which usually refers to a paragraph of text. A block comment has a start symbol and an end symbol and everything between is ignored by the computer.

For example:

```
/* First c program*/
```

or

```
/* First c program, see
   it is runni..iiing and
    i feel like a guru already! */
```

Notice how we introduced new lines without '*/ '. It means you open the comment with a single forward slash '/' append the asterisk '*' , write comments then close with an asterisk '*' and a single forward slash '/' (http://www.cs.utah.edu/~germain/PPS/Topics/commenting.html).

b. Header file

This line starts with #include statement. This statement is also known as pre-processor. The preprocessor is used to initialize program environment by linking it with the header files specified in the include statement. For example, in our code, #include links the program to stdio.h header file.

The #include preprocessor directive causes the compiler to replace that line with the entire text of the contents of the named source file (if included in quotes: "") or named header (if included in angle brackets: <>).

Header file is a collection of built-in functions that help us in our program. It contains definitions of functions and variables which can be incorporated into any C program by the pre-processor #include statement. Standard header files are provided with each compiler, and cover a range of areas like string handling, mathematical functions, data conversion, printing and reading of variables. To use any of the standard functions, the appropriate header file must be included. This is done at the beginning of the C source file. For example, the line #include <stdio.h> is responsible for the printf() and scanf() functions in a program. (http://www.studytonight.com/c/first-c-program.php)

c. Main() function

main() function is a function that must be used in every C program. It marks the starting point for program execution. This means, C program will always start the execution from main(). The int in front of main() function is the return type of main() function (this will be discussed later). The '()' brackets is used to define a function. Thus this uniquely identifies this statement as a function and so many other functions that we will create later in this course.

d. Function definition/body

The curly braces { } just after the main() function encloses the body of main() function. Inside the curly braces are functions defining what the program is executing. This is the reason it is known as function definition. In our code, the function body contain:

```
printf("Hello,World\n");

                return 0;
```

This program contains just one statement: a function call to the standard library function printf(), which prints a character string to standard output (usually the screen). Note, printf() is not a part of the C language, but a function provided by the standard library declared in header stdio.h). The standard library is a set of functions mandated to exist on all systems conforming to the ISO C standard. In this case, the printf() function takes one argument (or input parameter): the string constant "Hello World!\n". The \n at the end of the string is an escape character to start a new line. Escape characters provide a mechanism for representing hard-to-type or invisible characters (e.g., \t for tab, \b for backspace, \" for double quotes). Finally, the statement is terminated with a semicolon (;). C is a free-form language, with program meaning unaffected by whitespace in most circumstances. Thus, statements are terminated by ; not by a new line.

When a function completes, the program returns to the calling function. In the case of main(), the program terminates and control returns to the environment in which the program was executed. The integer return value of main() indicates the program's exit status to the environment, with 0 meaning normal termination (http://www2.le.ac.uk/projects/oer/oers/beyond-distance-research-alliance/introduction-to-c-programming)

The same program written using CodeBlocks C compiler:

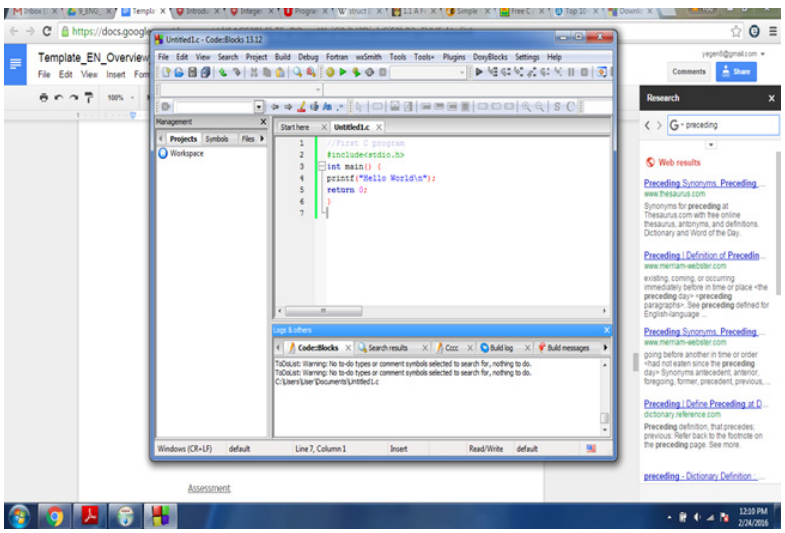


Figure 1.2 Sample program on CodeBlocks

After writing the program as shown above, click on Build menu then select compile and then run.

Reference Deitel and Deitel's textbook for more on C structure:

Deitel H. (2016). C How to program. 8/e. Pearson Education. Page 40.

Conclusion

This activity looked at the history of C language. It was clearly noted that that C is a robust programming language and can be used to develop Operating Systems, language Compilers, assemblers, text editors, print spoolers, network drivers, modern programs, data bases, language Interpreters, utilities among others. C also has various strengths which include practical considerations, the portability of the compiler, the standard library concept, a powerful and varied repertoire of operators, an elegant syntax; ready access to the hardware when needed, and the ease with which applications can be optimised by hand-coding isolated procedures. We also looked at the structure of C language and saw how easy it is to learn and understand.

Assessment

Research on:

1. Program execution has four steps. Name the errors that can occur during;

i. compilation time (4marks)

a. ii. execution time (4marks)

2. When an error occurs what do you do? (2marks)

3. C is considered a middle level language. What does it mean? (2marks)

Lab work 1.1: Compiler installation and Practice

Objective of the lab is to:

- Make sure that each student has C compilers in their computers

Details of the lab exercises/activities for compiler installation

Objectives of the exercise is to

- Download and
- Install C compiler

Resources required

- Computer
- The internet

Time required:

30 minutes

Description of lab exercise/activity

Download C compiler (see activity 1.2 for the links), install into your computer and launch the program. Send a screen snapshot of your compilers window to your instructor. Answer the following and submit with the screen snapshot (10marks)

- Which C compiler did you download?
- Elaborate the download and installation process.

Results and submission requirements

You will be required to submit your lab activity as per your instructor's direction

Assessment criteria

It has been assigned marks

At the end it will be computed with other labs to 10%

References or key links

See activity 1.2 for the links on links to compilers.

Activity 1.3 - Data types, variable, constants and operators.

Introduction

In activity 1.2, we wrote a simple program to illustrate the structure of C. A program is not limited to strings or displaying strings but it is able to process and display other values as well. Now, let us learn how to make use of the memory and the various types of data that we can store in the memory.

Remember, C is a high level language which means we do not need to directly use the memory addresses in order to store or locate data. This is because it provides us with the cheapest way of accessing and manipulating our data in the memory. We can accomplish this with the aid of variables- identifiers (a name that uniquely identifies a particular location in the memory). In order to understand variable in C programming, we need to first learn data types.

Data types in C Language

Data types define the kind of data that we enter into our programs. C language has some predefined set of data types to handle various kinds of data that we use in our program and they have have different storage capacities. There are two types of data types:

1. Primitive/ primary data types

These are basic data types provided by a programming language as a basic building block or we can say that they are data type for which the programming language provides built-in support. For example integers (int), floating (float), characters (char) and void

2. Derived/Secondary data types

Derived data types are object types which are aggregates of one or more types of basic data types. The most common derived data types are pointers, arrays, structures and unions (https://arcc.uwyo.edu/wiki/c-programming-language-derived-data-types). We shall discuss them later.

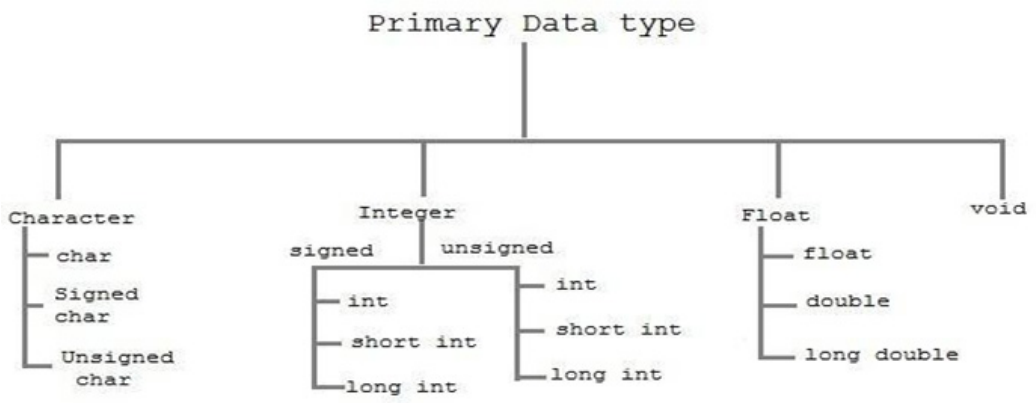


Figure 1.3 Primary data types

Integer data type

Integer type are used to store whole numbers. Size and range of Integer type on 16-bit machine are as shown in table 1.1

Table 1.1 Integer data types

Type	Size(bytes)	Range
int or signed int	2	-32,768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
long int or signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

Floating data type

Floating types are used to store real numbers. Their sizes and range of Integer type on 16-bit machine are as shown in table 1.2

Table 1.2 Floating data type

Type	Size(bytes)	Range
Float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

Character data type

Character types are used to store characters value. Their sizes and range of Integer type on 16-bit machine are as shown in table 1.3

Table 1.3 Character data type

Type	Size(bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

Void type

It is often used with functions. It means no value. We will use it when we get to functions. Variables

A variable is a memory location where we can store our data. Identifier is the name of that location see figure 1.4. Figure 1.4 Simple view of memory

Unlike constant, variables are changeable, we can change value of a variable during execution of a program. A programmer can choose a meaningful variable name. Example : average, height, age, total etc.

Rules to define variable name

1. Variable name must not start with a digit.
2. Variable name can consist of alphabets, digits and special symbols like underscore '_'
3. Blank or spaces are not allowed in variable name.
4. Keywords are not allowed as variable name.

Declaration of variable

Always, variables need to be declared before they are used in the program. Declaration means variable definition. This declaration is very important as:

1. It tells the compiler what the variable name is.

2. It specifies what type of data the variable will hold.

for example;

```
int a;

float b;

char c;
```

Means we have declared a variable a as an integer, b as a float(real value) and c as a character. Now, our memory has;

Figure 1.5 Simple memory snapshot

Initializing variables

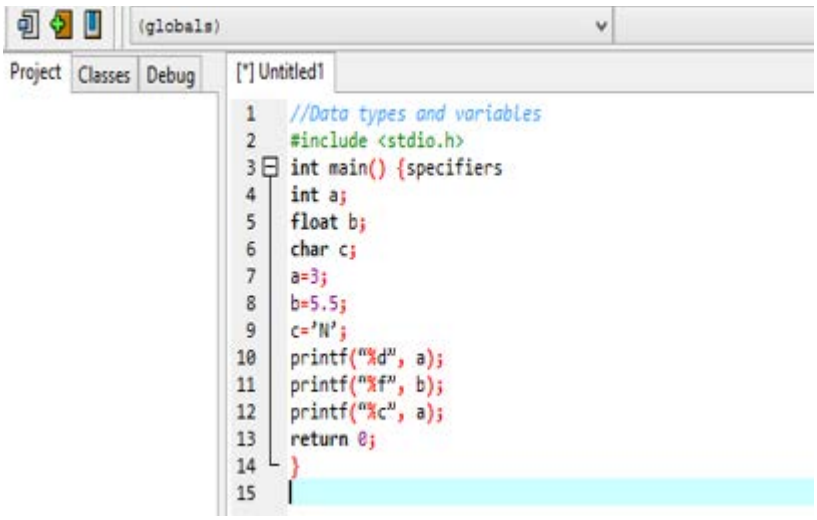
Initialization means assigning values to variables. For example;

```
a=3;

b=5.6;

c='N';
```

Now our memory as shown in figure 1.9

Figure 1.6 Initialized variable

Note: A character is enclosed in a single quotation mark ' '. Now let us write a simple program.



We have introduced new elements to the program; "%d, %f and %c". These are known as formatting specifiers. They are used to output and input values to a C program.

Table 1.4 The % Format Specifiers

Specifier	Type
%c	char single character
%d (%i)	int signed integer
%e (%E)	float or double exponential format
%f	float or double signed decimal
%g (%G)	float or double use %f or %e as required
%o	int unsigned octal value
%p	pointer address stored in pointer
%s	array of char sequence of characters
%u	int unsigned decimal
%x (%X)	int unsigned hex value

Constant Data Types

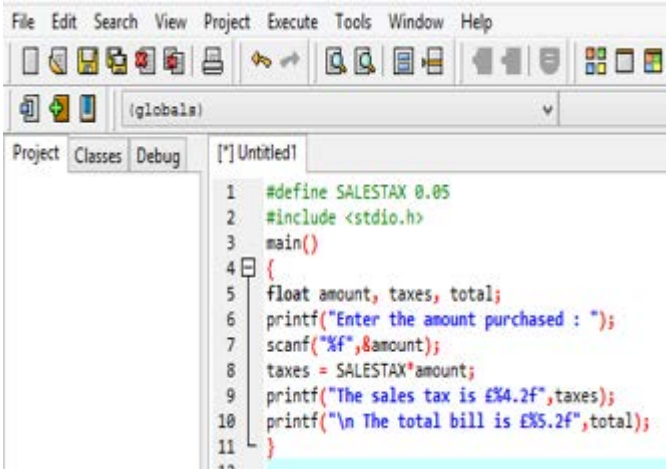
Constants refer to fixed values that may not be altered by the program. All the data types we have previously covered can be defined as constant data types if we so wish to do so. The constant data types must be defined before the main function. The format is as follows:

#define CONSTANTNAME value

for example:

#define SALESTAX 0.05

The constant name is normally written in capitals and does not have a semi-colon at the end. The use of constants is mainly for making your programs easier to be understood and modified by others and yourself in the future. An example program now follows:



The float constant SALESTAX is defined with value 0.05. Three float variables are declared amount, taxes and total. Display message to the screen is achieved using printf and user input handled by scanf. Calculation is then performed and results sent to the screen. If the value of SALESTAX alters in the future it is very easy to change the value where it is defined rather than go through the whole program changing the individual values separately, which would be very time consuming in a large program with several references. The program is also improved when using constants rather than values as it improves the clarity (http://www.le.ac.uk/users/rjm1/cotter/page_52.htm).

Read more on constants from the textbook by McGrath M. (2014). C programming in easy steps. 4th ed. Published by Easy Steps Limited in UK. page 40-48

Operators in C Language

An operator is a symbol that tells the compiler to perform certain mathematical or logical manipulations. They manipulate individual data items and returns a result. C supports a number of these operators. C operators can be categorized into two;

1. Unary operator

These operators operate or work on only one operand. Its form looks as shown;

operator operand;

For example:

```
-a, +b, -12, +16, ++c, --d etc
```

2. binary operators

These operators operate/work on two operands. Its form looks as shown;

operand1 operator operand2;

For example;

a + b

Types of operators in C,

- Arithmetic operators
- Relation operators
- Logical operators
- Bitwise operators
- Assignment operators
- Increment operators
- Conditional operators
- Special operators

Arithmetic operators

The following table shows all the basic arithmetic operators.

Table 1.5 Arithmetic operators

Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denominator
%	remainder of division
++	Increment operator increases integer value by one
--	Decrement operator decreases integer value by one

Relation operators

Relational operators test or defines some kind of relation between two entities. These include numerical equality (e.g., 5 = 5) and inequalities (e.g., 4 ≥ 3).The following table shows all relational operators in C.

Table 1.6 Relational operators

Operator	Description
==	Check if two operands are equal
!=	Check if two operands are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check if operand on the left is smaller than right operand
>=	check left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal to right operand

Logical operators

C language supports following logical operators. We shall use them later. For example a=1 and b=0,

Table 1.7 Logical operators

Operator	Description	Example
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

Bitwise operators

A bitwise operation operates on one or more bits patterns or binary numerals at the level of their individual bits. It is a fast, primitive action directly supported by the Processor, and is used to manipulate values for comparisons and calculations. On simple low-cost processors, typically, bitwise operations are substantially faster than division, several times faster than multiplication, and sometimes significantly faster than addition. While modern processors usually perform addition and multiplication just as fast as bitwise operations due to their longer instruction pipelines and other architecture design choices, bitwise operations do commonly use less power because of the reduced use of resources

Table 1.8 Bitwise operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

Read more on bitwise operators from

https://en.wikipedia.org/wiki/Bitwise_operation and http://www.studytonight.com/c/operators-in-c.php

Assignment Operators

They are used to assign values to a variable. The form looks as follows;

expression(operand) assignment_operator expression;

For example;

```
a=3;
```

C language supports the following assignment operators;

Table 1.9 Assignment operators

Operator	Description	Example
=	assigns values from right side operands to left side operand	a=b
+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	multiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

Increment and decrement operators

These are unary operators and have higher priority than the other operators. Increment operator increments the current value of variable by 1. This operator is only applied on variables. Increment operator is denoted by ++. Decrement operator decrements the value in a variable by 1. It is denoted by --.

Both operators can be presented as;

1. Prefix (Pre-Increment)

Presenting an operator as prefix increments or decrements the value of variable before using it in the expression. This means that prefix format first increments or decrements the value then use it inside the expression.

For example;

```
++x or ++b;

--x or --b;
```

For example if the value in the memory a is 3 and is incremented using a postfix, value 3 will be incremented then used.

```
b=++a;
```

```
printf("%d %d", b, a)
```

output will be:

```
4       4
```

2. Postfix (Post-Increment Operator).

In postfix format, the value of variable is incremented or decremented after executing an expression. This means that the current value is first used in a expression and then incremented.

For example;

```
x++ or b++;

x-- or b--;
```

For example if the value in the memory a is 3 and is incremented using a postfix, value 3 will be used then incremented.

```
b=a++;

printf("%d %d", b, a)
```

output will be:

3 4

Special operator

Table 1.10 Other special operators

Operator	Description	Example
sizeof	Returns the size of an variable	sizeof(x) return size of the variable x
&	Returns the address of a variable	&x ; return address of the variable x
*	Pointer to a variable	*x ; will be pointer to a variable x

Read more on operators from the textbook by Deitel P. and Deitel H. (2016). C How to program. 8/e. Pearson Education --see chapter 2 section 2.5 on page 40. Also you can run the programs by McGrath M. (2014). C programming in easy steps. 4th ed. Published by Easy Steps Limited in UK. page 52-68

Input and output function in C (scanf() and printf())

Even with arithmetic you can not do very much other than write programs that are the equivalent of a pocket calculator. The real breakthrough comes when you can read values into variables as the program runs. Notice the important words here: "as the program runs". You can already store values in variables using assignment. That is: `a=3;`

stores 3 in the variable a each time you run the program, no matter what you do. Without some sort of input command every program would produce exactly the same result every time it was run. This would certainly make debugging easy! But in practice, of course, we need programs to do different jobs each time they are run. There are a number of different C input commands, the most useful of which is the scanf() command. To read a single integer value into the variable called a you would use:

```
scanf("%d",&a);
```

For the moment don't worry about what the &a means - concentrate on the difference between this and:

```
a=3;
```

When the program reaches the scanf statement it pauses to give the user time to type something on the keyboard and continues only when users press "enter" to signal that the user finished entering the value. Then the program continues with the new value stored in a. In this way, each time the program is run the user gets a chance to type in a different value to the variable and the program also gets the chance to produce a different result!

The final missing piece in the jigsaw is using the printf function, the one we have already used to print "Hello World", to print the value currently being stored in a variable. To display the value stored in the variable a you would use:

```
printf("The value stored in a is %d",a);
```

The %d, both in the case of scanf and printf, simply lets the compiler know that the value being read in, or printed out, is a decimal integer - that is, a few digits but no decimal point.

Note: the scanf function does not prompt for an input. You should get in the habit of always using a printf function, informing the user of the program what they should type, before a scanf function. (http://www.le.ac.uk/users/rjm1/cotter/page_28.htm).

Now, let us write a simple program to illustrate the use of scanf.

Read more on formated output from http://www.le.ac.uk/users/rjm1/cotter/page_31.htm

Conclusion

This activity taught us how to invent variables and apply appropriate data types. A variable is a location in the memory (space) where we can store data while a data type define the kind of data stored in a particular memory location. We reference the memory using identifiers which are unique names given to memory locations by the programmer. We input and output data using formatted specifiers. Operators manipulate individual data items and returns a result. C supports a number of these operators for example arithmetic operators, assignment operators, logical operators among others. Even with arithmetic you can not do very much other than write programs that are the equivalent of a pocket calculator. The real breakthrough comes when you can read values into variables as the program runs. Scanf() is used to read values to a program.

Assessment

1. Write a program that makes use of all the format specifiers (20marks)

2. Differentiate between: (Give example) (12marks)

   i. Signed and unsigned integers

   ii. Short and long integers

   iii. Float and long double

Reading assignment

3. Read on gets and puts function then;

Write a program that illustrates how gets and puts functions can be used

(5marks)

Lab 1.2: Practising use of variables,operators and input and output functions

Objectives of the lab are to:

- Use variables, operators
- Use input and output functions in programs

Details of the lab exercises/activities for compiler installation

Resources required

- Computer
- Unit 1 reference links (optional)

Time required:

2 hours

Description of lab exercise/activity

- You are required to start your compiler then from file menu select new file, then save it as lab13a.c then write a program that defines and declares variable r to 55, x to 67.9 and w to A then displays them on the screen.
  (5 marks)
- Open another new file from file menu, save it as lab13b.c then write a program that reads in two real values using the keyboard then computes and displays the product.
  (5 marks)
- Open another new file from file menu, save it as lab13c.c then write a program that reads two integers then computes and displays the modulo (5 marks)

- Open a new file then write a program that computes the area of a square whose side is 13m. save this program as lab13d.c (5 marks)
- Open a new file then write a program to compute the volume V of a sphere of radius r. (Recall that V=(4/3)πr3. save this program as lab13e.c (5 marks)
- Open a new file then write a program to convert miles to kilometers. (Recall that 1 mi=1.6093440km). save this work as lab13f.c (5 marks)
- Open a new file then write a program that read and displays two character and an integer then increments the integer by 3. save this program as lab13g.c (5 marks)

Results and submission requirements

You will be required to submit your lab activity as per your instructor's direction

Assessment criteria

Grade this work out of 35 marks as allocated

At the end it will be computed with other labs to 10%

References or key links

see activity 1.2 for the links on links to compilers.

Unit Summary

This unit looked C programming language concepts like variable, data types and operators. C language entail various elements (comments, header files, functions etc) as we saw in its structure. We defined a variable as a location in the memory (space) where we can store data while a data type define the kind of data stored in a particular memory location. We reference the memory using identifiers which are unique names given to memory locations by the programmer. We input and output data using formatted specifiers. Operators manipulate individual data items and returns a result. C supports a number of these operators for example arithmetic operators, assignment operators, logical operators among others. Even with arithmetic you can not do very much other than write programs that are the equivalent of a pocket calculator. The real breakthrough comes when you can read values into variables as the program runs. Scanf() is used to read values to a program.

Unit Assessment

1. Define the following:
(14marks)

   i. Structured programming

   ii. A variable

   iii. An identifier

   iv. An operator

   v. Top down analysis

   vi. Modular programming

   vii. Structured coding

2. Explain the functions of the C structure parts labeled 1 to 4
(4marks)

3. List any two advantages of structured programming
(2marks)

4. Differentiate between
(6marks)

   - i. structured and unstructured programming
   - ii. Primary and secondary data types
   - iv. Unary and binary operators

5. Name any three rules that must be followed when naming a variable (3marks)

6. Give any two primary data types.
(2marks)

7. Write a program that displays your first name initial, and age.
(4marks)

8. Write a program that reads and outputs your current age, and next year's age. Apply appropriate operator. (5marks)

9. Write a program that reads and computes the product of two integers (6marks)

10. IBM developed System/360 and software developers created an Operating system for System/360 known as OS/360. The OS/360 had many problems and the result was called software crisis.

    i. what do you understand by the term software crisis? (1mark)

    ii. Name two factors that were associated with software crisis. (2marks)

11. What is the output of the following code. (3marks)

```
int a, b, c, d=8;
a=++d;
b=a++;
c=b--;
printf("%d %d %d %d", a, b, ++c, d, --d, c--);
```

12. Convert the following for statement to a while statement: (3marks)

```
for(int i=2; i<=10; ++i){
    sum+=i;
    printf("%d\n",sum); }
```

Instructions

- You are required to attempt all the assessment
- Submit them to your instructor according to the agreed time (on time)
- You will be required to do thorough revision If your score falls below 60%.

Grading Scheme

All the questions have been assigned marks.

Compute all the assessments by totalling all of them then convert to 5%

Solutions

Q1.i. Structured programming is a method of writing computer program that uses top down analysis for problem solving, modularization for program structure and organization and structured code for individual modules.

ii. A variable is a memory space used to store data

iii. An identifier is a unique name given to a particular memory location

iv. An operator is a character or symbol that tells the compiler to perform a particular mathematical operation

v. Top down analysis is a problem solving technique which involve subdivision of larger problems into several smaller parts or tasks.

vi. Modular programming is breaking down of larger programs into smaller separate sections, thus organizing program instructions

vii. Structured coding is the organization of instructions within various control structures.

Q21. A comment for documenting the code

2. A preprocessor directive that provide necessary library functions to the program

3. It is the main function of the program, marking the point at which program execution begins

4. It is an instruction in the body of the function that tells the system to output the string "Hello world"

Q3i. programs are more easily and more quickly written

ii. programs have greater reliability

iii. programs require less time to debug and test

iv. programs are easier to maintain

Q4i. structured programming decomposes a program into smaller subprograms and minimises the use of goto statement whereas unstructured programming represents a program as a block of code and uses goto statement

ii. Primary data types are made up of primitive types types provided by the compiler whereas secondary data types are user defined types

iii. Unary operators operate on a single operand whereas a binary operator operates on two operands

Q5i. Should not be a C language keyword

ii. Should start with a letter or an underscore

iii. Should not start with a digit

Q6i. Integers (int)

ii. Float (float)

iii. Character (char)

Q10i. Software crisis was a situation where computer programs were unable to deliver the functionalities, full of bugs, unstructured, costly(in terms of production and maintenance) and exceeded the scheduled delivery time

ii. late program delivery

Program does not meet its functionalities

Exceeded cost (budget)

Q11Output:

10 8 9 8 8 9

Q12

```
int i=2, sum=0;
while(i<-10) {
        sum+=i;
++i;
}
        printf("%d\n",sum);
 }
```

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

- Deitel P. and Deitel H. (2016). C How to program with an introduction to C++. 8/e. Pearson Education.
- Chhabra, K. J. (2010). C programming concepts with problems and solutions. Tata McGraw Hill Education Private Limited. New Delhi pp1-86.
- McGrath, M. (2014). C programming in easy steps. 4th ed. Easy Steps Limited in UK. page 7-64
- https://scs.senecac.on.ca/~btp100/pages/content/sarra.html
- http://www.le.ac.uk/users/rjm1/cotter/page_62.htm
- http://beginnersbook.com/2014/01/2d-arrays-in-c-example/

Unit 2. Control structure and functions

Unit Introduction

The essence of writing computer program is to automate a process or an activity or provide a solution. Part of the solution is decision making and repetition of an action as required. This means a program is not limited to linear way of execution, at a particular time it selects an option from the available ones or loops for as long the condition is fulfilled. Let us go back a bit to unit 1 (activity 1.1) where we defined and mentioned three elements of structured programming. This is how we defined structured programming "a method of writing a computer program that uses (1) top-down analysis for problem solving, (2) modularization for program structure and organization, and (3) structured code for the individual modules" http://users.csc.calpoly.edu/~jdalbey/308/Resources/StructuredProgramming. Do you now remember! if not go back to that unit and using the link read the three elements. This unit will emphasize element 3 "structured code for the individual modules". Therefore we will go through decision making (conditional) structures, repetition structures and functions.

Unit Objectives

Upon completion of this unit you should be able to:

- write programs using conditional structures
- apply repetition structures in your program.
- create functions to modularize your program.

Key Terms

Conditional structures: are features of a programming language, which perform different computations or actions depending on whether a programmer-specified boolean condition evaluates to true or false. Apart from the case of branch predication, this is always achieved by selectively altering the control flow based on some condition (https://en.wikipedia.org/wiki/Conditional_(computer_programming))

Loops: is a sequence of instructions that is continually repeated until a certain condition is reached (http://whatis.techtarget.com/definition/loop).

Function: is a group of statements that together perform a task

Function prototype: It is a function declaration that tells the compiler about a function's name, return type, and parameters.

Learning Activities

Activity 2.1 - Conditional structures

Introduction

Consider this….

The university admission officer wants to admit 50 students to applied computer science programme. He found out that more than 300 students applied for the same. Not only that the officer wants to admit 50 students but it is required to follow a set course admission criteria (defining qualifications to the program). If a student has a B in mathematics, B in english and B in physics he or she automatically qualifies for the course. Another criteria is, if a student has a B in mathematics, B in english and either a B in physics or B Geography he or she qualifies for the course. Priority is given to the first criteria (B in mathematics, B in english and B in physics). If more than 50 students qualify, then rank according to the criteria and overall marks. The program will be expected to generate admission letters for the selected candidates such that as long as the letters are <=50, it will get next student in the ranked admission list.

How can we write such a program? And this gives us the reason to learn conditional structures in this activity.

Decision making in C

Decision making also known as conditional structure is about deciding the path to take based on a given condition. One path is executed if the condition is met and another one if not. This is achieved by applying these statements;

if statement

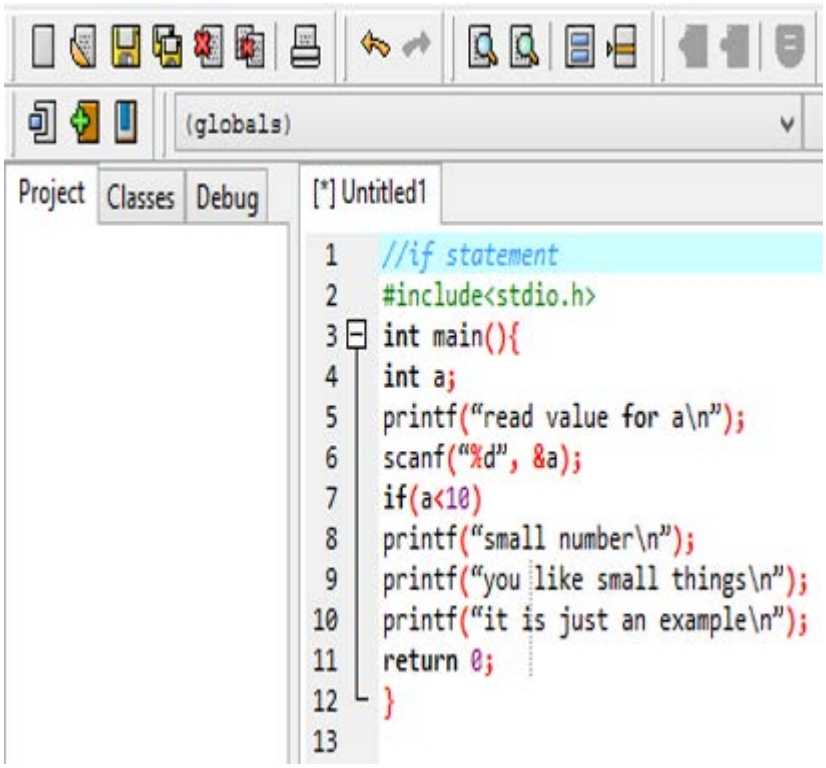
If statement evaluates an expression or condition and if it is true then executes its immediate statement. Otherwise it passes control to the statement below if statement.

Form:

```
if (expr) {statement/s;}
```

An if statement is not terminated simply because it is executing immediate statement. It is also blocked using the curly braces especially when it is meant to execute more than one statement or block of statements.

For example:



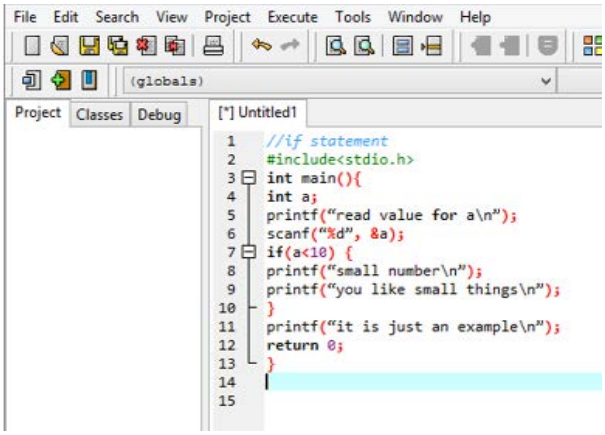
When you run this program, enter value 6 after the prompt, the program will output:

```
small number

you like small things

it is just an example
```

But when you run this program, enter value 11 after the prompt, the program will output:

```
you like small things

it is just an example
```

Now let us introduce curly braces, then see the difference.



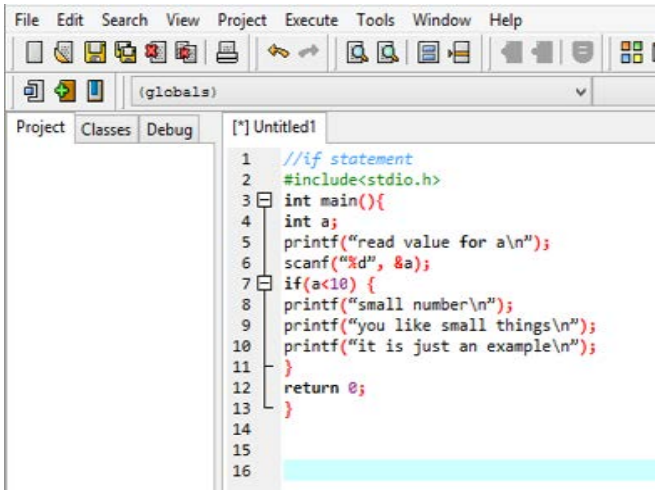
Now our program executes two instructions. This time when you run this program, enter value 6 after the prompt, the program will output:

```
small number

you like small things

it is just an example
```

Again when you run this program, enter value 11 after the prompt, the program will output:

```
it is just an example
```

This means that the value after the if statement is executed whether or not the condition is fulfilled (it is just an example). But when you block all the statements, see program;



Then read 6 after the prompt, it will output

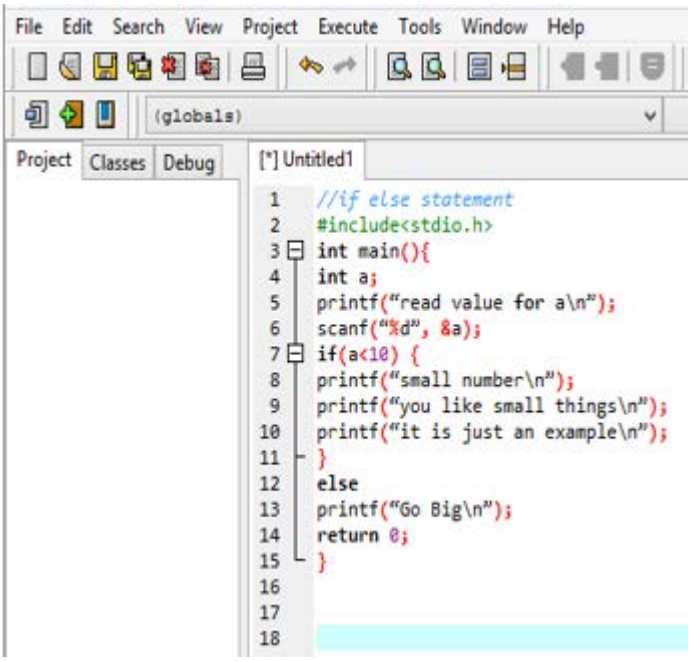
```
small number

you like small things

it is just an example
```

But when you read 11, it will output nothing on the screen.

If/else statement

If statement is not telling us what to do if condition or expression is not met. Therefore, we want to have a more interactive program by use of is/else statement. The statement says;

Let us rewrite our program;



Again when you run this program, enter value 6 after the prompt, the program will output:

```
small number

you like small things

it is just an example
```

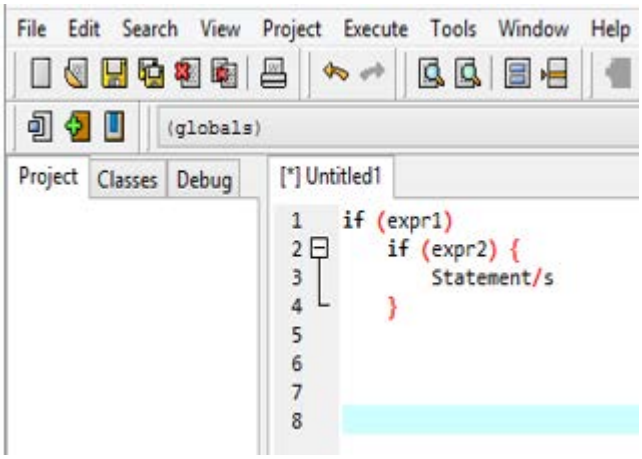
When you run this program, enter value 11 after the prompt, the program will output:

```
Go Big
```

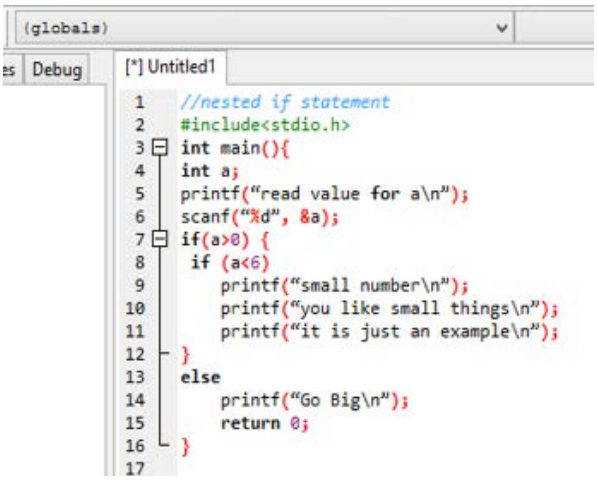
Now the program is telling us what to do if condition is not met.

Nested if statement

This statement has this form;



The outer if statement executes the inner if statement and if the condition for the outer statement is not met, then inner if statement will not be executed (it is skipped), meaning the statement/s will not be executed. We rewrite our program



Now when you run this program, enter value 6 after the prompt, the program will pass control to the inner if statement. Here the condition in the inner if statement is not met therefore control is passed to else statement and the output will be:

```
Go Big
```

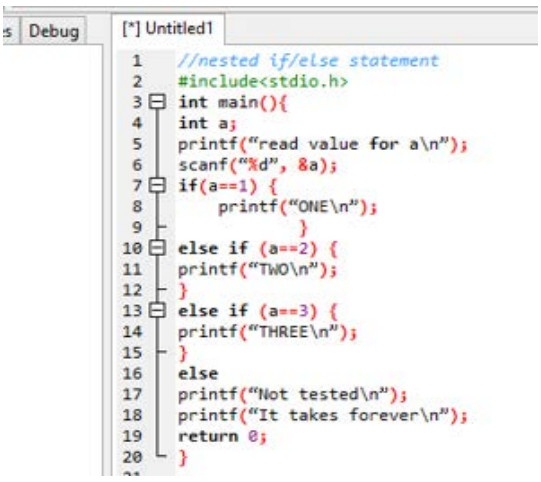
But when you run this program, enter value 5 after the prompt, the program will output:

```
small number

you like small things

it is just an example
```

Nested if/else statement

This statement tests an expression against a list of constants and returns results only when a match is found. It has this form;

See program:

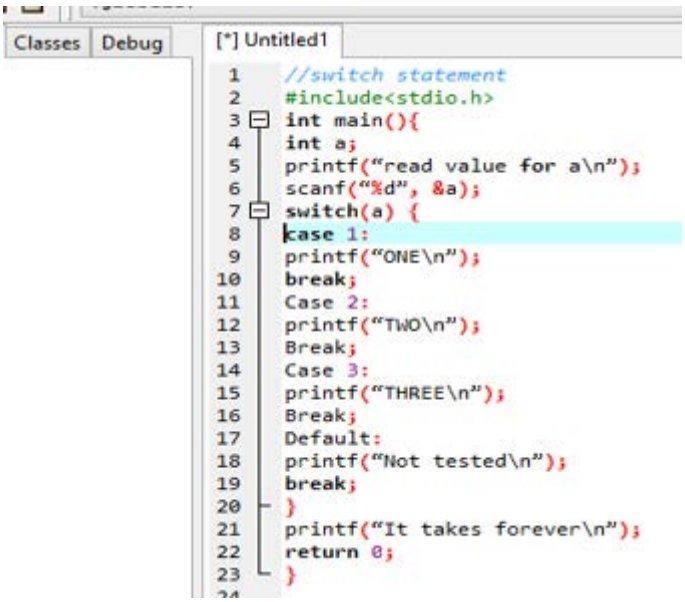


Switch statement

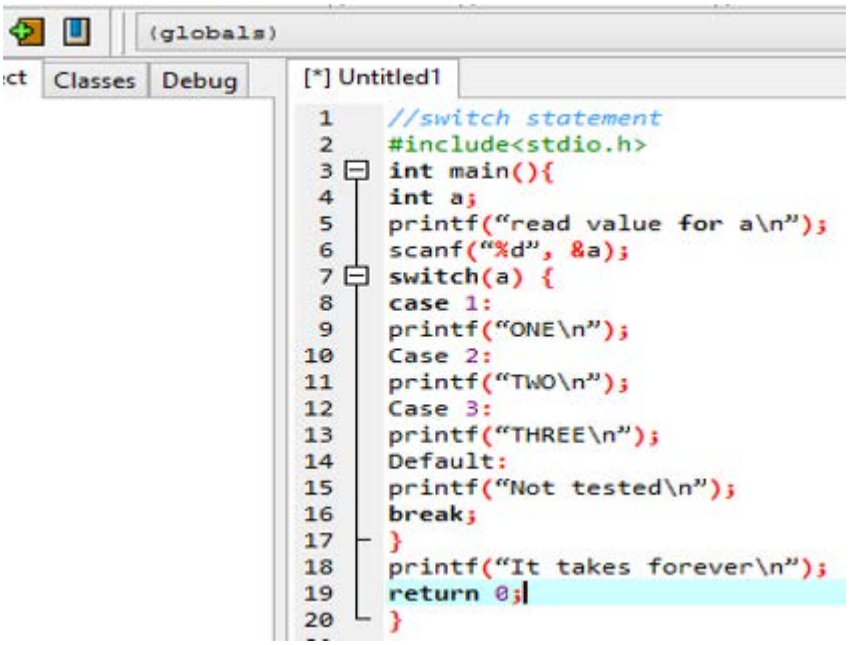
It is also known as multiple selection statement. Just like a nested if/else statement switch statement evaluates an expression against a list of constants then return results when a match is found.

It has this form:

We can rewrite the above nested if/else program as:



The break statement is used to terminate a case or ends a case and limits execution to a particular case. See program without break statement.



When value 1 is entered after the prompt, the program will output:

ONE

TWO

THREE

Not tested

It takes forever

Do not forget to break your cases while using a switch statement. Default case acts like else statement in nested else program. It gives an option when a match is not found.

Read chapter four of Deitel and Deitel textbook to learn more on switch statement. Page 123.

Refer to chapter three , pages 71 and 74 of Deitel and Deitel textbook on selection statements. Read and run the programs that are provided in the textbook.

Conclusion

Conditional statements aid in decision making or branching in a program. Expression is evaluated and if condition is met then the intended statement is executed. There are a number of conditionals statements supported by C which include: if, if/else, nested if, nested if/else statement and lastly we looked at multi-selection statement or switch statement. At this point we have a clue on how to implement our scenario (see introduction of activity 2.1)

Assessment

1. Name any two jump statements (2marks)
2. Rewrite the following code using the if-else statement. (4marks)
3. What is the output of the following program code; (5marks)

Activity 2.2 - Loops/Repetition

Introduction

We repeat the same scenario as activity 2.1….

The university admission officer wants to admit 50 students to applied computer science programme. He found out that more than 300 students applied for the same. Not only that the officer wants to admit 50 students but it is required to follow a set course admission criteria (defining qualifications to the program). If a student has a B in mathematics, B in english and B in physics, he or she automatically qualifies for the course. Another criteria is, if a student has a B in mathematics, B in english and either a B in physics or B Geography he or she qualifies for the course. Priority is given to the first criteria (B in mathematics, B in english and B in physics). If more than 50 students qualify, then rank according to the criteria and overall marks. The program will be expected to generate admission letters for the selected candidates such that for as long as the letters are <=50 it will get next student from the ranked admission list.

Emphasising on the last sentence, the program is required to repeat an action 50 times. Write the first letter, second……..the last. This will be possible only with the use of loops/repetition structures. And here we are, let us start learning it.

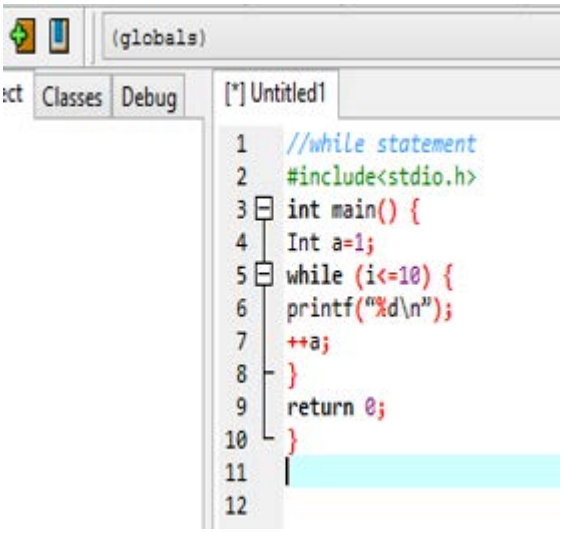
Loops are used to repeat a set of statement a number of times for as long as the condition is satisfied or met. In our scenario activity 2.2-introduction, the program will repeat letter writing for as long as the number of letter is less or equal to 50. There are three repetition statement; while, do/while and for statement. Let us discuss each statement;

While statement

While statement executes an instruction or block of instruction as long as the condition is true. The loop stops when the condition turns to false. It has the following form;

```
while(expr) {
                    Statement/s;
}
```

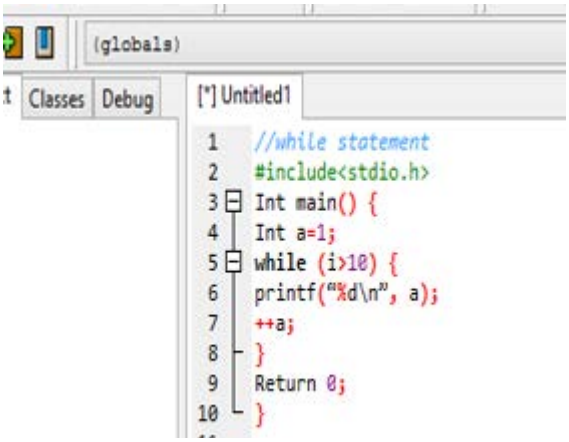
For example



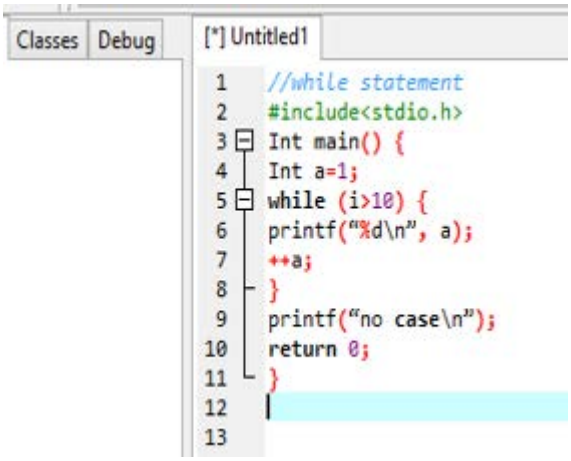
The condition is true because a =1 hence it is less than 10. The output will be:

1 2 3 4 5 6 7 8 9 10

But when we rewrite the same program as;



The condition turns to false because a =1 and the condition tests for values greater than 10. Hence there will be no output. Because of this, control is passed to the statement below while condition. For example;



This will output:

No case

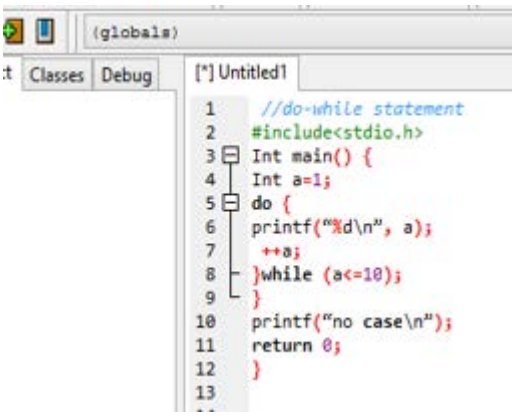
Which is the statement below while statement.

Do/while statement

Just like a while statement, do/while executes an instruction or block of instruction as long as the condition is true. The loop stops when the condition turns to false. It has the following form;

```
do{
                    Statement/s
}while(expr);
```

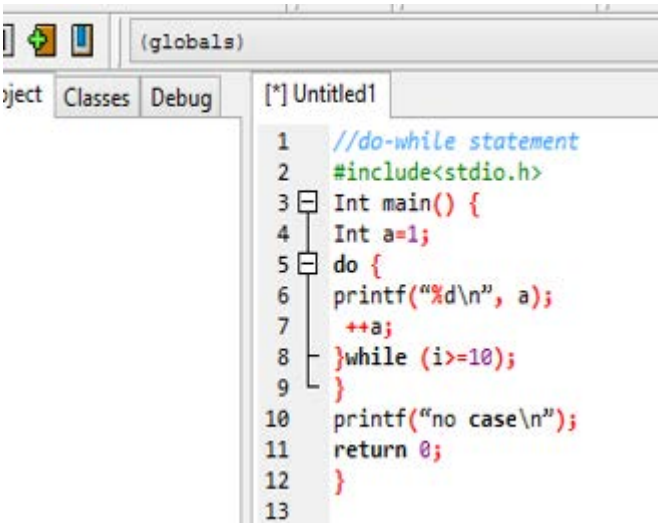
It says do statement/s then test the condition. Therefore, do/while returns a statement or value at least once whether or not the condition is met. For example;



This will output

1 2 3 4 5 6 7 8 9 10 No case

But when the condition is not true, see program;



It will output:

1

No case

For simple reason that do statement instructs the system to do something before evaluation. In this case it was instructed to display 1 before evaluating or testing the condition. Understood! If not, consult.

For statement

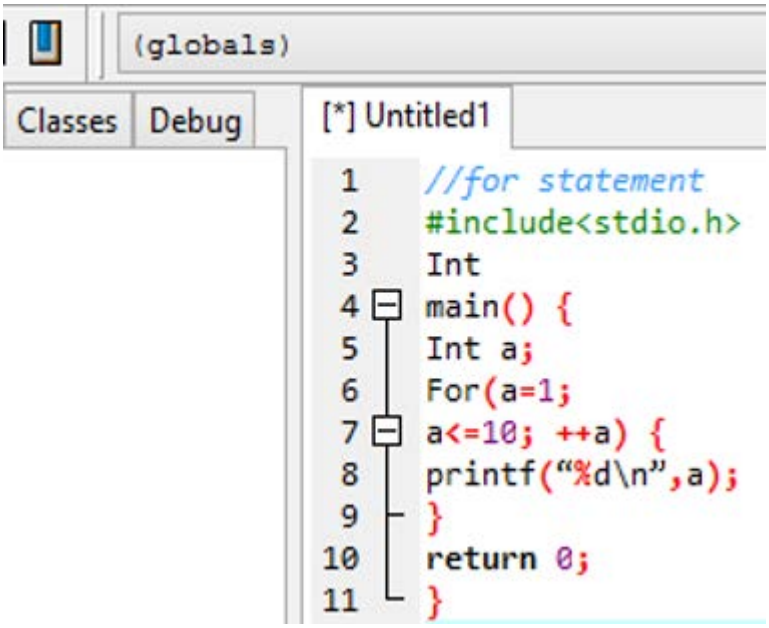
For statement works the same way as while statement. The difference is, for statement defines all expression inside the for delimiters (). See its format;

```
      for (exp1; exp2; exp3){

                        Statement/s

}
```

Or

```
for(initialization; condition; increment/decrement)

                          {

                        statement ;

                        }
```

Expression 1 denotes the minimum or maximum value (starting value), expression 2 is the condition to be evaluated and expression three is the increment/decrement operator. For example;



The condition is true because a =1 hence it is less than 10. The output will be:

1 2 3 4 5 6 7 8 9 10

Remember, expressions are terminated using a semicolon.

Nested for statement

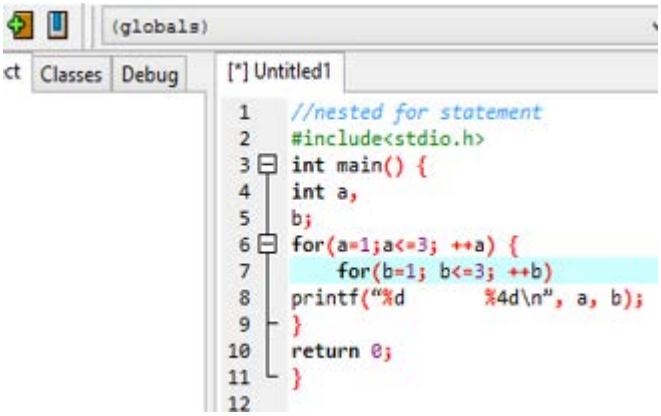
Nested for statement works as the nested if statement where the outer for statement executes the inner for statement. Its form :

```
for (exp1;
exp2; exp3){
            for (exp1; exp2; exp3){
            Statement/s
}
}
```

Or

```
for(initialization; condition; increment/decrement)
{
for(initialization; condition; increment/decrement)
   {
       statement;
    }
}
```

If the condition for the outer for statement is not met, then the inner for statement will not be executed. Otherwise if the condition for the outer for statement is met then the inner for statement is executed but if the inner for statement is false the immediate for statement instructions will not be executed. For example;



All the values in the inner for loop will cycle through an individual value in the outer for loop. The output therefore will be:

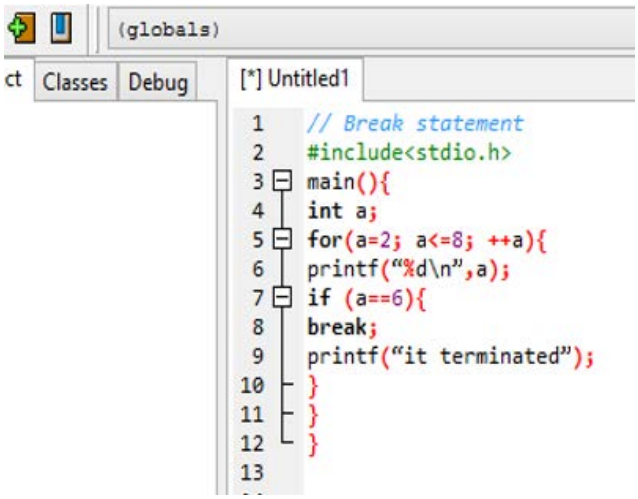
a	b
1	1
1	2
1	3
2	1
2	2
2	3
3	1
3	2
3	3

Jump statements

C language support two jump statements namely; break and continue. Jump statements are used to exit a loop.

Break statement

This statement totally exits a loop. When the program control encounters a break statement, it will terminate the loop then pass control to other statements outside the loop. For example,



The output will be

2

3

4

5

6

it terminated

Note that 7 and 8 were not printed because the control encountered a break statement before the end of the loop.

Continue statement

Continue statement is also a jump statement that jumps (or exits) the current iteration to the next iteration. Let us look at an example;

The output will be

2

3

4

5

i was not printed

7

8

Note that the value 6 was not printed. Program control passed control to the next iteration immediately it encountered continue statement.

Read and run the programs that are provided in chapter four of Deitel and Deitel textbook to learn more on repetition and jump statements. Page 114 - 130.

Conclusion

Loops repeat a statement or instructions for as long as the condition is met. There are three types of repetition statements namely; while, do/while and for statement. While statement test the condition before executing the intended instructions, do/while executes the intended instructions at least once before testing the condition and the for statement work the same way as while statement.

Assessment

1. For problem i to ii below, determine the number of times that the for loop is executed. (4marks-2each)

i).
```
for (int count=1; count<=14; count++)
                    {
                                statements;
                    }
```
ii).
```
for (int time=10; time<=5; time++)
                    {
                                statements;
                    }
```

2. Write a program to convert meters to Kilometers. (Recall that 1km=1000m). (5marks)

3. Rewrite the following for statement using a while structure (4marks)

```
for(int x=0; x<=10; ++x)
{
                    cout<<x <<",";  }
```

Activity 2.3 - Functions

Introduction

So far we have encountered one function known as main() function. Our programs are not limited to this function. We can have or invent other function depending on the nature and size of programs. Therefore in this unit we will learn how to create functions and use functions in our programs.

C is a structured programming language which means programs are decomposed or divided into functions. In unit 1 (activity 1.1) we mentioned three elements of structured programming and also defined it as "a method of writing a computer program that uses (1) top-down analysis for problem solving, (2) modularization for program structure and organization, and (3) structured code for the individual modules" http://users.csc.calpoly.edu/~jdalbey/308/Resources/StructuredProgramming. What we are looking at here is the top-down analysis for problem solving and modularization for program structure and organization which we have said entail decomposing a program into subprograms for ease of coding. Again read http://users.csc.calpoly.edu/~jdalbey/308/Resources/StructuredProgramming on these two elements then come back to our activity.

Importance of using functions

1. It enhances code reusability
2. It makes it easy to debug, and edit code
3. It makes the program more readable
4. It improves on the program development time

Now, what exactly is a C functions?

From http://www.le.ac.uk/users/rjm1/cotter/page_43.htm C functions are the equivalent of what in other languages would be called subroutines or procedures. If you are familiar with another language you also need to know that C only has functions, so don't spend time looking for the definition of subroutines or procedures - in C, function does everything!

A function is simply a chunk of Code (statements) that you have grouped together and given a name. The value of doing this is that you can use that "chunk" of code repeatedly simply by writing its name.

A program can have both user defined functions and library functions. A user-defined function (UDF) is a function provided by the user of a program or environment, in a context where the usual assumption is that functions are built into the program or environment (https://en.wikipedia.org/wiki/User-defined_function) while library functions are inbuilt functions in C programming. Function prototype and data definitions of these functions are written in their respective header file. For example: If you want to use printf() function, the header file <stdio.h> should be included (http://www.programiz.com/c-programming/library-function).

A function has this form:

```
return_type function_name(parameter list){

function body/definition;

}
```

Return_type define the type of value returned by the function

Function_name is a unique name that identifies a function

Parameter list: It contains the variable that will receive data send from another function (calling function). These are also known as formal parameters. Thus, these parameters act as placeholders for the actual parameters (these are arguments from the calling function).

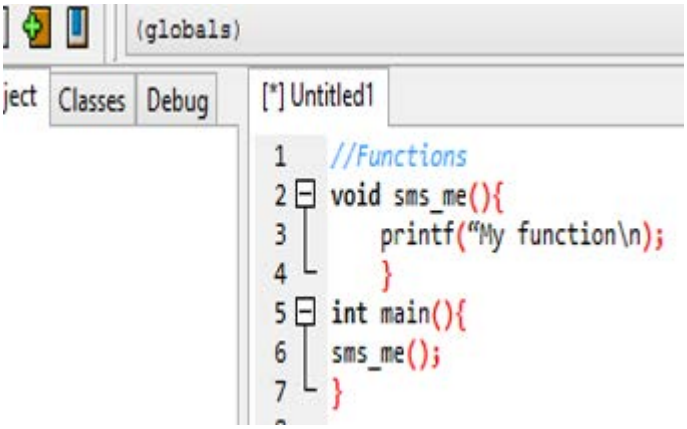
A parameter is a special kind of variable, used in a subroutine (function) to refer to one of the pieces of data provided as input to the subroutine. These pieces of data are called arguments. An ordered list of parameters is usually included in the definition of a subroutine, so that, each time the subroutine is called, its arguments for that call can be assigned to the corresponding parameters.

Just as in standard mathematical usage, the argument is thus the actual input passed to a function, procedure, or routine, whereas the parameter is the variable inside the implementation of the subroutine. For example, if one defines the add subroutine as def add(x, y): return x + y, then x, y are parameters, while if this is called as add(2, 3), then 2, 3 are the arguments. Note that variables from the calling context can be arguments: if the subroutine is called as a = 2; b = 3; add(a, b) then the variables a, b are the arguments, not only the values 2, 3 (https://en.wikipedia.org/wiki/Parameter_(computer_programming)).

This line return_type function_name(parameter list) is known as a function header. it defines the return type, name and list of parameters (types and number).

Function body/definition: just as the main() function the function body is enclosed using {}. Inside the curly braces are variables and instructions that define what the function is executing and thus the name function definition.

Let us write a simple program;



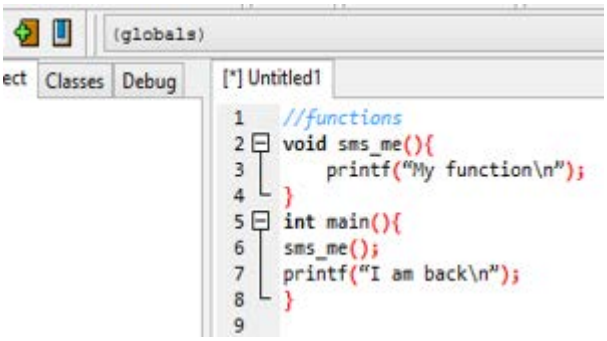
This program outputs:

```
My function.
```

Let us understand it-- First it starts from the main function, all programs start running from the main(). Control encounters the name of a function in main function's body. When it encounters this name, it then calls or invokes that function (called function), in this case sms_me. Just like when your friend calls you by name. What do you do? You react to the call by saying "here i am, what can i do for you". This applies to functions, now that a function has been called by name means program control will be passed to that function. In our example, program control is passed to sms_me() function.

At this time sms_me() has the program control thus executes what it was meant to do (responding to the call). Thus returning or displaying "My function". After execution, it returns back control to the calling environment (in this case main function).

Calling environment will continue executing other statements. See this other program, it returns control to printf function.

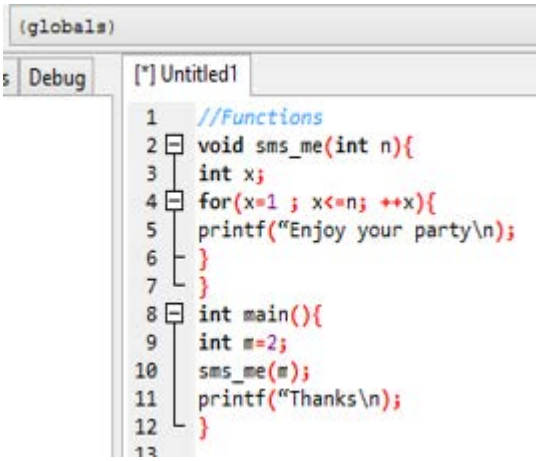


Thus output is:

```
My function

I am back
```

Understood! Good. Now, notice how the return type is void (meaning it is not returning any value) and our parameter list is empty (meaning it is void--it does not receive any parameter or argument).

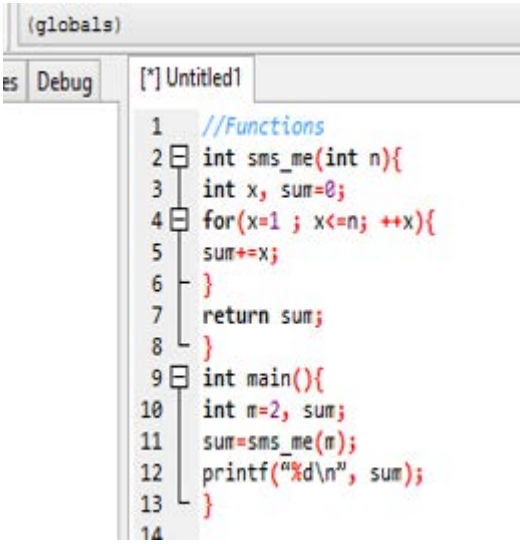
Let us improve it once again.



Our function (sms_me) contains a parameter n that accepts an integer. Now let us run the program from the main function. Integer m is passed as an argument to sms_me function . A copy of the value in m is copied to n (n=m;) and now n has 2. The output will be;

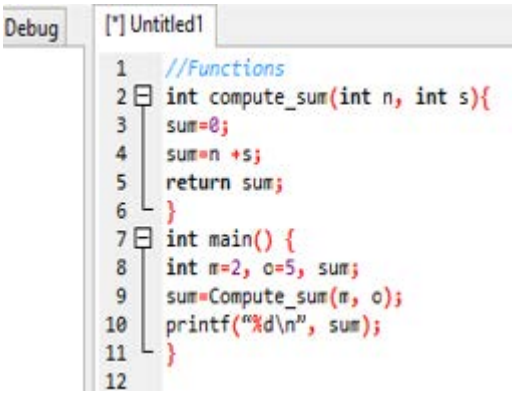
```
Enjoy your party

Enjoy your party

Thanks
```

We improve it even further.



This program has a return type and one parameter. sms_me function will receive the argument from the main function then compute and return sum. Return statement inside the function tells the compiler to return results to the calling environment. The function is returning an integer hence its return type is int.

Let us write one more program that receive two parameters;



Read more on functions (http://www.le.ac.uk/users/rjm1/cotter/page_42.htm)

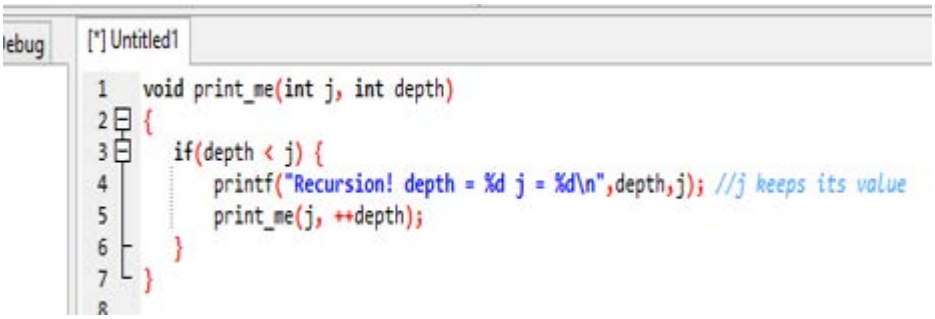
- Read on function prototyping

Recursion

Here's a simple function that does an infinite loop. It prints a line and calls itself, which again prints a line and calls itself again, and this continues until the stack overflows and the program crashes. A function calling itself is called recursion, and normally you will have a condition that would stop the recursion after a small, finite number of steps.

```
// don't run this!

void infinite_recursion()

{

    printf("Infinite loop!\n");

    infinite_recursion();

}
```

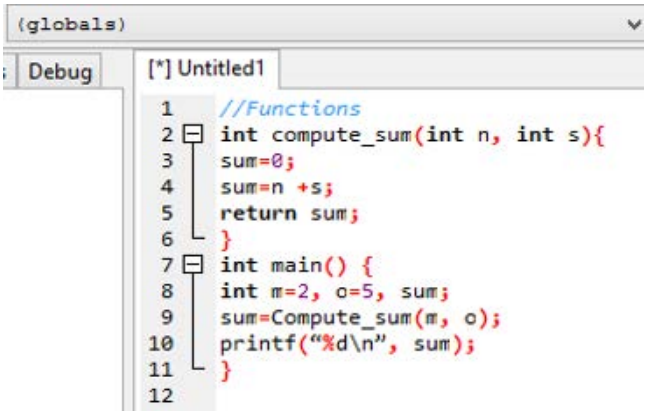
A simple check can be done like this-see program code below. Note that ++depth is used so the increment will take place before the value is passed into the function. Alternatively you can increment on a separate line before the recursion call. If you say print_me(3,0); the function will print the line Recursion 3 times.



Recursion is most often used for jobs such as directory tree scans, seeking for the end of a linked list, parsing a tree structure in a database and factorising numbers (and finding primes) among other things (https://en.wikibooks.org/wiki/C_Programming/Procedures_and_functions).

Passing arguments by value and by reference (it is recommended that you read passing arguments after learning pointers)

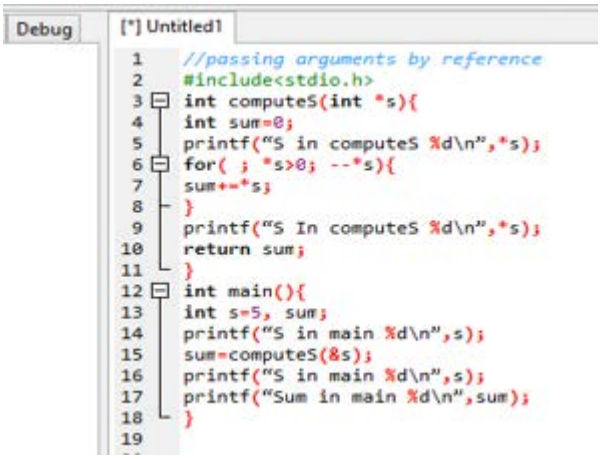
Arguments can be passed by value or by reference. Let us start with passing arguments by value. To understand this we need to start with a program.



This program passes a copy of the value stored in s in the main function to the parameter in computeS (s=s). (Do not worry that the variable names are the same, we shall know why later). The output will be:

S in main	5
S in computeS	5
S in computeS	0
S in main	5
Sum in main	15

Now lets us rewrite the same program then pass arguments by reference



This program passes the address of variable s in the main function to the parameter in computeS which is a pointer to an integer *s (int *s=&s). (Do not worry that the variable names are the same, we shall know why later). The operator & is known as a reference operator (address) while '*' operator is known as redirectional or dereference operator.

The output will be:

S in main 5

S in computeS 5

S in computeS 0

S in main 0

Sum in main 15

This tells us that when an argument is passed by value, a copy of the passed value is copied to the formal (receiving) parameters and the value gets modified in the called function but not in the calling environment. On the other hand, when an argument is passed by reference, the address of the passed argument is assigned to the formal parameter. Hence any modifications to the value in called function will automatically affect the value in the calling environment. Compare the results (passing by value and by reference)

Passing an arguments by reference (address) is very efficient. It avoids copying all of the member values, saving both time and space especially in cases where a member is an array with a large number of elements. Passing an object by address only copies the address, which typically occupies 4 bytes (https://scs.senecac.on.ca/~btp100/pages/content/struc.html)

Scope of variable

Variables can be declared as either local variables or global variables. Global variable is declared at the start of the program, their global scope means they can be used in any procedure or subroutine in the program. It is seldom advisable to use Global variables as they are liable to cause bugs, waste memory and can be hard to follow when tracing code. If you declare a global variable it will continue to use memory whilst a program is running even if you no longer need/use it. Local variable is declared within subroutines or programming blocks, their local scope means they can only be used within the subroutine or program block they were declared in. Local variables are initiated within a limited scope, this means they are declared when a function or subroutine is called, and once the function ends, the memory taken up by the variable is released. This contrasts with global variables which do not release memory (https://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/Global_and_Local_Variables)

For example:

Debug [*] Untitled1

```
1  // simple program on global and local variables
2  #include<stdio.h>
3  double INTEREST =0.12;              //global variable
4  main(){
5  float principal = 6000, totalP;     //local variables
6  totalP=((principal * INTEREST) + principal);
7  printf("%f\n", totalP);
8  }
```

Read and run the programs that are provided in chapter five of Deitel and Deitel textbook to learn more on functions and recursive functions. Page 158 - 195.

Conclusion

A function is a group of instructions that execute a particular task. It performs a specific activity. This forms the essence for structured programming where programs are decomposed into small programs known as functions. Functions has various benefits include, code reuse, ease of program documentation, debugging, readability and understandability among others. C language supports two types of functions, library function which are built in functions and user defined function; functions that are invented by the developer based on the nature of the program. A function has an header that defines function's return type, name and the list of parameters including the number of parameters and their types. A parameter can be formal or actual. Formal parameters are the parameters receiving arguments sent from a calling function while actual parameters entail arguments being sent to a function.Hence formal parameters act as placeholders for the actual parameters. A function may need to be defined or declared. Declaration of a function is known as a function prototype. When arguments are passed by value, a copy of the value in the actual parameter is passed to the called function and any modifications to that value in the called function will not affect the value in the calling environment. Lastly we said that a variable can be either local or global.

Assessment

1. What is a function?
   (2marks)
2. Differentiate between local and global variables
   (2marks)
3. Give three benefits of using a function in a program
   (3marks)
4. Write a program with a function that returns the square of value to the main function
   (5marks)
5. Write a program that reads in first name, last name as a string and an initial for the middle name as a character. Also, reads in current age then increments age by 1 to give age in a year's time. the program outputs names in one line, current age on new line, then next year's age on another line
   (8marks)
6. What is a function prototype?
   (1mark)
7. What is the significance of a function prototype in a program
   (2marks)

Lab 2.1: Conditional structures/loops/functions

Objectives of the lab are to:

- Learn how to use conditional structures
- Learn how to use loops
- learn how to use functions

Resources required

- Computer
- Unit 2 reference links (optional)

Time required:

2 hours

Description of lab exercise/activity

- You are required to start your compiler then from file menu select new file, then save it as lab21a.c then write a program displays "admitted to ACSP" only if X has above 80 marks in mathematics and over 75 in physics (5marks)
- Open another new file from file menu, save it as lab21b.c then write a program that reads then displays "no lights" if the value is either 0 or 7, displays "full lights" if the value is 1 or 6 otherwise displays "battery not charged" (5marks)
- Open another new file from file menu, save it as lab21c.c then write a program that reads character and if it is 'A' it outputs "World class", 'B' it displays "class" and 'C' it shows "classless". (5marks)
- Open a new file then write programs that outputs the following patterns: (5marks @pattern)

Save pattern as patterna.c

6

6 6

6 6 6

6 6 6 6

6 6 6 6 6

6 6 6 6 6 6

Save pattern as patternb.c

5 5 5 5 5 5 5 5

5 5 5 5 5 5

5 5 5 5

5 5 5

5 5

5

5 5

5 5 5

5 5 5 5

5 5 5 5 5

5 5 5 5 5 5

save pattern as patternc.c

- Open a new file then write a program that reads an integer then passes it to a function which computes the sum and average of the value series from 1 to n(read) then return sum to the calling function. save this program as lab21d.c (5marks)
- Write a program that passes an integer to a function which returns sum of the values less or equal to the keyed in value

Results and submission requirements

You will be required to submit your lab activity as per your instructor's direction

Assessment criteria

Apply the allocated marks for grading

References or key links

see unit 2 for the links on links to reference notes (optional).

Unit Summary

Conditional statements aid in decision making or branching in a program. It entails evaluation of an expression and if condition is met then the intended statement is executed. The conditionals statements include: if, if/else, nested if, nested if/else statement and lastly we looked multi-selection statement or switch statement. Loops repeat a statement or instructions for as long as the condition is met. There are three types of repetition statements namely; while, do/while and for statement. Late in this unit we looked at functions where we defined functions as a group of instructions that execute a particular task. Functions has various benefits include, code reuse, ease of program documentation, debugging, readability and understandability among others. A function may need to be defined or declared. Declaration of a function is known as a function prototype. When arguments are passed by value, a copy of the value in the actual parameter is passed to the called function and any modifications to that value in the called function will not affect the value in the calling environment. Here also, we categorized variables as either local or global.

Unit Assessment

Check your understanding!

1. Write a program to compute the area A of the surface of a sphere of radius r. (Recall: A=4πr2)
(8marks)

2. Write a program that increments time by 2 if the dist is less than 50.0 and dist is greater than 10.0 otherwise, increments time by 2.5,.
(5marks)

3. Write a program that prints values 20 to 10 and executes a break statement on the seventh iteration.
(7marks)

4. Write a function that returns a remainder of two values multiplied by 10 to the main function.
(6marks)

5. Write a program that converts the following if/else statement to a switch statement

```
if(code==10||code==11)
      printf("Too hot- turn equipment off\n");
else
if(code==12||code==13)
      printf("Cation- recheck in five minutes\n");
```

```
else

if(code==14)

                printf("Turn on circulatingfan\n");

else

          printf("Normal mode of operation\n");
```

6. Name and give functions of the following parts 1 and 2 of a function (4marks)

```
return_type functuion_name ( … )          1

                          {

                                  //statement block          2

                    }
```

7. Recursivity is the property that functions have to be called by themselves. Write a recursive function that returns factorial of a number to the main function. (5marks)

Instructions

Answer all the questions

If you score below 60% then you will be required to reread the entire unit activities.

Grading Scheme

All the unit activities will be computed to 5%

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

- Deitel, P. and Deitel, H. (2016). C How to program with an introduction to C++. 8/e. Pearson Education.
- (Pages and chapters to be referenced by this book are indicated in the content)
- Chhabra, K. J. (2010). C programming concepts with problems and solutions. Published by Tata McGraw Hill Education Private Limited.
- New Delhi. pp93-171
- https://scs.senecac.on.ca/~btp100/pages/content/sarra.html
- http://www.le.ac.uk/users/rjm1/cotter/page_62.htm
- http://beginnersbook.com/2014/01/2d-arrays-in-c-example/

Unit 3. Data Structures

Unit Introduction

C language support a number of data structures which include arrays, strings, pointers, structures and unions. These structures provide the cheapest way of organizing and handling data. This unit's activities will introduce them one after another.

Unit Objectives

Upon completion of this unit you should be able to:

- write programs using arrays and strings.
- apply pointers in C programs.
- create structures and unions in C programs.

Key Terms

Array: is a data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type (http://www.tutorialspoint.com/cprogramming/c_arrays.ht).

Strings: is an array of characters

Pointers: is a variable whose value is the address of another variable

Structures: is a user defined data type available in C that allows to combine data items of different kinds.

Unions: is a special data type available in C that allows to store different data types in the same memory location.

code reuse: is the use of existing software, or software knowledge, to build new software, following the reusability principles

Learning Activities

Activity 3.1 - Arrays and strings

Introduction

So far we know how to declare and use variables of different types. This activity will come in handy as it will help you learn how to organize data and handle strings in C programming. Thus the activity will take us through arrays and strings. We shall start with arrays then proceed to strings.

Arrays

If you are told to compute the sum of the following values;

10, 20, 30, 40, and 50.

The best thing to do is to declare five variables. For example;

```
int a=10;

int b=20;

int c=30;

int d=40;

int e=50;
```

followed by the computation instructions.

What is an array?

An array is a series of data items that are of the same type stored in a contiguous manner and are indexable. Its form is as follows;

```
type array_name[array size/index];
```

Type define the kind of data types stored by the array

array_name is an identifier that uniquely identifies the array

And [ ] depending on the context, these delimiters are used to define size of the array or index

For example:

```
int abc [5];
```

defines array named abc that stores five integer values.

```
float abc[5];
```

defines an array named abc that stores 5 real values

An array is indexed from 0 to n-1. For example we have defined arrays abc to five integers, means it starts from; abc[0].....abc[4]

Array initialization

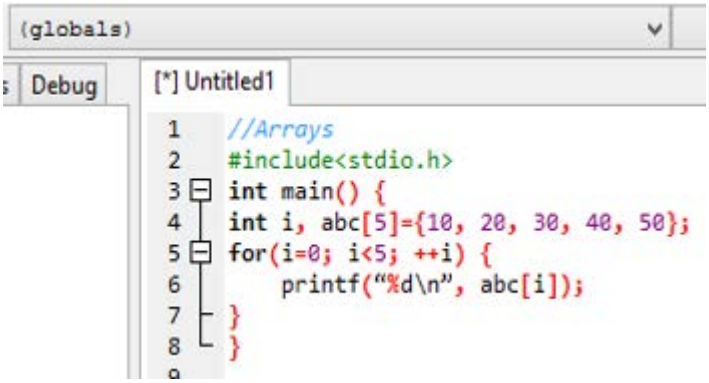
We can assign values to an array as follows;

```
int abc[0]=10;

int abc[1]=20;

int abc[2]=30;

int abc[3]=40;

int abc[4]=50;
```

This still is a long process. To minimize this, we can initialize the array with values using curly braces as shown below;

int abc[5]={10, 20, 30, 40, 50};

Let us write a program to illustrate arrays;



To manipulate an array, we require a for statement.

Read more on arrays (one dimensional arrays) from http://www.le.ac.uk/users/rjm1/cotter/page_54.htm C also supports other types of arrays known as multidimensional arrays. We will look at simplest multidimensional array (in this case a two dimensional array).

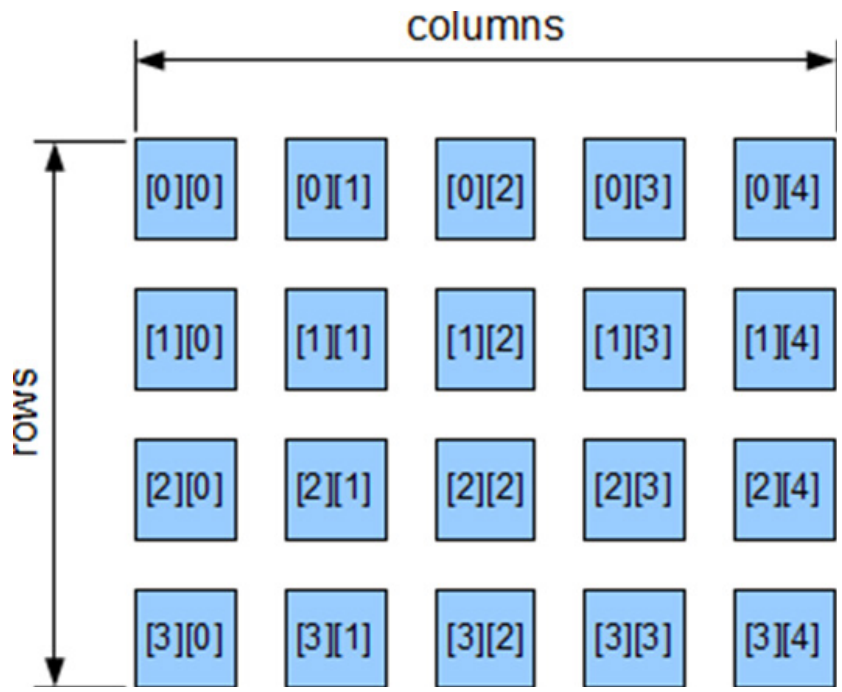
Two dimensional array

The simplest data structure for organizing tabular data is a two-dimensional array. The C language supports multi-dimensional arrays. C compilers treat a two-dimensional array as an array of arrays.

This activity introduces two-dimensional arrays, describes their syntax and organization in memory and demonstrates how to code an array of strings.

Two dimensional syntax

A table of values is a useful analogy for understanding a two-dimensional array. A simple way of identifying a table entry is by its position in the table; that is, by its row and its column. Consider the row and column indices in the figure below.



To identify an element of a two-dimensional array we use two pairs of brackets. The index in the left pair identifies the row, while the index in the right pair identifies the column:

array[row][column]

Definition

The definition of a two-dimensional array takes the form

```
type identifier[r][c]=init;
```

where r is the number of rows in the array and c is the number of columns. r and c are integer constants or constant integer expressions. The total number of elements in the array is r * c. init is a braces-enclosed, comma-separated list of initial values. The assignment operator together with init are optional. If we add an initialization list, we may omit the value of r. If r * c exceeds the number of initial values, the compiler initializes the remaining elements to 0. If we omit the initialization list, we must specify r. We must always specify c.

For example,

int a[4][5] = {11, 12, 13, 14, 15, 21, 22, 23, 24, 25, 31, 32, 33, 34, 35, 41, 42, 43, 44,

45};

To improve clarity, we may enclose each subset of initial values for each row in additional braces:

```
int a[4][5] = {{11, 12, 13, 14, 15},
               {21, 22, 23, 24, 25},
               {31, 32, 33, 34, 35},
               {41, 42, 43, 44, 45}};
```

Order

The C language stores the elements of a two-dimensional array in row-major order: the first row, column-element by column-element, then the second row, column-element by column-element, then the third row, etc..

For example, the elements of the array

int a[4][5];

are stored as follows

```
a[0][0] a[0][1] a[0][2] a[0][3] a[0][4]
 a[1][0] a[1][1] a[1][2] a[1][3] a[1][4]
 a[2][0] a[2][1] a[2][2] a[2][3] a[2][4]
 a[3][0] a[3][1] a[3][2] a[3][3] a[3][4]
```

Not all programming languages store two-dimensional arrays in row-major order.

Read and run the programs in chapter six of Deitel and Deitel's textbook referenced for this unit from page 215 - 253.

Conclusion

Arrays are data structures that store data items that are of the same type. A for structure is used to manipulate the contents of an array. C offers different types of arrays also known as multidimensional arrays and this activity looked at one and two dimensional arrays.

Assessment

1. Write a program that prints all elements of an array. The array is initialized as:

```
int frac[]={20, 30, 40, 50,60};
```
(4marks)

2. Basing on the above array (1.),

   i. Name the third element from the beginning of array (2marks)

   ii. Assign the 45 to array element four (2marks)

3. Answer the following questions regarding another array called sqre

   i. Declare the array sqre to be an integer array with 3 rows and 3 columns. (2marks)

   ii. How many elements does the array contain (2marks)

iii. Write a program that initializes each element of the sqre array to the product of its subscripts. Assume the integer variable x and y are declared as control variables. Print array elements and show the output. (6marks)

Activity 3.2 - Strings

Introduction

A string is generally understood as a data type and is often implemented as an array of bytes of bytes (or words that stores a sequence of elements, typically characters, using some character encoding. A string may also denote more general arrays or other sequence (or list) data types and structures. Let us now look at strings with its operations.

Strings

C has no string handling facilities built in; consequently, strings are defined as arrays of characters. C allows a character array to be represented by a character string rather than a list of characters, with the null terminating character automatically added to the end. For example, to store the string "Merkkijono", we would write

char string[] = "Merkkijono";

String "Merkkijono" stored in memory

or

char string[] = {'M', 'e', 'r', 'k', 'k', 'i', 'j', 'o', 'n', 'o', '\0'};

In the first example, the string will have a null character automatically appended to the end by the compiler; by convention, library functions expect strings to be terminated by a null character. The latter declaration indicates individual elements, and as such the null terminator needs to be added manually.

Strings do not always have to be linked to an explicit variable. As you have seen already, a string of characters can be created directly as an unnamed string that is used directly (as with the printf functions.)

To create an extra long string, you will have to split the string into multiple sections, by closing the first section with a quote, and recommencing the string on the next line (also starting and ending in a quote):

```
char string[] = "This is a very, very long "
                "string that requires two lines.";
```

Strings may also span multiple lines by putting the backslash character at the end of the line, this method is deprecated.

There is a useful library of string handling routines which you can use by including another

header file (https://en.wikibooks.org/wiki/C_Programming/Arrays). A number of functions can be performed on strings. These functions include copying, concatenating, computing the length. Read more on strings and string functions from the text book by Chhabra, K. J. (2010) C programming concepts with problems and solutions. Published by Tata McGraw Hill Education Private Limited. New Delhi (page 181-198) or http://www.le.ac.uk/users/rjm1/cotter/page_62.htm

Also, read and run the programs on C characters and strings in chapter eight of Deitel and Deitel's textbook referenced for this unit from page 334 - 351.

Conclusion

Strings are also defined as array of characters. There are many functions that can be performed with strings like copying, concatenating, computing the length etc.

Assessment

1. Define the following;

   i. a string
   (2marks)

   ii. String literal
   (2marks)

   iii. '\0' at the end of a string
   (2marks)

2. Writeaprogramthatreadsincharactersandconvertslowercaseletterstouppercase
   (6marks)

3. C program can perform a number of operations with strings. Using the following strings,

string str1 Egerton

string str2 University

string str3;

4. Write a program that to demonstrate how the following can be achieved. (12marks)

   i. Copying the first string to str3

   ii. Concatenating str3 and str2

   iii. Length of str3

iv. Comparing str1 with str2

Activity 3.3 -Pointers

Introduction

We have looked at static data structures like array and have known how to apply them. C provides a better way of handling the memory or using the memory in a efficient way. Thus this unit will take us through a structure in C that is able to create dynamic memory allocations or can be used to allocate and deallocate the memory as required. It can also be used in conjunction with other data structures in C like arrays and structures for enhanced efficiency.

Pointers

Pointer are a fundamental part of C. If you cannot use pointers properly then you have basically lost all the power and flexibility that C allows. The secret to C is in its use of pointers. The pointers in C are used explicitly with arrays, structures, and functions.

C is powerful because of pointers and this are some of the reasons why pointer are mostly used in C (https://www.cs.cf.ac.uk/Dave/C/node10.html):

- It is the only way to express some computations.
- It produces compact and efficient code.
- It provides a very powerful tool.

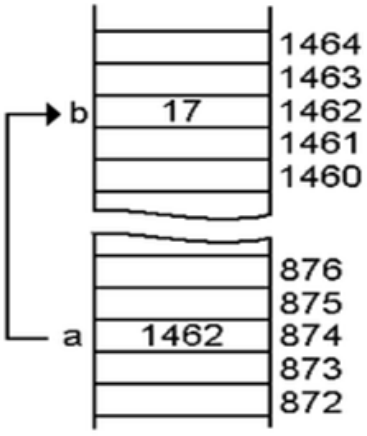
A pointer is a variable which contains the address in memory of another variable. We can have a pointer to any variable type. The unary operator & gives the ``address of a variable'' and the indirection or dereference operator * gives the ``contents of an object pointed to by a pointer'' (https://www.cs.cf.ac.uk/Dave/C/node10.html):

A pointer has this form:

type *pointer_name;

for example

int b, *a; declares a variable b to an integer and a pointer a to an integer see diagram



Pointer a pointing variable b. Note that b stores number, whereas a stores address of b in memory (1462). This is achieved by assigning the address of b to the pointer as;

```
int *a=&b;
```

which can also be assigned like this;

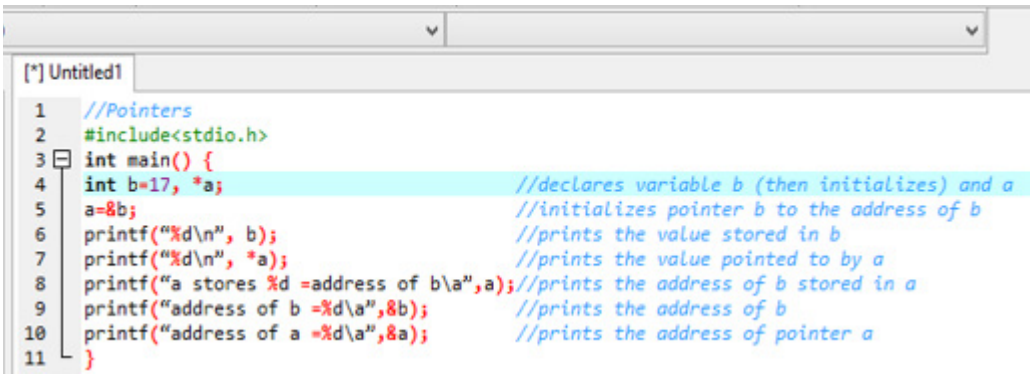
```
int *a;

        a=&b;
```

Refer to C pointers in chapter seven of Deitel and Deitel's textbook referenced for this unit from page 275 - 308.

You can aslo read http://www.le.ac.uk/users/rjm1/cotter/page_59.htm for more information on pointer

```
printf("%d",*a);     //this will print the value of b.

printf("%d",*&b);  //this will also print the value of b.

printf("%u",&b);  //this will print the address of b.

printf("%u",a);  //this will also print the address of b.

printf("%u",&a);  //this will print the address of a.
```

Let us illustrate the use of pointers using a simple program



output will be:

17

17

a stores 1462=address of b

address of b =1462

address of a =874

In C there is a very close connection between pointers and arrays. In fact they are more or less one and the same thing! When you declare an array as:

int a[10];

you are in fact declaring a pointer a to the first element in the array. That is, a is exactly the same as &a[0]. The only difference between a and a pointer variable is that the array name is a constant pointer - you cannot change the location it points at. When you write an expression such as a[i] this is converted into a pointer expression that gives the value of the appropriate element. To be more precise, a[i] is exactly equivalent to *(a+i) i.e. the value pointed at by a + i . In the same way *(a+ 1) is the same as a[1] and so on.

Being able to add one to a pointer to get the next element of an array is a nice idea, but it does raise the question of what it means to add 'one' to a pointer. For example, in most implementations an int takes two memory locations and a float takes four. So if you declare an int array and add one to a pointer to it, then in fact the pointer will move on by two memory locations. However, if you declare a float array and add one to a pointer to it then the pointer has to move on by four memory locations. In other words, adding one to a pointer moves it on by an amount of storage depending on the type it is a pointer to.

This is, of course, precisely why you have to declare the type that the pointer is to point at! Only by knowing that a is a pointer to int and b is a pointer to float can the compiler figure out that a + 1

means move the pointer on by two memory locations i.e. add 2, and b + 1

means move the pointer on by four memory locations i.e. add 4. In practice you don't have to worry about how much storage a pointer's base type takes up. All you do need to remember is that pointer arithmetic works in units of the data type that the pointer points at. Notice that you can even use ++ and -- with a pointer, but not with an array name because this is a constant pointer and cannot be changed. So to summarise: An array's name is a constant pointer to the first element in the array that is a==&a[0] and *a==a[0].

Array indexing is equivalent to pointer arithmetic - that is a+i=&a[i] and *(a+i)==a[i].

It is up to you whether you want to think about an array as an array or an area of storage associated with a constant pointer. The view of it as an array is the more sophisticated and the further away from the underlying way that the machine works. The view as a pointer and pointer arithmetic is more primitive and closer to the hardware. In most cases the distinction is irrelevant and purely a matter of taste.

Read more on pointers and arrays from http://www.le.ac.uk/users/rjm1/cotter/page_59.htm and the text book by Chhabra, K. J. (2010) C programming concepts with problems and solutions. Published by Tata McGraw Hill Education Private Limited. New Delhi (page 206-238)

Conclusion

This activity looked at pointers in C. Pointers are variables that point to another variable and stores the address of the "pointed to" variable. Pointer is a tool that makes C powerful. It is used when creating dynamic memory allocations, passing arguments to a function, among other uses.

Assessment

1. What is a pointer?
(2marks)

2. Give memory snapshots after each of these sets of statements are executed.

   i.
   ```
   int a(1), b(2), c(5), *ptr=&c;

   b=*ptr;

   *ptr=a;
   ```
   (2marks)

   ii.
   ```
   int a(1), b(2), c(5), *ptr;

   ptr =&c;

   c=b;

   a=*ptr;
   ```
   (2marks)

3. Write a program that prints the contents and addresses of two integers a and b, initialized to 1 and 2 respectively. (6marks)

4. For each of the following, write a single statement that performs the indicated task. Assume that integer variables x and y have been declared, and that x has been initialized to 3

5.

   a. Declare the variable ptr to be a pointer to an object of type integer (2marks)

   c. Assign the address of variable x to pointer variable ptr (2marks)

   d. Print the value of the object pointed by ptr (2marks)

   e. Assign the value of the object pointed to by ptr to variable y (2marks)

   f. Print the value of y (2marks)

   g. Print the address of x. (2marks)

   h. Print the address stored in ptr. (2marks)

Lab 3.1: Arrays/ Strings/ pointers

Objectives of the lab are to:

a. Learn how to use arrays

b. Learn how to use strings

c. learn how to use pointers

Resources required

i. Computer

ii. Unit 3 reference links (optional)

Time required:

2 hours

Description of lab exercise/activity

1. You are required to start your compiler then from file menu select new file, then save it as lab31a.c then write a program that outputs array values using a pointer. Assume the array was declared as : (5marks)

    i. `int a[7]={5,8,9,3,7,3,8};`

2. Open another new file from file menu, save it as lab31b.c then write a program that passes an array to a function that compute the minimum from from the array elements. Assume the array was declared as; (5marks)

    i. `int a[6]={3,7,5,8,2,4};`

3. Open another new file from file menu, save it as lab31c.c then write a program that reads in and displays your first, second and last name. (5marks)

4. Open another new file from file menu, save it as lab31d.c then write a program that reads in and displays 10 real values using a two dimensional array. (7marks)

5. Write another program that outputs the address of the 6th array element and array element 7. (5marks)

Results and submission requirements

You will be required to submit your lab activity as per your instructor's direction

Assessment criteria

Apply the allocated marks

References or key links

see unit 3 for the links on links to reference notes (optional).

Unit Summary

Arrays are data structures that store data items that are of the same type. A for structure is used to manipulate the contents of an array. C offers different types of arrays also known as multidimensional arrays and this unit looked at a one and two dimensional arrays. Strings are also defined as array of characters. There are many functions that can be performed with strings like copying, concatenating, computing the length etc. This unit defined pointers as variables that point to another variable and stores the address of the "pointed to" variable. A pointer is a tool that makes C to be a powerful language. It is used when creating dynamic memory allocations, passing arguments to a function, among other uses. The lab work was meant to give us hands on experience.

Unit Assessment

Check your understanding!

1. Define the following;

i. A pointer ii. A string (4marks)

2. Answer the following questions regarding an array called **myarray**

i). Define a symbolic constant **SIZE** to be replaced with the value 7 (2 marks)

ii). Declare **myarray** with **SIZE** elements of type **float** and initialize the elements to 0. (2marks)

iii). Name the fourth element from the beginning of array **myarray** (2marks)

iv). Assign the value **2.7** to array element six and value **5.4** to the seventh element of the array (4marks)

3. Declare pointer ptr1 and ptr2 to an integer (2marks)

4. Initialize ptr1 to the address of variable b below;

int b=7; and initialize the pointer *ptr*2 to value pointed to by pointer *ptr*1. (4marks)

5. Write a program that prints the contents and addresses of objects in 4 above (6marks)

6. Differentiate between an array and a pointer (2marks)

Instructions

Answer all the questions

If you score below 60% then you will be required to reread the entire unit activities.

Grading Scheme

All the unit activities will be computed to 5%

Solutions:

Q1

i. A pointer is a variable that points to another variable and stores the address of the variable pointed to by the pointer

ii. A string is an array of characters which is terminated by a null value

Q2

i. #define SIZE 7

ii. float myarray[SIZE]={0};

iii. myarray[3];

iv. myarray[6]=2.7; myarray[6]=5.4;

Q3

```
Int ptr1, ptr2;
```

Q4

```
ptr1=&b;
*ptr2=*ptr1;
```

Q6 An array is a static data structure whereas a pointer is dynamic data structure

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

- Deitel, P. and Deitel, H. (2016). C How to program with an introduction to C++. 8/e. Pearson Education.
- (Pages and chapters to be referenced on this book are indicated in the content)
- Chhabra, K. J. (2010). C programming concepts with problems and solutions. Published by Tata McGraw Hill Education Private Limited.
- New Delhi. pp179-261
- https://scs.senecac.on.ca/~btp100/pages/content/sarra.html
- http://www.le.ac.uk/users/rjm1/cotter/page_62.htm
- http://beginnersbook.com/2014/01/2d-arrays-in-c-example/

Unit 4. C structures and File processing

Unit Introduction

We have learned and known how to create variables, arrays and apply the operators. This unit also is very critical as it will take us through other C structures that will help us to efficiently and effectively organize our data. It will give us an experience on how to create or invent our own data types. Remember we have been using built in types and we have been using arrays and variables to store data. The data stored in this structures may not be permanent. Now let us move a step further and learn how we can store data permanently with the use of files. We will also get hands on experience on how to create and manage files.

Unit Objectives

Upon completion of this unit you should be able to:

- write a program using structures and unions
- Use pointers and structures to link the records
- write a program that can open, read and close a file
- apply file access modes.

Key Terms

Structures: "Structure is a user-defined data type in C which allows you to combine different data types to store a particular type of record. Structure helps to construct a complex data type in more meaningful way. It is somewhat similar to an Array. The only difference is that array is used to store collection of similar data types while structure can store collection of any type of data". (http://www.studytonight.com/c/structures-in-c.php).

Union: A union is a special data type available in C that allows to store different data types in the same memory location (http://www.tutorialspoint.com/cprogramming/c_unions.ht).

fOpen(): function to create a new file or to open an existing file (http://www.tutorialspoint.com/cprogramming/c_file_io.htm.

fClose(): used to close a file

File: A file is a collection of bytes stored on a secondary storage device, generally a disk (https://www.sites.google.com/site/projectdala/c-file-processing).

Learning Activities

Activity 4.1 - C structures

Introduction

We have been using built in C data structures. Now, this activity will allow us to define our own data types and we need to embark on this without delays.

Introduction to Structure

We have just concluded on arrays where we learned that an array stores data items that are of the same type. On the contrary, a structure is a user defined data type that can store different types of data. They are used to create records. For example, student records can be: name. registration number, age, course etc. As you can see, these records are of different types.

Defining a Structure

Structure is defined using the struct keyword. This keyword defines a new data type which can handle various data types. structure has the following form:

```
struct structure_ tag {

   member definition;

   member definition;

   ...

   member definition;

} structure variables;
```

The members are represented as the normal variables we know and have encountered. For example int age, double balance etc. The structure variables are declared after the closing curly brace but before the semicolon, see structure for Stud_rec below (studentDetails is the structure variable). Variables also can be defined later in the program. For example;

```
struct Stud_rec {

   char  name[50];

   char  course[50];

   float   feeBals;

   int   age;

}studentDetails;
```

Accessing Structure Members

Structure members are accessed using the member access operator (.). For example;

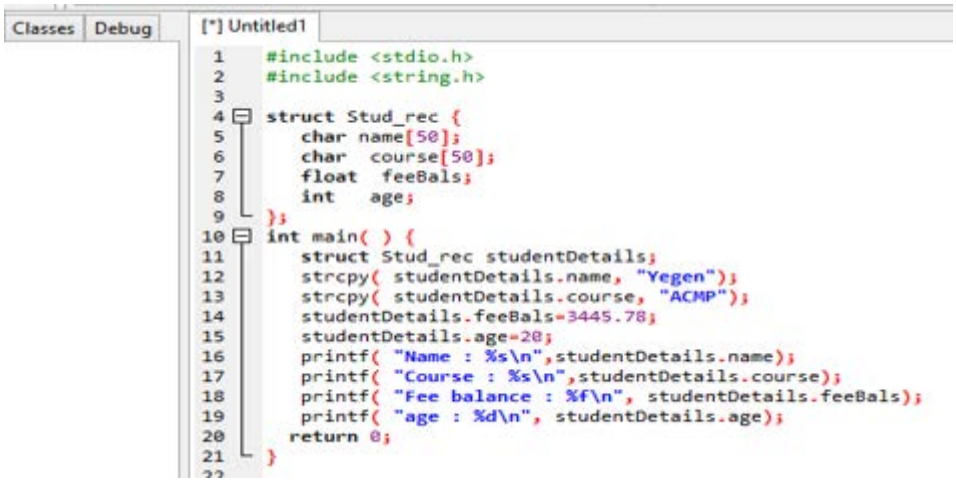
`studentDetails.name`

`studentDetails.course`

`studentDetails.feeBals`

`studentDetails.age`

see the program below

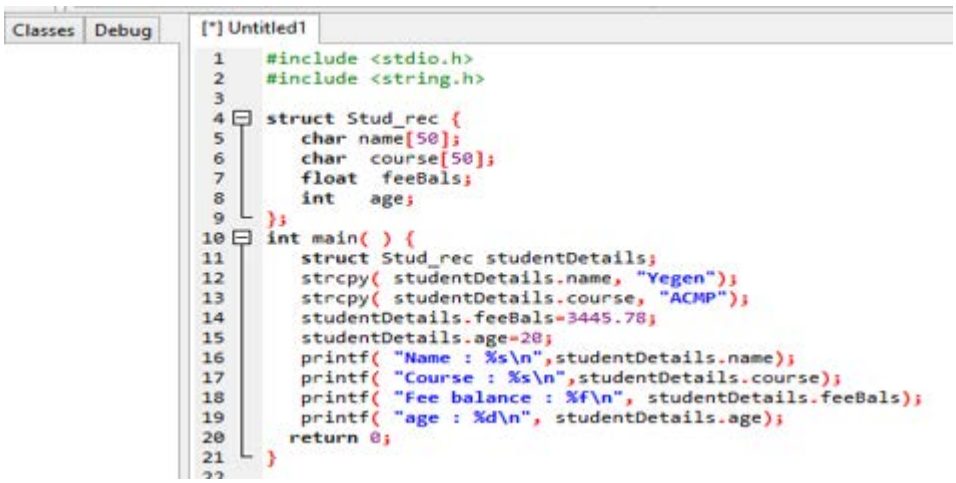


output will be:

Name	Yegen
Course	ACMP
Fee Balance	3445.78;
Age	20;

Structures as Function Arguments

Structures can be passed as arguments to a function. See program below



You are required to read Deitel and Deitel C programming textbook as referenced in this unit pages 405-411. Read and run all the programs presented in the book on C structures.

Conclusion

Unlike arrays, structures and unions are able to handle data items of different types. They are user defined data types. Structures are used to create records.

Assessment

1. What is a structure? (2marks)

2. Differentiate between an array and a structure. (2 marks)

3. Write a program that defines a structure tagged stud_rec with the following members and that assigns and prints values of two students: (8marks)

```
char name[20];

char programe[20];

char campus[20];

int age;
```

4. In relation to the above program, create a function for printing the values of the structure.

Activity 4.2 - Unions

Introduction

A C structure stores different data items in different memory locations. A union on the other hand is efficient and saves computer resources like memory. This activity will define a union, give its uses and show how it works

Unions

"A union is a special data type available in C that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose" (http://www.tutorialspoint.com/cprogramming/c_unions.htm)

Defining a union

Just like structures, a union can be defined using the union keyword. The union syntax looks as follows:

```
union union_tag {

   member definition;

   member definition;

    member definition;

} union variables;
```

Members are defined like any other variable for example int a, int c or char s. Union variables can be declared after the closing curly brace and before the semicolon, see union below-- munion is the variable for the union named Myunion. Let us define a union:

```
union Myunion {
   int age;
   float height;
   char name[20];
} munion;
```

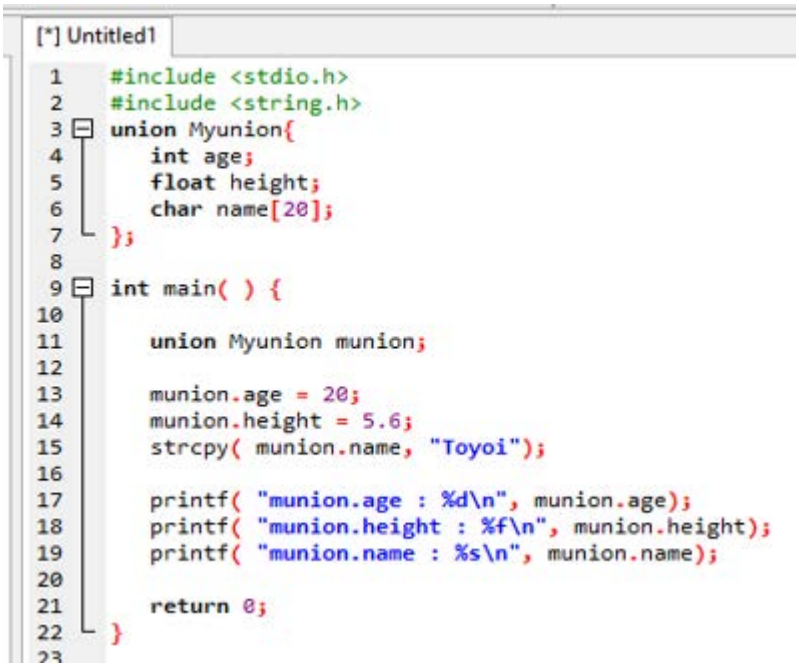
The variable of Myunion type can be used store age, height and name (all of different types). This shows that union variable has the capacity to store various types of data. Note that the union's memory space is defined or based on the largest member of the union

Accessing Union Members

Union members can be accessed by member access operator (.). For example;

```
munion.age = 20;
munion.height = 5.6;
strcpy( munion.name, "Toyoi");
```

See program below;



This program will output:

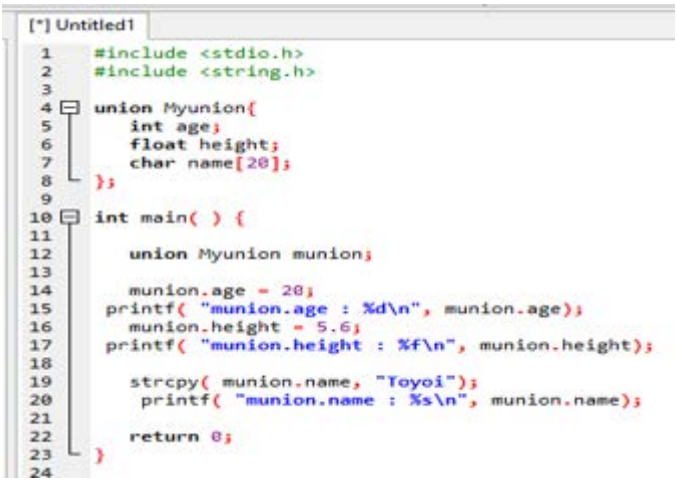
munion.age: 223457780

munion.height : 987732546

munion.name: Toyoi

The output is correct because the memory space was occupied by the name while the rest got disrupted.

Let us correct this mess by rewriting the program;



Now, one member uses the memory at a time. Thus the output will be:

munion.age: 20

munion.height: 5.6

munion.name: Toyoi

You are required to read Deitel and Deitel C programming textbook as referenced in this unit pages 414-415. Read and run all the programs presented in the book on C unions.

Difference between Union and Structure

A union is a class all of whose data members are mapped to the same address within its object. The size of an object of a union is, therefore, the size of its largest data member.

In a structure, all of its data members are stored in contiguous memory locations. The size of an object of a struct is, therefore, the size of the sum of all its data members.

Unions lead to memory space efficiency. This gain in space efficiency, while valuable in certain circumstances, comes at a great cost of safety: the program logic must ensure that it only reads the field most recently written along all possible execution paths. The exception is when unions are used for type conversion: in this case, a certain field is written and the subsequently read field is deliberately different (https://en.wikipedia.org/wiki/).

An example illustrating this point is:

```
struct { int a; float b; }  gives;

   union { int a; float b; }  gives;
```

Conclusion

A union is a data type in C that allows to store different data in the same memory location. A union can be defined with many members, but only one member can contain a value at any given time. It provide provides an efficient way of using the same memory location for multiple-purpose.

Assessment

1. Differentiate between unions and structures (4marks)
2. What are some of the advantages of using unions and structures (4marks)
3. What are some of the uses of structures and unions (6marks)

Activity 4.3 - File processing

Introduction

We have been storing data in variables and array but now we want to learn other ways of storing data in a more permanent manner. This will entail the use of Files. Various operations can be performed on files like opening, closing, appending or editing. We may also want to manage files by applying access properties. This activity aims to show us how to achieve these file operations

The reading for this activity was adapted from http://www.le.ac.uk/users/rjm1/cotter/page_73.htm. Except for some few definitions.

The Stream File

"A file is a collection of bytes stored on a secondary storage device, generally a disk. A byte consists of 8 bits.A bit is either off or on, i.e it can have only two states which is 1 and 0. These bytes can be interpreted as decimal digits, letters or special symbols.A file is durable i.e. it is available for the programs even after the current program which created the file terminates" (https://www.sites.google.com/site/projectdala/c-file-processing).

Although C does not have any built-in method of performing file I/O, the C standard library contains a very rich set of I/O functions providing an efficient, powerful and flexible approach.

A very important concept in C is the stream. In C, the stream is a common, logical interface to the various devices that comprise the computer. In its most common form, a stream is a logical interface to a file. As C defines the term "file", it can refer to a disk file, the screen, the keyboard, a port, a file on tape, and so on. Although files differ in form and capabilities, all streams are the same.

A stream is linked to a file using an open operation. A stream is disassociated from a file using a close operation. The current location, also referred to as the current position, is the location in a file where the next file access will occur. There are two types of streams: text (used with ASCII characters some character translation takes place, may not be one-to-one correspondence between stream and what's in the file) and binary (used with any type of data, no character translation, one-to-one between stream and file).

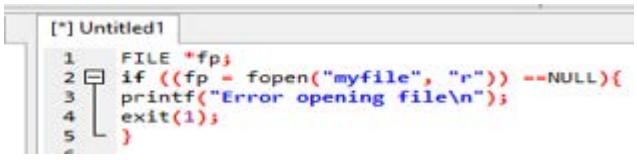
To open a file and associate it with a stream, use fopen(). Its prototype is shown here:

```
FILE *fopen(char *fname,char *mode);
```

The fopen() function, like all the file-system functions, uses the header stdio.h . The name of the file to open is pointed to by fname (must be a valid name). The string pointed at for mode determines how the file may be accessed as shown:

Mode	Meaning
r	Open a text file for reading
w	Create a text file for writing
a	Append to a text file
rb	Open a binary file for reading
wb	Open a binary file for writing
ab	Append to a binary file
r+	Open a text file for read/write
w+	Create a text file for read/write
a+	Append or create a text file for read/ write
r+b	Open a binary file for read/write
w+b	Create a binary file for read/write
a+b	Append a binary file for read/write

If the open operation is successful, fopen() returns a valid file pointer. The type FILE is defined in stdio.h. It is a structure that holds various kinds of information about the file, such as size.The file pointer will be used with all other functions that operate on the file and it must never be altered or the object it points to. If fopen() fails, it returns a NULL pointer so this must always be checked for when opening a file. For example:



To close a file, use fclose(), whose prototype is

```
int fclose(FILE *fp);
```

The fclose() function closes the file associated with fp, which must be a valid file pointer previously obtained using fopen(), and disassociates the stream from the file. The fclose() function returns 0 if successful and EOF (end of file) if an error occurs.

Once a file has been opened, depending upon its mode, you may read and/or write bytes to or from it using these two functions.

```
int fgetc(FILE *fp);

int fputc(int ch, FILE *fp);
```

The getc() function reads the next byte from the file and returns its as an integer and if error occurs returns EOF. The getc() function also returns EOF when the end of file is reached. Your routine can assign fget()'s return value to a char you don't have to assign it to an integer.

The fput() function writes the bytes contained in ch to the file associated with fp as an unsigned char. Although ch is defined as an int, you may call it using simply a char. The fput() function returns the character written if successful or EOF if an error occurs.

You are required to read Deitel and Deitel C programming textbook as referenced in this unit pages 442-447. Read and run all the programs presented in the book on C file processing.

Conclusion

Unlike other data structures, like arrays, structures and unions, including variables files are used to store data in a more permanent manner. A file is a collection of bytes stored on a secondary storage device, generally a disk. "A file is durable i.e. it is available for the programs even after the current program which created the file terminates" (https://www.sites.google.com/site/projectdala/c-file-processing). There are a number of functions that can used while performing operations on files which include: fOpen(), fClose(), among others.

Assessment

Using a very simple illustration without writing a program show how to;

i. Create a file "names.txt" (2marks)

ii. Read the file names.txt (2marks)

iii. Close the file name.txt (2marks)

iv. convert the file to read only (2marks)

v. convert the file to write only (2marks)

vi. make the file with both read and write access modes (2marks)

Unit Summary

This unit looked at arrays, structures and unions. In comparison to arrays, structures and unions are able to store data items that are of different types. Structures and unions are used to create user defined data types hence can be termed as user defined data types. Unlike structures, unions are data types in C that can handle different data in the same memory location but only one member can contain a value at any given time. It provide provides an efficient way of using the same memory location for multiple-purpose. On the other hand, files are used to store data in a more permanent manner. A file is a collection of bytes stored on a secondary storage device, generally a disk. There are a number of functions that can used while performing operations on files which include: fOpen(), fClose(), among others.

Unit Assessment

Check your understanding!

1. Define the following:
(6marks)

i. a structure

ii. Union

iii. File

2. What are some the uses of a union and structures (4marks)

3. Differentiate between structures and unions (4marks)

4. Write a simple program to illustrate how a file can be opened for reading and writing
(8marks)

Instructions

Answer all the questions

If you score below 60% then you will be required to reread the entire unit activities.

Grading Scheme

Grade as per the awarded marks

All the unit activities will be computed to 5%

Answers

Solutions:

Q1 i.A structure is a collections of related variables under one name. Structures may contain variables of many different data types—in contrast to arrays, which contain only elements of the same data type. Structures are commonly used to define records to be stored in files.

ii.A union is a derived data type—like a structure—with members that share the same storage space.

iii.A file is a collection of bytes stored on a secondary storage device, generally a disk. A byte consists of 8 bits.A bit is either off or on, i.e it can have only two states which is 1 and 0. These bytes can be interpreted as decimal digits, letters or special symbols.A file is durable i.e. it is available for the programs even after the current program which created the file terminates

Q2 Creating records

Minimizing memory space usage

Q3A union is a class all of whose data members are mapped to the same memory location and the size of an object of a union is, therefore, the size of its largest data member whereas in a structure, all of its data members are stored in contiguous memory locations. The size of an object of a struct is, therefore, the size of the sum of all its data members.

Q4Student to write correct program (compile and run the program to ascertain its correctness)

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

- Deitel, P. and Deitel, H. (2016). C How to program with an introduction to C++. 8/e. Pearson Education.
- (Pages and chapters to be referenced on this book are indicated in the content)
- Chhabra, K. J. (2010). C programming concepts with problems and solutions. Published by Tata McGraw Hill Education Private Limited.
- New Delhi. pp289-315
- https://scs.senecac.on.ca/~btp100/pages/content/sarra.html
- http://www.le.ac.uk/users/rjm1/cotter/page_62.htm
- http://beginnersbook.com/2014/01/2d-arrays-in-c-example/

Module summary

Structured programming was necessitated by crisis in software industry where software did not meet the functionalities, it was delivered late and exceeded the budget. We noted various advantages that come with structured programming like; programs are more easily and more quickly written, programs have greater reliability, programs require less time to debug and test, and programs are easier to maintain. This unit described C as a robust programming language and can be used to develop Operating Systems, language Compilers, assemblers, text editors, print spoolers, network drivers, modern programs, data bases, language Interpreters, utilities among others. The unit also looked at the structure of C language, how to invent variables and apply appropriate data types. We defined a variable as a location in the memory (space) where we can store data while a data type define the kind of data stored in a particular memory location. C supports a number of these operators for example arithmetic operators, assignment operators, logical operators among others. Later we proved that a program is not limited to linear (sequential) way of execution. At times it is required to make decisions using conditional statements or repeat a statement (instruction) several times with the aid of repetition statements. Conditional statements aid in decision making or branching in a program.It entails evaluated of an expression and if condition is met then the intended statement is executed. C also obtains its power by the use of functions. Functions has various benefits include, code reuse, ease of program documentation, debugging, readability and understandability among others. C language supports two types of functions, library function which are built in functions and user defined function; functions that are invented by the developer based on the nature of the program. Arrays were defined as data structures that store a series of data items that are of the same type. Strings also defined as array of characters. There are many functions that can be performed with strings like copying, concatenating, computing the length etc. This unit defined pointers as variables that point to another variable and stores the address of the pointed to variable. A pointer is a tool that makes C to be a powerful language. It is used when creating dynamic memory allocations, passing arguments to a function, among other uses. The unit included a lab work which was meant to give us hands on experience.

Course Assessment 1

Knowledge and comprehension assessment

Define the following: (14marks)

i. an identifier

ii. an array

iii. a pointer

iv. a string

v. Structure

vi. a union

vii. a file

2. In relation to C operators, answer the following questions (i to ii).

i. What is an operator? (2mark)

ii. Name two categories of operators in programming (2marks)

3. Problems i to iii below refer to the following statements (assume the program is complete (6marks)

i. What would you see on the screen if these statements are executed?

ii. What is the value of count after execution of the for loop?

iii. How many times is the for loop executed?

4. Convert the following while statement to a for statement (4marks)

5. For each of the following, write a single statement that performs the indicated task. Assume that integer variables x and y have been declared, and that x has been initialized to 3

a) Declare the variable ptr to be a pointer to an object of type integer (1mark)

b) Assign the address of variable x to pointer variable ptr (1mark)

c) Print the value of the object pointed by ptr (2marks)

d) Assign the value of the object pointed to by ptr to variable y (2marks)

e) Print the value of y (2marks)

f) Print the address of x. (2marks)

g) Print the address stored in ptr. (2marks)

6. Answer the following in relation to functions:

i. What is a function prototype? (1mark)

ii. Differentiate between formal and actual parameters (2marks)

7. What is the output of the following code. (5marks)

```
int a, b, c, d=5;
a=++d;
b=a++;
c=b--;
printf("%d %d %d %d", a, b, ++c, d, --d);
```

8. Using illustrations, discuss three elements of structured programming (9marks)

9. Differentiate between:

i. a union and a structure (2marks)

ii. a file and an array (2marks)

iii. an array and a structure (2marks)

10. Give the syntax for: (6marks)

i. file opening

ii. structure definition

iii. a function

11. IBM developed Syetem/360 and software developers created an Operating system for System/360 known as OS/360. The OS/360 had many problems and the result was called software crisis.

i. what do you understand by the term software crisis? (2mark)

ii. Name three factors that were associated with software crisis. (3marks)

Instructions

You are required to answer all the questions in course assessment 1.

If your score in any of the assessments fall below 50% then you be required to do a thorough revision of the course work then redo the assessment.

Grading Scheme

All the questions have been assigned marks

Feedback

Q1

i. an identifier is a variable name or a unique name that identifies a particular memory location

ii. an array is a series of data items that are of the same type; stored in a contiguous manner and are indexable

iii. a pointer is a variable that stores the address of another variable. It is a dynamic data structure

iv. a string is an array of characters which is terminated by a null value

v. Structure is a collections of related variables under one name.

Structures may contain variables of many different data types—in contrast to arrays, which contain only elements of the same data type. Structures are commonly used to define records to be stored in files.

vi. a union is a derived data type—like a structure—with members that share the same storage space.

vii. a file is a collection of bytes stored on a secondary storage device, generally a disk. A byte consists of 8 bits.A bit is either off or on, i.e it can have only two states which is 1 and 0. These bytes can be interpreted as decimal digits, letters or special symbols.A file is durable i.e. it is available for the programs even after the current program which created the file terminates

Q2 i.An operator is a symbol/character that tells the compiler to perform a particular mathematical operation

ii.Unary and binary operators

Q3

i.`Sum=105`

ii.`count =15`

Iii.`15 loops`

Q4

```
for( int k=3; k<=20; ++k)
{
printf("%d", k);
}
```

Q5

a. `int ptr`

b. `ptr=&x`

c. `printf("%d\n",*ptr);`

d. `y=*ptr`

e. `printf("%d\n",y);`

f. `printf("%d\n", &x);`

g. `printf("%d\n",ptr);`

Q6

i A function prototype is a declaration of a function

ii Actual parameters are the arguments passed to a function whereas formal parameters are the placeholders for the actual parameters (store actual parameters)

Q7Output:

10 8 9 8 8 9

Q8Student is required to discuss top-down analysis, modular programming and structured coding

Q9i. a union stores data in one memory location whereas a structure stores data in different memory locations

ii. a file stores data in a more permanent way whereas an array stores data temporarily iii. an array holds data that are of the same type whereas a structure stores data of different types

Q10i. f`open();`

ii.
```
struct struct_tag_name{
Structure data members;
}variable;
```

iii.
```
function_return_type function_name(parameter list)
{
function definition/body;
}
```

Q11

i.Software crisis can be defined as too many cases of mulfuning programs

ii.Late delivery, over budget and not meeting user's requirements

Course Assessment 2

Applications assessment

1. Write a program to convert meters to miles. (Recall that 1mi= 1.6093440km) (5marks)

2. Using continue statement, write a program that outputs: 20, 19, 17,16,15. (4marks)

3. Write a program that increments time by 2 if distance is greater than or equal to 100.0; if distance is between 50 and 100, increment time by 1. Otherwise, increments time by 0.5. (5marks)

4. Write a program that stores data in a structure and displays the values. Assume the structure stores two values 100 and 101. (6marks)

5. Write a program that contains the following series of elements. Print all the elements of the array. Show the output. (6marks)

```
int shelfNumber [] = {12, 13, 14 15 16};
```

6. In reference to the above shelfNumber array,

i. Give the fifth element of the array (2marks)

ii. Replace the value in array element three with number 16 (2marks)

7. Write a program that gives the following output; (8marks)

sqary		0	1	2	3
	0	0	0	0	0
	1	0	1	2	3
	2	0	1	4	6
	3	0	3	6	9

8. Write a program that computes and returns the volume of a cube to the main function (5marks)

9. Write a program that passes an integer to a function which returns sum of the values less or equal to the keyed in value (6marks)

10. Write a program that reads in values and compares them. If the first value is greater than the second value and second value is greater than the third value the program calculates and outputs the sum of the three values else it subtracts third value from the first value (5marks)

11. C program can perform a number of operations with strings. Using the following strings,

string1 Bachelor of Science

string2 in applied computer science

string3 ?

Write a program that can: (16 marks)

i. Copying the first string to string3

ii.Concatenating string3 and string2

iii. Length of str3

iv. Comparing str1 with str2

12. Write a program that creates a file known as "mystory.txt" for reading and writing. (10marks)

13. Using unions, write a program that stores and outputs student details: name, age, place of birth, height and weight. (12marks)

Instructions

You are required to answer all the questions in course assessment 2.

If your score in any of the assessments fall below 50% then you be required to do a thorough revision of the course work then redo the assessment.

Grading Scheme

All the questions have been assigned marks

Course References

- Deitel, P. and Deitel, H. (2016). C How to program with an introduction to C++. 8/e. Pearson Education.
- (Pages and chapters to be referenced on this book are indicated in the content)
- Chhabra, K. J. (2010). C programming concepts with problems and solutions. Published by Tata McGraw Hill Education Private Limited. New Delhi.
- https://scs.senecac.on.ca/~btp100/pages/content/sarra.html
- http://www.le.ac.uk/users/rjm1/cotter/page_62.htm
- http://beginnersbook.com/2014/01/2d-arrays-in-c-example/

The African Virtual University Headquarters

Cape Office Park

Ring Road Kilimani

PO Box 25405-00603

Nairobi, Kenya

Tel: +254 20 25283333

contact@avu.org

oer@avu.org

The African Virtual University Regional Office in Dakar

Université Virtuelle Africaine

Bureau Régional de l'Afrique de l'Ouest

Sicap Liberté VI Extension

Villa No.8 VDN

B.P. 50609 Dakar, Sénégal

Tel: +221 338670324

bureauregional@avu.org