African Virtual University

Applied Computer Science: CSI 1104

PRINCIPLES OF COMPUTER 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

Jules Degila

Instructional Designers

Elizabeth Mbasu

Benta Ochola

Diana Tuel

Media Team

Sidney McGregor	Michal Abigael Koyier
Barry Savala	Mercy Tabi Ojwang
Edwin Kiprono	Josiah Mutsogu
Kelvin Muriithi	Kefa Murimi
Victor Oluoch Otieno	Gerisson Mulongo

Copyright Notice

Supported By

AVU Multinational Project II funded by the African Development Bank.

Table of Contents

Course Overview

Welcome to Principles of Programming

You have heard of computer games, right! Probably you might have had a chance to play with a friend or the computer itself. If you have not, try now (take five minutes); that computer you are using has a number of games including and not limited to Solitaire or Ink Ball. Wow! how does it work? how does it judge its moves? Outsmarts you ha! What are we thinking now--- it puzzles; there must be some logic behind this! True, there is, and that 'puzzle' is the act of programming. "A beginner in programming must emphasize the how of programming: how to develop the solution to a given problem, how to organize a program, and how to make effective use of the standard techniques that represent the "tricks" of trade" (http://users.csc.calpoly.edu).Therefore, this is a very interesting Module as we introduce what programming is and also familiarize ourselves with frequently applied terms in the programming world. This serves as an entry level programming course designed to teach students the basics of programming. The primary goal of the course is to learn how to efficiently solve programming problems and provide foundation of basic knowledge regardless of the programming language. It introduces the fundamental building blocks of programming such as variables, operators, control structures, arrays and subroutines. The student will learn how to apply problem solving techniques in programming through creating flowcharts and pseudo codes. A high level programming language (C) will be used to write small programs to reinforce concepts learned during design.

Prerequisites/Required Knowledge

This course has no prerequisite. However, knowledge on computer basics is an advantage to the learner. This is because, in programming, some elements of the computer are frequently referred to or mentioned for example computer memory. We have included references to materials including books that will guide the learner through the basic elements of a computer.

Materials

The materials needed to complete this course include:

- A Computer
- An Internet connection
- C compiler.

Course Goals

1. Upon completion of this course the learner should be able to:
2. explain what programming is and how it is used in problem solving
3. analyze problems and break it down into programmable units
4. design solutions to problems using standard methodology
5. code the design using C programming language
6. apply the debugging techniques.

Units

Unit 0: Computer Basics

This is a pre-assessment unit. It reminds and tests you on some of the concepts that you need and are assumed in the module. This module will assess basic computer skills including basic hardware components, software and data representation.

Unit 1: Introduction to programming

This unit serves as the entry point to programming. It involves history of programming, programming languages including generations of programming languages. The unit also gives an overview of programming paradigms

Unit 2: Computer based problem solving

The unit discusses problem solving techniques at a basic level including identification, analysis, and solution design (Top down design and bottom up program design). It also discusses how to translate solutions into flowcharts and/or pseudo code.

Unit 3: Programming in C Language

This unit discusses programming constructs like variable, operators, control structures, arrays, and subroutines. Simple programs are written using C language to reinforce all the concepts learned in this unit.

Unit 4: Program Testing, Debugging and Documentation

This unit discusses different types of errors that emerge during program development and how to debug. It also discusses the basics of documentation in software development.

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	Mid-term examination	10%
2	Assignments	10%
3	Practical	10%
4	Final exam	70%

Schedule

The duration for this module is 90 hours, divided into reading, research and hands-on activities, tutoring, formative and summary assessment.

Unit	Unit title	Activities	Estimated time
0	Preliminary assessment	Assessment	2 hours
1	Introduction to programming	Activity 1.1: History of programming	3 hours
		Activity 1.2: levels, generations and paradigms of programming languages	6 hours
		Activity 1.3: Teamwork activity	8 hours
		Unit Assessment	2 hours

2	Computer based problem solving	Activity 2.1: Problem solving	10 hours
		Activity 2.2: Software design	10 hours
		Activity 2.3: Group work study activity	10 hours
		Unit assessments	2 hours
3	Programming in C language	Activity 3.1: Data types, variables and operators	10 hours
		Activity 3.2: control structures and functions	10 hours
		Activity 3.3: Arrays and strings	10 hours
		Unit assessments	2 hours
4	Program testing, debugging and documentation	Activity 4.1: Program errors and testing	10 hours
		Activity 4.2: Program debugging	10 hours
		Activity 4.3: Software documentation	10 hours
		Unit assessments	2 hours
	Final Examination	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:

Seema Kedar, (2007). Programming Paradigms and Methodology. 3rd revised edition. Technical Publication Pune (pp 3-11 and pp 24-27).

ISRD, (2008). Programming & Problem solving using C. Tata McGraw-Hill Education (pp 25-29).

Gerard O 'Regan, 2012. A brief history of computing. 2nd edition. Springer London (pp 121-143).

Optional readings and other resources:

- http://www.softpanorama.org/History/lang_history.shtml#General
- https://theory.stanford.edu/~jcm/books/cpl-sample.pdf (page 6-13)

These two, optional reading resources contain important content on the history of programming languages. They will add more information to what you have already read from the main reading materials. They very good for activity 1.1.

Unit 2

Required readings and other resources:

- Priya H., and Ranjeet, R., (2006). Programming and problem Solving Through "C" Language. Firewall Media. (page 1-13 and 20-35)
- ISRD, (2008). Programming & Prob solving using C. Tata McGraw-Hill Education (chapter 1, 2, and 3).

Optional readings and other resources:

The following optional reading materials for this unit will help you (the student) to access several examples on computer problem solving techniques. Each book also, has a unique and special way of presentation (without contradicting) which may help you acquire more knowledge and experience. We highly recommend you to read these books to improve on your understanding and programming skills as well.

- Gary Marrel, (2009). Fundamentals of Programming: with Object Oriented Programming. Gary Marrer. (pp 47-56).
- Joyce Farrel, (2012). Programming Logic and design, Comprehensive. 7th edition. Cengage Learning. (pp 15-19).
- Baviskar, D. P. (2009. Fundamentals of programming languages. Technical Publication Pune. (Chapter 1)

Unit 3

Required readings and other resources:

- Priya H., and Ranjeet, R., (2006). Programming and problem Solving Through "C" Language. Firewall Media. (pp 36-61)
- ISRD, (2008) Programming and PROB solving using C. Tata McGraw-Hill education. (pp 43-65).

Optional readings and other resources:

This optional reading is recommended for you because it has examples that will help you learn and acquire more knowledge and skills.

- Programming: Grade in Industrial Technology Engineering. Creative Commons. On: http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf

Unit 4

Required readings and other resources:

- Baviskar, D. P. (2009. Fundamentals of programming languages. Technical Publication Pune. (Chapter 3 and 4).
- Priya H., and Ranjeet, R., (2006). Programming and problem Solving Through "C" Language. Firewall Media. (pp 13 -14).

Unit 0. Preliminary Assessment

Unit Introduction

This initial test assesses learners on basic computer skills. It is also meant to refresh student memory on important elements of the computer. This will allow learners quickly grasp other subsequent units of this course. This should not be a big deal especially for the learners who are interacting with the computers for the first time, meaning they have no prior knowledge in the area of computer.

Unit Objectives

At the end of this unit, you should be able to:

- explain the functions of basic computer hardware components
- describe how a computer represents data
- explain the types of computer software

Key Terms

A Computer: It is electronic machines that accept data and instructions from a user, store the data and instructions, manipulate the data according to the instructions and store and/or output the results to the user.

Digital computers: They represent data in the form of discrete values by operating on it in steps.

Analog computers: They process data in the form of electrical voltages.

Hardware: These are the physical, tangible pieces that we can see and touch

Central processing unit: A microprocessor that interprets and carries out instructions given by software. It controls the computer's components.

Control unit: It is a device that Coordinates and controls all parts of the computer system.

Computer memory: It is the term used to describe devices that enable the computer to retain information. Program instructions and data are stored in memory chips for quick access by the CPU.

Primary memory: It is a temporary storage place for data, instructions, and information. Memory stores the operating system, application programs, and the data processed by application programs.

Memory address: It is an index that uniquely identifies a memory byte or cell.

Secondary memory: This external storage device that is not on the motherboard but found either inside or outside the computer and store programs and data permanently or semi-permanently.

Computer bus: It is an electrical channel that allows various devices inside and attached to the system unit to communicate with each other or transmit signals/information.

Binary number system: It is a numbering system that the computer uses to represent data using two unique values; either 0 or 1.

A register: It is a storage device that temporarily stores the most frequently used instructions and data.

A bit: it is considered as the basic unit of information; represented by 1s and 0s (binary numbers).

Byte: Eight bits grouped together to represent a character for example an alphabetical letter, a number, or a punctuation symbol. It comprises 256 different combinations.

Software: Also known as a program is sequence of instructions to accomplish a result

System software: all programs related to coordinating computer operations for example operating systems, programming language translators and utility programs

Operating system: This is a set of programs containing instructions that coordinate all the activities among computer hardware devices

Application software: These are programs that perform specific tasks for users, such as a word processing program, e-mail program, or Web browser.

Unit Assessment

1. What is a computer?
2. It processes data in a discrete manner. Which computer is this?

   a) Analog b. Digital c. Hybrid d. Analog and digital

3. A computer accesses information from the main memory using;

   a) A unique byte b). A unique name c). A unique address

4. Which one/s is an input device

   a) Mouse b). Microphone c). Plotter d). Disk drives

5. Name all these parts of a computer
6. A computer main memory does;

   a) Stores all or part of the software program that is being executed

   Process data

   b) Stores the operating system programs that manage the operations of the computer.

c). Hold data that the program is using

d). Store information permanently

7. Programming language translator is;

a) An operating system b). A Utility software c). A System software d) none

Answers to the assessment

1. It is electronic machines that accept data and instructions from a user, store the data and instructions, manipulate the data according to the instructions and store and/or output the results to the user.
2. b
3. c
4. a, b, d
5. 1. CPU 2. Power supply 3. Fan 4. Hard drive 5. Storage bays 6. Cd/DVD drive 7. Floppy drive 8. Zip drive. 9. Memory (RAM) 10. Motherboard 11. Expansion slot 12. Expansion cards.
6. a, c, d
7. c

Unit 1. Introduction to Programming

Unit Introduction

"Computers are everywhere. Be it departmental stores, hotels, atomic power stations or defense. Computers aid human force in effectively administering a task. The growth of computers is exponential, so is the need for good programmers. By good programmers we do not mean programmers who can give solutions, but programmers who can give effective solutions. Good programmers are not born rather they are groomed. A programmer's skills wear out well with experience and practice" (Priya and Ranjeet, 2006).

Therefore this interesting unit introduces you to what programming is and also helps you to familiarize yourself with frequently applied terms in the programming world. "At the end of the unit you will be speaking the language of programmers". We are also interested in knowing how and probably when programming started, and where we are in programming; that is, the current state of programming in terms of programming language generations which includes types of programming languages and various programming styles also known as programming paradigms.

Unit Objectives

Upon completion of this unit you should be able to:

1. explain basic programming concepts
2. describe developments in programming languages and influences on evolution of language designs
3. describe the levels and generations of programming languages
4. describe the characteristics of a good language
5. describe characteristics of programming language paradigms

Key Terms

Programming: is the activity of writing instructions to be executed by the computer. These instructions tell the computer to do something. For example solve a particular problem, say, read in student marks then compute and grade scores per student and provide a list of students proceeding to the next class and another list of students retaking the course. All these are instructions (logic). A computer is not able to act without instructions. Also, programming is the art of performing computational tasks and converting these tasks into machine readable form (Priya and Ranjeet, 2006)

Programming language: is a series of instructions for writing programs.

Programming language Paradigms: define the way programs are structured or programming styles.

Programmer: is the person writing instructions or develops a program through the act of programming.

A program: is a series of instructions that directs the computer to do something. It is the end product of programming activity also known as software.

Syntax: of a computer language is the set of rules that defines the combinations of symbols that are considered to be a correctly structured document or fragment in that language (http://en.wikipedia.org).

Semantics: define the meaning of syntactically legal strings defined by a specific programming language, showing the computation involved (http://en.wikipedia.org).

Learning Activities

History of Programming

Introduction

Programming can be defined as the act of using a programming language to write instructions that can be executed by the computer with the aim of solving problems by applying the correct language syntax and semantics. Hundreds of programming languages have been developed and some are in existence while some are not. This means that programming language development has come a long way thus having long history; from Plankalkul, Prolog, FORTRAN, C, Visual Basic.Net, to Rust and to Swift. Some of these languages were improved and gained more popularity than others. Below is a summary of the history of programming languages as presented by Seema Kedar; see reference.

1951–55: Experimental use of expression compilers.

1956–60: FORTRAN, COBOL, LISP, Algol60.

1961–65: APL notation, Algol 60 (revised), SNOBOL, CPL.

1966–70: APL, SNOBOL 4, FORTRAN 66, BASIC, SIMULA, Algol 68, Algol-W, BCPL.

1971–75: Pascal, PL/1 (Standard), C, Scheme, Prolog.

1976–80: Smalltalk, Ada, FORTRAN 77, ML.

Read: http://en.wikipedia.org/wiki/Plankalk%C3%BCl

Activity Details

This activity requires you to read on the history of programming; how programming started and the application of the program, their descendants - to the current languages. Influences on evolution of languages design and development. The readings can be obtained from http://www.softpanorama.org/History/lang_history.shtml#General and https://theory.stanford.edu/~jcm/books/cpl-sample.pdf (page 6-13). After reading, you are required to;

- List at least five problems that can be solved using a computer (just to check understanding on what programming is)
- Describe how programming started and the situations that led to development of programs
- Using a chart, describe the proliferation of programming languages; Name and describe the applications and significance of the programming languages. Show and describe all the descendants of a particular language.

Conclusion

This activity will help the learners appreciate the history of programming and also may spur learners to understand how to take advantage of situations and environments as they present themselves to propose and design new or improve on the existing languages.

Assessment

You can perform this assessment individually or as a group. This assessment is testing the just concluded activity.

1. According to what you know or read on the history of computing, when was the first programming language designed and what was its application and motivation?
2. What is the importance of learning history of programming?
3. What influenced the development of C and Fortran programming languages
4. By now, you are aware that hundreds of programming languages do exist. Some are widely used in the industry today and some are never mentioned at all or are being used by smaller group of programmers. why do you think most of these languages gained more popularity than others?

Levels, generations and paradigms of programming languages

Introduction

Programmers use programming languages to write programs. These programming languages have evolved over time with the earliest languages also known as first generation language which used machine code to program the computer - this can be dated back to 1940-1950. This language is termed as a low level language. The next development was the second generation language dated back to 1950-58 which used the assembly language to represent machine language instructions. These were then translated into machine code by an assembler. This language was still a low level language. Then the third generation languages, also known as high-level programming languages such as C, Pascal, FORTRAN and COBOL, can be dated back to 1958-85. These languages were easier to use than assembly languages and machine code and helped to improve quality and productivity. Compilers and interpreters are used to translate instructions in a high level language to machine language. There are also fourth generation languages dated 1985-onwards. These languages include report generators which are said to reduce programming effort. The fifth generation programming languages dated 1990-onwards are mainly used in the field of artificial intelligence.

Each generation of programming language has its own features and the newer generation languages are more improved than the older one. A programming language may be supported by more than one programming paradigm. Paradigm define how the program is structured (We shall discuss paradigms later in this unit). Figure below illustrates the levels of programming languages.

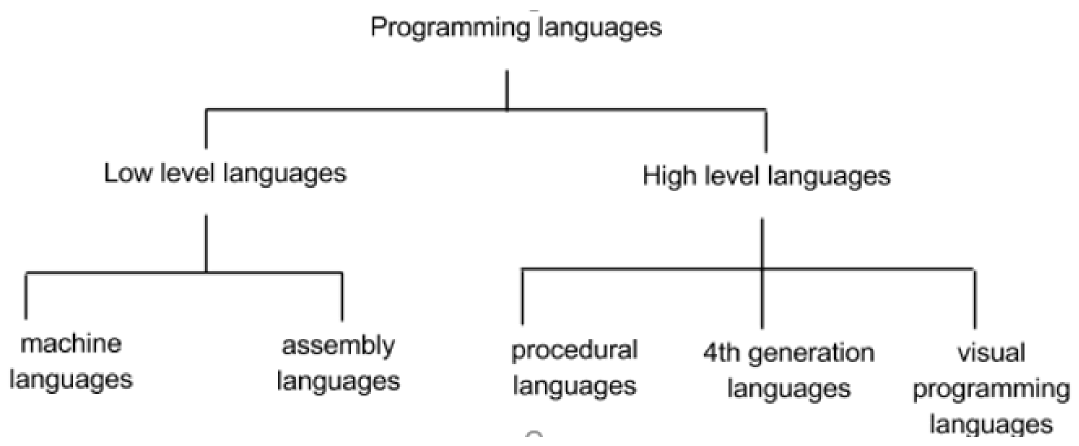


Figure 1.1 Levels of programming languages (source: ISRD)

The chart shows that there are two major levels of programming languages; low level and high level programming languages.

Machine language: Using this language, programs written using 0's and 1's. It was used in the earlier days to program a computer. It is faster because instructions are directly executable, and makes most efficient use of resources like the registers and storage units. Currently, this language is not highly appreciated because programs are not portable meaning they are machine dependent, it is more prone to errors and hard to debug, requires high level programming skills thus high cost of training.

Assembly language: This language uses mnemonics or short abbreviation representing an instruction. It is easier to use than machine language simply because mnemonics are nearer to the programmer than the computer (machine). Assembler is used to convert instructions to machine language.

High level languages: Programs are written in English like statements. They are not directly executable; therefore, translators which include compilers and interpreters are used to convert instructions to computer language. Languages that fall in this category are highly appreciated because, they are portable, easier to learn and write a program, availability of libraries, easy to maintain and document. High level languages have various attributes or characteristics which include; portability, readability, concurrency support, mixed language support, reliability, modularity, real-time support and orthogonality.

Fourth generation languages: These languages also known as 4GL emphasize on what is to be accomplished than how it should be accomplished. For example Oracle, VB, SQL e.t.c. They are mainly used to access the database. 4GLs increases productivity. 4GL has various features that make it attractive. These include;

- Ease of use
- Limited range of functions
- Availability of options
- Default options

Visual programming languages (VPL): is any programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually. A VPL allows programming with visual expressions, spatial arrangements of text and graphic symbols used either as elements of syntax or secondary notation. For example, many VPLs (known as dataflow or diagrammatic programming) are based on the idea of "boxes and arrows", where boxes or other screen objects are treated as entities, connected by arrows, lines or arcs which represent relations. An instructive counterexample for visual programming languages is Microsoft Visual Studio. The languages it encompasses (Visual Basic, Visual C#, Visual J#, etc.) are commonly confused with, but are not visual programming languages. All of these languages are textual and not graphical. MS Visual Studio is a visual programming environment, but not a visual programming language, hence the confusion (https://en.wikipedia.org/wiki/Visual_programming_language). See figure 1.2 below on VPL.

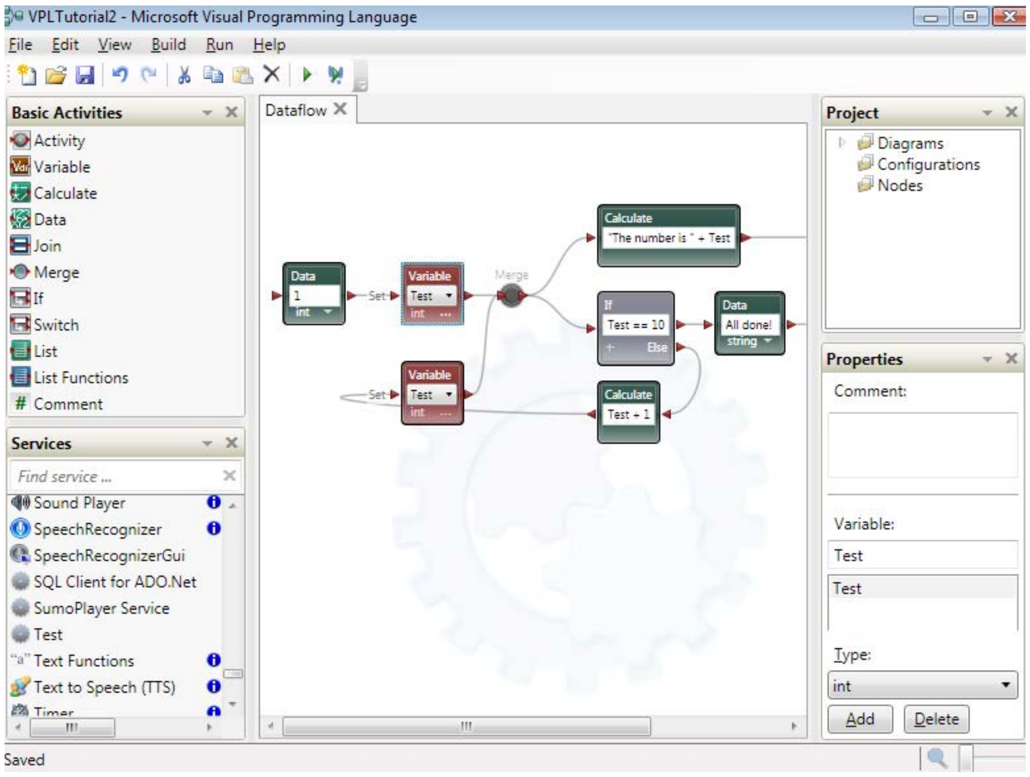


Figure 1.2 Sample Microsoft VPL diagram (Source:https://msdn.microsoft.com/en-us/library/bb483088.aspx.)

Access https://msdn.microsoft.com/en-us/library/bb483088.aspx to learn more on VPL.

Compilers

A compiler translates a high level program to a machine language. The high level program is called a source code, which is translated to produce machine readable code known as object code. Most or all high level languages are compiler based. For example, C compiler, C++, e.t.c

Interpreters

Just like the compilers, interpreters translate high level language program to machine readable format. The difference with the compilers is that, interpreters translate code statement by statement and if an error is encountered, it stops and will continue after the error has been corrected. While compilers translate the entire program, then list the errors, if any. This means that, compilers are faster than the interpreters.

Influences on evolution of language design

Seema Kedar discussed six influences;

1. Computer capabilities: Computers have greatly improved in the way they process information. There have been major improvements in the hardware and at the same time software (operating systems). These improvements call for the design of better languages.
2. Applications: Application means the use of computers. They started from performing small computations, handling military activities, scientific activities like research, medical, business to games and the internet. The requirements of the new application areas lead to the improvement of existing languages or the design of new languages
3. Programming methods: As problems become more complex, new methods are invented to aid reduce complexities and improve productivity.
4. Implementation methods: The development of better implementation methods has affected the choice of features to include in new language designs
5. Theoretical studies: Research into the conceptual foundation for language design and implementation using formal mathematical methods has increased our understanding of the strengths and weaknesses of language feature which has influenced the inclusion of these features in new language design
6. Standardization: The need for a standard language that can be implemented easily using a computer systems which allow programs to be translated from one computer to another has provided a strong influence on evolution of language designs.

Languages have different strengths. Some languages support particular applications than others. What therefore, define a good programming language?

1. Clarity, simplicity and unity
2. Orthogonality
3. naturalness for the application
4. Support for abstraction
5. Ease of program verification
6. Programming environment
7. Portability of the programs
8. cost, which include;

- cost of training
- cost of writing the program
- cost of executing the program
- cost of translation
- cost of implementation
- cost of maintenance

Programming language paradigms

Programming languages are structured in different ways. Therefore, languages can be grouped according to the way they are structured or designed simply known as paradigm. paradigms include;

1. Imperative or procedural languages: is a type of imperative programming in which the program is built from one or more procedures (also known as subroutines or functions).
2. Declarative languages: These languages contrast with imperative and procedural programming. Declarative programming is a non-imperative style of programming in which programs describe their desired results without explicitly listing commands or steps that must be performed.
3. Rule-based or logical languages: They support decision making based on the provided conditions
4. Object oriented languages: It is a paradigm that provides solutions that depict the real world scenario. It emphasizes more on data. It uses the concept of classes where an object is seen in terms of data (what it can process) and what it can do (methods).
5. Concurrent languages: They apply the concept of a process. A process corresponds to a sequential computation, with its own thread of control.

Read more on these paradigms from https://en.wikipedia.org/wiki/Programming_paradigm.

Activity Details

In this activity you are required to read on the levels (other authors call this types of programming languages) and generations of programming languages; note the characteristics, advantages and disadvantages of each type/level including generations of programming languages. Read on the characteristics of a good programming language. Read also on programming language paradigm; characteristics of each paradigm including languages that fit in each paradigm. We recommend Programming Paradigms and Methodology by Seema Kedar page 3 -11 and page 24 - 27 and Programming and PROB using C by ISRD page 25 - 29. After reading, write brief notes;

- Describing the levels and generations of programming languages and naming programming languages that fall into a particular level and generation.
- Describing the features of the programming languages that belong to a particular level and generation.
- Describing the characteristics of a good programming language. How is a good programming language defined?
- Describing the features of the programming languages that are grouped to a particular programming paradigm.

Hands-on activity

- Manually or using software, develop a chart that groups all the programming languages identified in activity 1.1 into the right level and generation of programming languages
- Manually or using software, develop a chart that groups the programming languages identified in activity 1.1 and in this activity into the right category of programming paradigm.

Conclusion

At this point of study, the learner should be well equipped with the knowledge on the characteristics, levels, generations and paradigms of programming languages. They should be able to describe programming languages that fit into each level, generation and paradigm.

Assessment

You can perform this assessment individually or in a group. This assessment is on the just concluded activity.

1. Machine language is known as a low level language and though assembler uses mnemonics, it is still referred to as a low level language. Why do you think so?
2. Programming languages are categorized according to generations. Why?
3. Describe at least four characteristics of a good programming language.
4. High level language can be translated to machine language using either a compiler or an interpreter. What are the advantages and disadvantages of a compiler and interpreter? Which of the two translators would you say is more efficient and why?
5. Describe the five programming language generations.
6. What influences the development of programming paradigms?
7. In your own opinion, what do you think is the future of programming languages? Do you think we shall have more generations and paradigms of programming languages?

Teamwork activity (collaboration)

Introduction

This activity entails a group activity where the learners will be required to understand the practicality of programming languages and paradigms. The learners are required to do a small study on the commonly used languages and the paradigms supported by those languages using accessible individual developer or software development firms or senior Applied computer science students in your institution especially the fourth year students.

Activity Details

By visiting individual developers and/or firms;

- Find out the most popular programming languages in the industry and the paradigms supported by the languages. Give reasons for their popularity.
- Find out how programmers select the programming language to be used in solving a particular problem.

<u>Conclusion</u>

The aim of this activity is to broaden learner's innovative capabilities and at the end, the learner should be able to appreciate the applications of various languages and paradigms and be able to collaboratively work with others.

<u>Assessment</u>

You can perform this assessment individually or in a group before attempting the formative assessments. This assessment is on the just concluded activity.

1. During your small study activity, which language was common and most preferred by developers?
2. What are some of the reasons for the preference of the mentioned (1 above) languages?
3. How do programmers choose the of language to be used when solving a particular problem?

Unit Summary

Computer programming has a very long history which can be dated back to 1940s or earlier. Computer programming (often shortened to programming) is a process that leads from an original formulation of a computing problem to executable computer programs. Programming involves activities such as analysis, developing understanding, generating algorithms, verification of requirements of algorithms including their correctness and resources consumption, and implementation (commonly referred to as coding) of algorithms in a target programming language. Source code is written in one or more programming languages. The purpose of programming is to find a sequence of instructions that will automate performing a specific task or solving a given problem. The notion of programming paradigms is a way to classify programming languages, according to styles of computer programming. Features of various programming languages determine which programming paradigms they belong to. Some programming languages fall into only one paradigm, while others fall into multiple paradigms. Some paradigms are concerned primarily with implications for the execution model, of the language, for example, whether the sequence of operations is defined by the execution model. Other paradigms are concerned primarily with the way that code is organized, such as grouping code into units along with the state that is modified by the code. Common programming paradigms include imperative which allows side effects, functional which does not allow side effects, declarative which does not state the order in which operations execute, object-oriented which groups code together with the state the code modifies, procedural which groups code into functions, logic which has a particular style of execution model coupled to a particular style of syntax and grammar, and symbolic programming which has a particular style of syntax and grammar. Upton here, the learner understands the basic concepts of programming and is ready to move to the next unit.

Unit Assessment

Check your understanding!

Formative assessments

1. Programming can be defined as the act of using a ______________________ to write_____________________that can be executed by the computer with the aim of ________________________________ by applying the correct programming language _____________________ and _____________________. (5 marks)

2. ___________________________was the first generation computer and used ____________________________language to write computer instructions (2 marks)

3. Which of the following languages uses mnemonics to represent instructions (1 mark)

   A. High level language B Low level language C. Assembly level language

4. A high level language uses ____________________________ or ___________________to translate instructions to machine language. (2 marks)

5. 5. Which of the following language/s support object oriented paradigm (1 mark)

   A. C B. C++ C. Pascal D. Java E. Visual Basic.Net

Instructions

The aim of this assessment is to check learner progress by determining how much the learner has learned in this unit. Questions cover everything that was presented in this unit in order to assess overall understanding. Answer them carefully and if your score falls

- below 40%, redo the readings.
- between 40% and 60%, redo the readings on your weak area
- above 60%, you have substantial amount of knowledge

Grading Scheme

The unit assessment (formative assessment) has been assigned marks. The instructor will assign marks to activity assessment then compute all. At the end of unit four all the assessments must be computed to 10% of the total examination marks.

Answers

Answers for unit assessment

6. programming language, instructions, solving a problem, Syntax and semantics
7. Computer language, low level
8. Assembly
9. Compilers or Interpreters
10. C++, Java, Visual Basic.Net

Unit Readings and Other Resources

- Seema Kedar, (2007). Programming Paradigms and Methodology. 3rd revised edition. Technical Publication Pune (pp 3-11 and pp 24-27).
- ISRD, (2008). Programming & Problem solving using C. Tata McGraw-Hill Education (pp 25-29).
- Gerard O 'Regan, 2012. A brief history of computing. 2nd edition. Springer London (pp 121-143).

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

Unit 2. Computer based problem solving.

"The sooner you start coding your program, the longer it is going to take" [H.F. ledgard, programming proverb]

Unit Introduction

In the previous unit, we defined programming as a way of using digital computers to solve problems with the sole purpose of making work easy and also increase productivity. Also in activity 1 we were tasked to identify few problems that can be solved using a computer; this unit will help you find out whether you did them right or not. Thus programming involves automating systems, processes, transactions or an activity. The solution (program) must satisfy end user needs or requirements. The end user can be a farmer in the village who needs daily updates on market prices including information that can help improve production. Therefore, for a solution to be realized, programmers need to first understand the problem and follow software development processes that entail; problem identification, analysis, solution design which can be achieved through the use of top down design or bottom up design techniques. Solutions can also be designed using flowcharts and algorithms and/or pseudo codes. Therefore, this unit looks at problem analysis and solution design techniques at a very basic level.

Unit Objectives

Upon completion of this unit you should be able to:

- formulate problems.
- design solutions by applying design techniques.
- design flowcharts.
- represent solutions using algorithms and pseudo codes.
- describe programming techniques

Key Terms

Algorithm: An algorithm is procedure consisting of a finite set of unambiguous rules (instructions) which specify a finite sequence of operations that provides the solution to a problem, or to a specific class of problems for any allowable set of input quantities (if there are inputs). In other word, an algorithm is a step-by-step procedure to solve a given problem.

Bottom up analysis: It is a design technique where a program is divided into a number of smaller and simpler tasks which can be tackled separately. In this technique, subprograms are written and tested first before the main program is written.

Design: design is the development of problem alternatives and models to represent problem and solution. It is a clear-cut process through which user requirements are transformed into a representation of machine understandable form.

Implementation: This involves coding and testing of the program

Flowchart: is a standardized technique for graphically representing graphics with a set of standardized symbols that represent program logic.

Deployment: deployment entails program installation in user's environment and training the users.

Logic path: is a sequence of instructions or statements also known as an algorithm

Maintenance: involve effecting changes on the program after installation

Model: is an abstract of reality

Module: is part of a program that perform a separate function

Modularization: entail breaking down of a large program into subprograms that are manageable in size in terms of complexity of logic and number of instructions.

Planning: it is the basic level of software development where the problem is defined and prioritized

Problem: is an undesirable situation that prevents users from fully achieving their objectives. A problem and the need to solve it arise from a desire to transform the current situation to a more desired one (programming and PROB solving using C by ISRD).

Program Development Process: is a standardized set of steps used to provide a logical, common sense approach (or process) to solving a difficult problem with a computer application.

Pseudo code: is a generic way of describing an algorithm without use of any specific programming language syntax. It is the representation of a solution using English like a programming language.

Program Logic: is used by programmer to model the programming language instructions carried out by the computer when the program is executed.

Requirements analysis: This is the second phase of software development after planning. It involves fact finding of problem specifications through questionnaires, interviews and observation among other methods.

Structured programming: is a programming paradigm that supports modularity by breaking the functions into trivial modules.

Programming Language Statements: are English like statements that when executed in the correct sequence can instruct the computer to perform a series of tasks. They are used to implement program logic by sending instructions to the operating system.

Software Development Life Cycle (SDLC): It is the methodology/process used by people who work in information technology to solve problems by providing effective and efficient solution.

Top down analysis: It is a design technique where a program is divided into a number of smaller and simpler tasks which can be tackled separately. In this technique, the main program is written and tested first before subprograms are written. Thus, it looks at the solution in a wider picture first.

Learning Activities

Problem solving

Introduction

In order to achieve a solution or develop software, a software developer is required to follow a sequence of steps. The first step of problem solving includes problem definition which entails understanding the nature of the problem at hand. This level of problem solving does not address 'how' to solve the problem, but tells us 'what' the problem really is. It involves understanding the input, operations and the expected output of the program. The following diagram illustrates the steps followed when solving a problem;

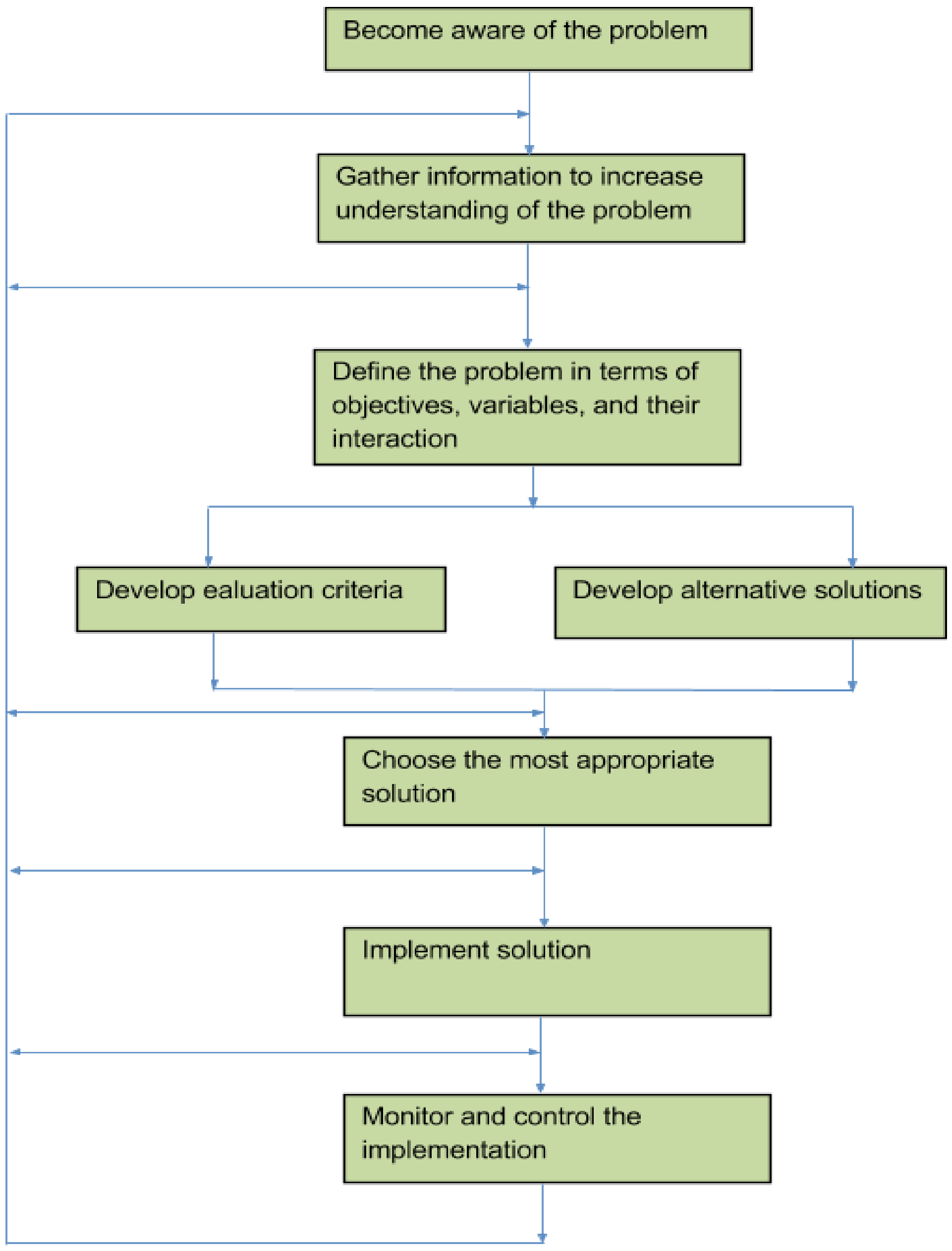

Become aware of the problem
Gather information to increase understanding of the problem
Define the problem in terms of objectives, variables, and their interaction
Develop ealuation criteria
Develop alternative solutions
Choose the most appropriate solution
Implement solution
Monitor and control the implementation

Figure 2.1 Steps in problem solving (source: Programming and PROB solving using C by ISRD)

"There is no cookbook strategies to replace intelligence, experience and good taste in programming", (Priya and Ranjeet, 2006). Thus we need to emphasize more on these steps and other problem techniques to allow us achieve our objectives which include production of software products that meet customer's needs, produce them on time and within budget.

Activity Details

Therefore, in this activity you are required to read the topics on what programming problems really are, why computers and other digital systems are used to solve problems, problem definition, similarities between problems, problem solving strategies, and the steps involved in computer problem solving. The readings can be obtained from (Programming & Problem solving using C by ISRD, Programming and problem Solving Through C Language by Harsha Priya, R. Ranjeet, and Problem solving and program design in C by Jeri R. Hanly & Elliot B. Koffman) and from the list of books provided in the reference section for this unit. Learners should also complement the readings by doing individual research in order to improve their understanding. In this activity you are required to write brief notes on;

- Why a computer is a necessary tool for problem solving
- Problem solving strategies and their benefits in programming.
- Software development process.
- Similarities between the problems that can be solved using a computer.

Conclusion

This activity is meant to help the learner gain deeper understanding of what problems are in programming, know the strategies and the steps for designing a solution and the similarities between problems.

Assessment

1. In relation to software development, what do you understand by the term problem?
2. What do you understand by the term problem definition?
3. Computers and other digital devices including mobile phones are used to solve problems. Give reasons why these devices are used as tools for problem solving.
4. Why should you apply strategies in problem solving? and what are these strategies? Discuss
5. Program development entail following some steps and one of the steps entail modeling of logic. However, it is possible to come up with a solution without adhering to these steps. Why is it necessary to apply software development process? Explain
6. Problem definition conveys input, output and operations of the program. Discuss.
7. Similarities between problems may include speed of processing, volume of work, reliability, accuracy, security, cost, and searching and string manipulations. Discuss these similarities.

Software Design

Introduction

Program design involves how to solve the problem and is a problem solving phase that comes after problem definition. It involves providing problem solutions by applying various design techniques including modeling. These techniques include breaking down of program into subprograms. This aspect of dividing a program into subprograms leads to modularization. Modularization on the other hand, is one of the basic elements of structured programming.

Algorithms

Algorithms are set of steps generated by the programmer to help in solving a particular problem. Because an algorithm is used as a design tool, it must therefore be finite, unambiguous, simple (definite) and precise, complete, effective and allow for input and output of data. Algorithms are programming language independent and can be developed using three major programming constructs which include;

1. sequential construct. The steps are written in a particular order, for example first, step A is executed then B and C comes after B e.t.c. This defines step by step flow of control. For example;

```
input a, b
sum=a+b
print sum
```

2. Conditional construct. Meaning at some point in the execution of steps, and to achieve a task, there is need to branch or make a decision by evaluating a condition. For example,

```
input a, b
sum=a+b
if (sum>d)
        then print sum
otherwise
        print d
```

Looping construct is. In this construct, a step is iterated a number of times before executing the next step or other steps based on a condition. When the condition turns to false, the loop is exited and other steps are then executed. For example;

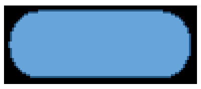
```
input a, b
sum=a+b
while (a<b)
      print a
input next a, b
repeat the evaluation
```

Remember we defined an algorithm as a finite set of instructions meaning they have a start and an end point. An example of an algorithm that adds two numbers.

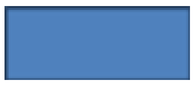
```
start
input values for a and b
add a to b and store in total
display value of sum
end
```

Flowcharts

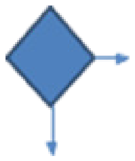
Flow chart is a pictorial representation of an algorithm. A flowchart is designed using the following symbols:



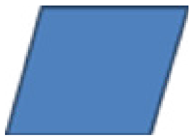
Start/ End



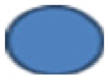
Processs



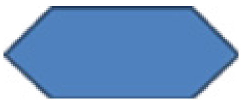
Decision Making



Input/ Output



Connector



Preparation (Looping)

Figure 1.4 Flow Chart representation of an algorithm

Now we can represent our algorithm (above) using a flow chart.

Two techniques that can be applied in software design (problem solving) include; top down design technique and bottom up design techniques.

Top down design technique

This design strategy eliminates the complexity of handling a big problem by breaking them into sub modules that can be easily understood. The process of dividing a task into sub-task is repeated until an easy to implement task is achieved. Figure below illustrates how top-down technique can be achieved.

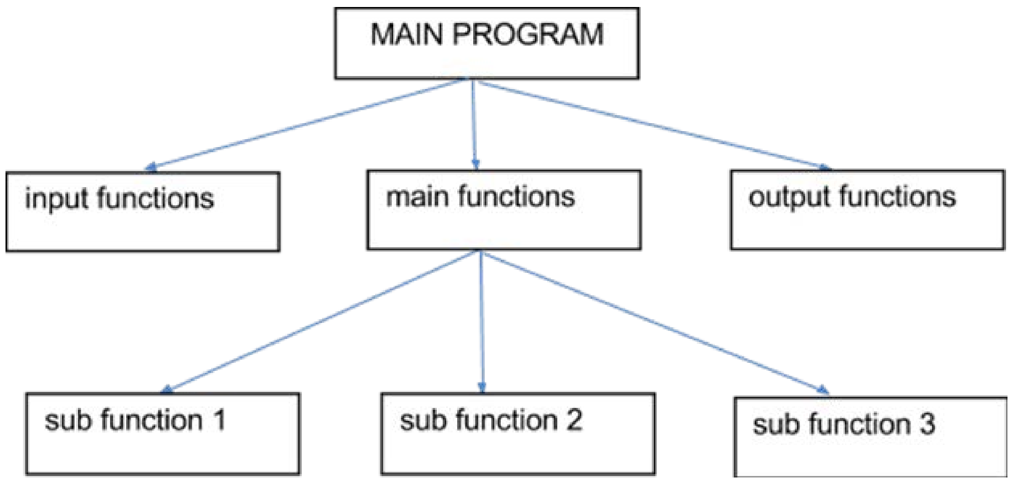


Figure 2.2 Top-down design development (source: Priya and Ranjeet, 2006)

Read programming and problem solving through c language by Priya H., and Ranjeet R, also, Programming and PROB solving using C by ISRD. Understand top down problem solving technique, its advantages and disadvantages.

Bottom-up design

Here, the most basic subroutines are written and later used to make sophisticated subroutines. Why is this approach not recommended on its own? It is like reversing top-down technique, see figure below.

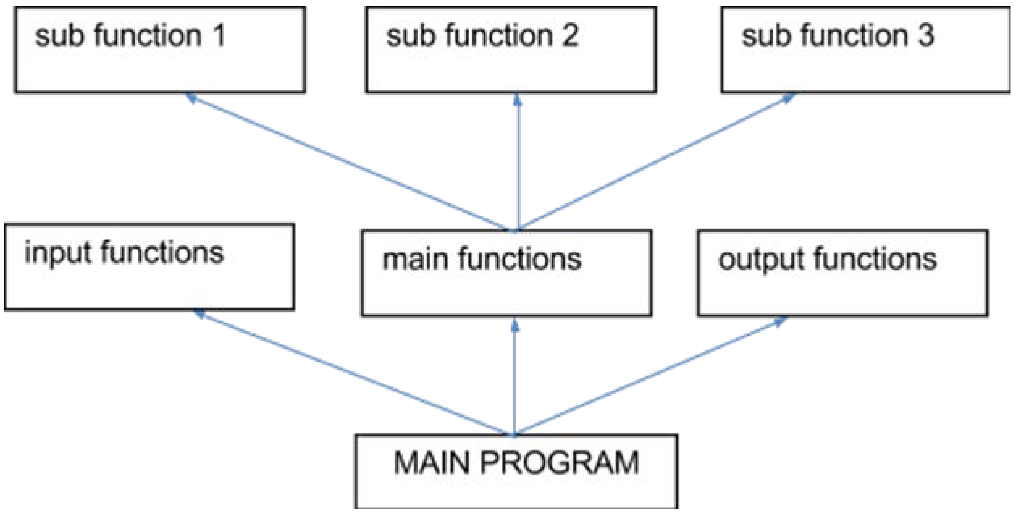


Figure 2.3 Bottom-up design techniques

Read programming and problem solving through c language by Harsha Priya, R. Ranjeet, also, Programming and PROB solving using C by ISRD. Understand bottom-up technique, its advantages and disadvantages.

Overview of programming design techniques

Linear Programming. This entails programming in a sequential manner. Decision making and iteration constructs are not included. It is a direct translation of a sequential algorithm.

Structured programming: This programming design technique uses three major constructs of programming; sequential, loops and conditional structures. It also embraces the use of top-down problem solving technique (explained above). Therefore, programs written using this approach can be easily understood, corrected (debugged), reusable e.t.c. Read chapter three-page 17-21 of Programming and PROB solving using C by ISRD. Understand the advantages and elements of structured programming.

Modular design of programs. Top-down techniques leads to the concept of modularization. Programs can be logically separated into; initialization, input, input data validation, processing, output, error handling and closing procedure. Read chapter three-page 21 of Programming and PROB solving using C by ISRD and understand basic attributes of a module, control relationships between modules, communication between modules and module design requirements.

Activity Details

Thus you are required to read the topics on design techniques which include top down and bottom up design techniques; their advantages and disadvantages and where each technique is more applicable. Also, read on algorithms; characteristics of a good algorithm, algorithm logic (sequential, conditional and repetitive flow), how to test an algorithm- desk checking and walkthrough, difference between an algorithm and pseudo code, advantages of using pseudo code over flowcharts and vise versa and flowchart construction; including advantages and disadvantages of using flowcharts, rules for designing a flow chart. Programming techniques which include, linear programming, structured programming (advantages and constructs of structured programming), and modular design of programs. The readings can be obtained from chapter one, two and three of Programming and PROB solving using C by ISRD and chapter 1, page 1-13, chapter two, page 20-35 of Priya and Ranjeet (2006). Programming and problem Solving Through "C" Language textbook published by Firewall Media.

books provided in the reference section for this unit and from related and relevant information in the Internet. Thereafter write brief notes;

- Describing top down and bottom up design techniques.
- Describing the software development process.
- Describing programming techniques

Hands-on activity

1. Using pencil and paper or a computer, constructs a flowchart for one of the two problems that you listed in activity 1.
2. Write an algorithm for the flowchart you constructed above (1)
3. Test your algorithm using desk check technique.

Conclusion

This activity equips the learner with the solution design techniques. The learner also acquires hands on experience on how to design solutions using flowcharts and algorithms/pseudo codes.

Assessment

1. What do you understand by the word program design?
2. Name and describe two problem solving techniques.
3. Describe the difference between the two problem solving techniques mentioned above (2)
4. Algorithms can be presented by use of pseudo code or a flowchart. What is the difference between an algorithm and a pseudo code?
5. Pseudo code is language independent. What does it mean?
6. What is the significance of documentation in problem solving process?
7. Algorithms that are developed using three basic constructs are easier to follow. Describe these constructs.
8. Describe the characteristics of a good algorithm
9. What are the advantages of flowcharts over algorithms
10. Name the rules for constructing a flowchart
11. The design of programs defines the structure of program code. Describe three programming techniques
12. What are some of the advantages of structured programming
13. Describe three elements of structured programming
14. An algorithm can be tested using desk checking and walk through approach. Describe these testing approaches
15. Explain the term modular design of programs

Group work study activity

Introduction

This activity involves groups or collaboration to research on some aspects of programming. The activity requires the learner to find out from practicing programmers the kind of problems they solve on daily basis, the strategies they apply and the steps followed to achieve a solution and common operations in all the programs. They are required to know the practicality of various problem solving techniques and their significance.

Activity Details

In this activity you are required to;

Inquire from a developer/s the kind of problems they solve on daily basis. Use the examples gathered to help you think of a much larger program (and not too big) and by applying the concepts learnt in this unit,

- Specify or formulate the problem and define it by showing the persons involved, inputs, operations and the output.
- Design the solution to the problem using (1). top down technique and (2). bottom up technique.
- Use flowcharts and pseudo code/algorithm to represent the solution.
- Use desk checking technique to test your pseudo code/algorithm.
- Collaborate with another group, swap or exchange your designed solutions (algorithms) then perform a peer checking (walkthrough).

Conclusion

Through this activity, the learner will be able to gain more knowledge on computer based problem solving. The first step of problem solving includes problem definition which entails understanding the nature of the problem at hand. This level of problem solving does not address 'how' to solve the problem, but tells us 'what' the problem really is. By collaborating with others, the learner will be able to enhance skills on the applications of problem solving concepts.

Assessment

1. Name operations that are common to all solutions
2. Apart from input, operations and output, what other features do you consider when defining a problem?
3. describe one property of a well stated problem
4. What are the advantages of top down technique over bottom up technique
5. Describe the results you obtained from desk checking
6. What is the significance of testing an algorithm?
7. What benefits did you achieve by collaborating with your teammates and members of another team?

Unit Summary

This unit discussed computer problem solving strategies, steps followed when developing a program, Design techniques which include top down and bottom up, algorithms and two ways of presenting an algorithm; pseudo code and a flow chart. Program design involves how to solve the problem and is a problem solving phase that comes after problem definition. It involves providing problem solutions by applying various design techniques including modeling. These techniques include breaking down of program into subprograms. This aspect of dividing a program into subprograms leads to modularization. Modularization on the other hand, is one of the basic elements of structured programming. The unit also looked at ways of testing an algorithm; desk checking and walkthrough (peer checking), and an overview of programming techniques.

Unit Assessment

Check your understanding!

Formative assessment

1. A flowchart is______________________________________ (2marks)

2. We need computers to solve problems because of; (2marks)

   a. speed in performing computation tasks

   b. less error-prone

   c. reduction in paperwork

   d. versatility of operations performed

   e. all of the above

   f. alb and c

3. Consider a system used for controlling nuclear weapon. Exposure of human beings to this hazardous environment is not advisable. Moreover the accuracy of the operation involved is mandatory as the system is very critical in nature. What are the advantages of a computer based system in such a scenario. (2marks)

   a. accuracy of operation

   b. safety of the human kind

   c. availability of the system

   d. none of the above

   e. all of the above

4. Problem definition involves (2marks)

   a. Identifying key persons involved

   b. identifying input

   c. specifying the output of the solution

d. specifying the operations of the program

e. b, c, and d

f. all of the above

5. Simplicity, clarity and elegance are the hallmarks of good programs, but various implementation issues must be taken into consideration when designing a solution. Which ones; (2marks)

a. minimize memory requirements

b. maximize output readability

c. maximize source text readability

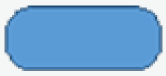
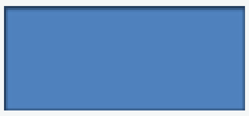
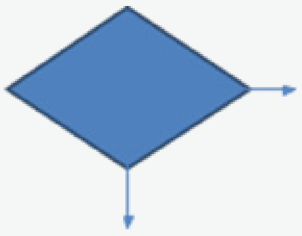
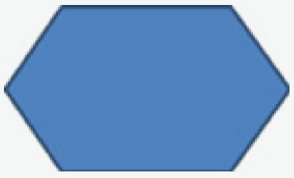
d. minimize number of source code statements

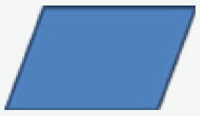
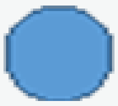
e. minimize development time

f. a and c

g. all

6. Match the following flow chart notations with their correct functions and names. (6marks)

	Notation	**Function**	**Name**
a		Connecting Two Steps	Processing
b.		Represents A Condition	Looping
c.		Taking In/Outputting Values	Connector
d.		Performing Calculations Processing	Input/Output And

e.		Loop a set of steps till	decision making condition is satisfied
f.		Indicating start/end of a process	begin/end

Instructions

The aim of this assessment is to check learner progress by determining how much the learner has learned in this unit. Questions cover everything that was presented in this unit in order to assess overall understanding. Answer them carefully and if your score falls

- below 40%, redo the readings.
- between 40% and 60%, redo the readings on your weak area
- above 60%, you have substantial amount of knowledge

Grading Scheme

The unit assessment (formative assessment) has been assigned marks. The instructor will assign marks to activity assessment then compute all. At the end of unit four all the assessments must be computed to 10% of the total examination marks.

Answers

Formative assessment Answers

1. flowchart is a pictorial representation of an algorithm

2. e

3. e

4. f

5. g

6.

a. indicate start and end of a process	begin/end
b. performing calculations and processing	processing
c. represents a condition	decision making
d. loop a set of steps till condition is satisfied	looping
e. taking in/outputting values	input/output
f. connecting two steps	connector

Unit Readings and Other Resources

- Priya, H., & Ranjeet, R. (2006). Programming and problem Solving Through "C" Language. Firewall Media.
- ISRD, (2008). Programming & Problem solving using C. Tata McGraw-Hill Education
- Gary Marrer, (2009). Fundaments of Programming: with Object Oriented Programming. Gary Marrel.
- Joyce Farrell, (2012). Programming Logic and design, Comprehensive. 7th edition. Engage Learning.
- Baviskar, D. P. (2009. Fundamentals of programming languages. Technical Publication Pune.

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

Unit 3. Programming in C Language.

Unit Introduction

Programming details may look different in different languages, but a few basic instructions appear in just about every language. These instructions include input instructions that are used to get data from the keyboard, a file, or some other device, output instructions used to display information on the screen or send information to a file or other device, arithmetic instructions to perform basic arithmetical operations like addition and multiplication, conditional execution instructions are used to check for certain conditions and execute the appropriate sequence of statements and repetition instructions perform some action repeatedly, usually with some variation. Therefore, this unit discusses programming constructs like variable, data types, operators, control structures, arrays and subroutines or functions. Subroutines or functions will reinforce the concept of breaking down a problem into smaller problems (top down design and/or bottom up design techniques). Simple programs will be written using C language to reinforce all the concepts learned in the previous unit.

Unit Objectives

Upon completion of this unit you should be able to:

- write programs using data types, variable and operators
- develop solutions with control structures and functions
- apply arrays and strings to solutions in C programming language.

Key Terms

Data type: Data types define the way in which values and range of values are represented in a system

Variables: variables are the names given to the stored region of memory. The value of a variable changes during the execution of the program

Variable declaration: Variable declaration states the type of the variable and allocates memory for the variable

Identifier: Identifier is a variable name

String constant: it is a set of characters enclosed in double quotation marks (fundamentals)

A constant: Is a value that does not change during execution of a program (fundamental)

An operator: It is a symbol which helps the user to command the computer to do a certain mathematical computation or logical operation. (fundamental)

Console application: A console application is a computer program designed to be used via a text-only computer interface, such as a text terminal, the command line interface of some operating systems (Unix, DOS, etc.) or the text-based interface included with most Graphical User Interface (GUI) operating systems, such as the Win32 console in Microsoft Windows, the Terminal in Mac OS X, and xterm in Unix (http://en.wikipedia.org/wiki/Console_application).

Integrated Development Environment (IDE): An integrated development environment (IDE) or interactive development environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger (http://en.wikipedia.org/wiki/Integrated_development_environment).

Derived data types: Derived data types are those that are defined in terms of other data types, called base types. Derived types may have attributes, and may have element or mixed content (http://msdn.microsoft.com)...

User defined data types: Data types that are defined by the user based on existing data types.

Source code editor: A source code editor is a text editor program designed specifically for editing source code of computer programs by programmers. It may be a standalone application or it may be built into an integrated development environment (IDE) or web browser. Source code editors are the most fundamental programming tool, as the fundamental job of programmers is to write and edit source code (http://en.wikipedia.org/wiki/Source_code_editor).

Header files: Header files contain definitions of functions and variables which can be incorporated into any C program by using the pre-processor #include statement. Standard header files are provided with each compiler, and cover a range of areas, string handling, mathematical, data conversion, printing and reading of variables (http://gd.tuwien.ac.at/languages/c/programming-bbrown/c_011.htm).

A comment: A "comment" is a sequence of characters beginning with a forward slash/asterisk combination (/*) and ends with asterisk/slash (*/) that is treated as a single white-space character by the compiler and is otherwise ignored. A comment can include any combination of characters from the represent able character set, including newline characters, but excluding the "end comment" delimiter (*/). Single-line comments is preceded by two forward slashes (//) (http://msdn.microsoft.com/en-us/library/wfwda74e.aspx).

Primitive data types: Primitive data types are those that are not defined in terms of other data types. Because primitive data types are the basis for all other types, they cannot have element content or attributes (http://msdn.microsoft.com).

Type cast: Typecasting concept in C language is used to modify a variable from one data type to another data type. New data type should be mentioned before the variable name or value in brackets which is to be typecast (http://fresh2refresh.com/c/c-type-casting/).

Formal parameters: These are parameters that act as placeholders for the actual parameters.

Actual Parameters: These are the parameters (or arguments or values) passed to the function from the calling function (environment).

Local variables: Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own (http://www.tutorialspoint.com/cprogramming/c_scope_rules.htm).

Global variables: Global variables are defined outside of a function, usually on top of the program. The global variables will hold their value throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program. A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration (http://www.tutorialspoint.com/cprogramming/c_scope_rules.htm)

An expression: an expression is a combination of variables, constants, and operators according to the syntax of the language (Book by ISRD)

Primitive data types: Primitive data types are those that are not defined in terms of other data types. Because primitive data types are the basis for all other types, they cannot have element content or attributes (http://msdn.microsoft.com).

Type cast: Typecasting concept in C language is used to modify a variable from one data type to another data type. New data type should be mentioned before the variable name or value in brackets which is to be typecast (http://fresh2refresh.com/c/c-type-casting/).

Formal parameters: These are parameters that act as placeholders for the actual parameters.

Actual Parameters: These are the parameters (or arguments or values) passed to the function from the calling function (environment).

Local variables: Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own (http://www.tutorialspoint.com/cprogramming/c_scope_rules.htm).

Global variables: Global variables are defined outside of a function, usually on top of the program. The global variables will hold their value throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program. A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration (http://www.tutorialspoint.com/cprogramming/c_scope_rules.htm)

An expression: an expression is a combination of variables, constants, and operators according to the syntax of the language (Book by ISRD)

Source code: It is a computer program written in a high-level language which is converted into object code or machine code by a compiler. Source code is the only stage where a programmer can read and modify a computer program. Read more: (http://www.businessdictionary.com/definition/source-code.html#ixzz3LassqJW2).

Library: A library is a collection of (usually) precompiled, reusable programming routines that a programmer can "call" when writing code so that the programmer doesn't have to write it (searchsqlserver.techtarget.com/definition/library).A library in C is a group of functions and declarations, exposed for use by other programs. The library therefore consists of an interface expressed in an .h file (named the "header") and an implementation expressed in a .c file. This .c file might be precompiled or otherwise inaccessible, or it might be available to the programmer. (Note: Libraries may call functions in other libraries such as the Standard C or math libraries to do various tasks) (http://en.wikibooks.org/wiki/C_Programming/Libraries).

Scope of rules: A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable cannot be accessed. There are three places where variables can be declared in C programming language: Inside a function or a block which is called local variables, Outside of all functions which is called global variables and In the definition of function parameters which is called formal parameters (http://www.tutorialspoint.com/cprogramming/c_scope_rules.htm).

Learning Activities

Data types, variable and operators

Introduction

For us to discuss and illustrate these concepts using C program, first we must start by familiarizing ourselves with the language's environment (overview of the C language). There are good reference materials that give a good start to C programming including the references provided for this unit -programming and problem solving through 'C' languages by Priya and Ranjeet, 2006 and Programming and PROB solving using C. by ISRD. Many C compilers exist in the market and were designed or developed by different groups of program developers. Therefore, the preference of a compiler is left to you (learner). This course's preference is Dev C++ compiler; if you can download it, the better (recommended) from https://sourceforge.net/projects/orwelldevcpp (Dev-cpp 5.11 TDM-Gcc 4.9.2 setup.exe) or the latest, updated version that can run on latest version of windows. Request your instructor to help you if you face any challenges when downloading (retrieving), or installing the compiler. The compiler reads C++ and this should not be a worry to you because you will learn how to compile a program in C by renaming files to have C extension (.c). After installation, start the program (compiler). It should appear as follows, see figure 3.1.

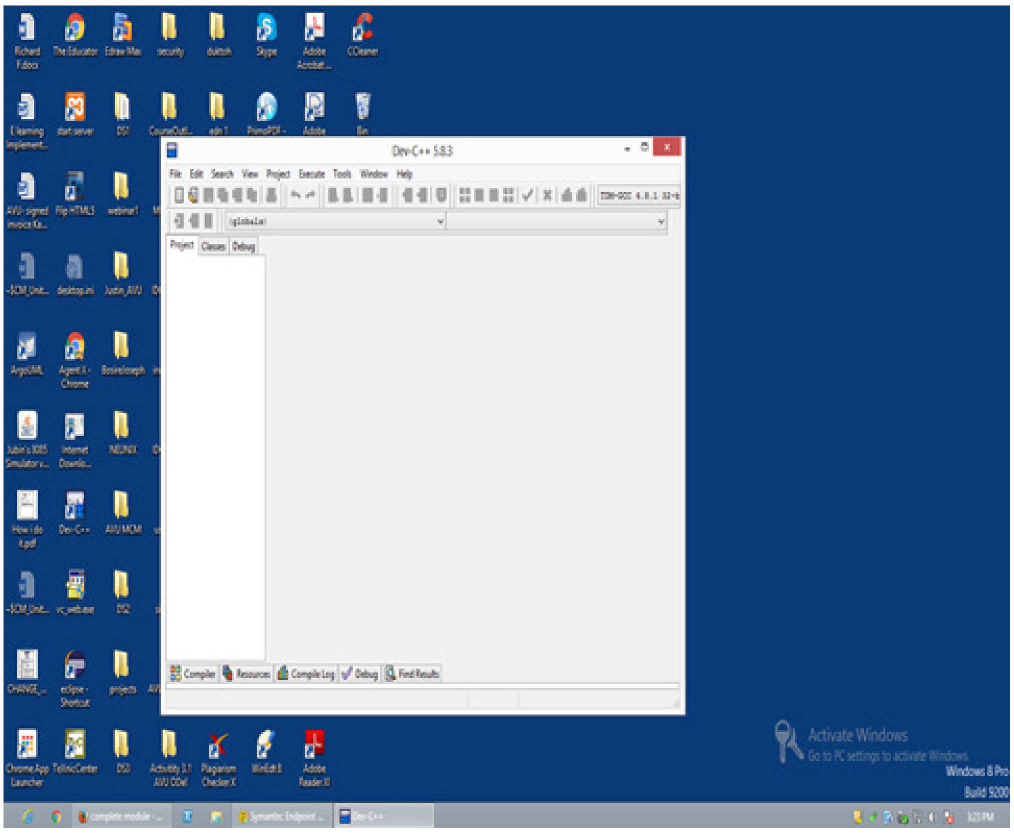
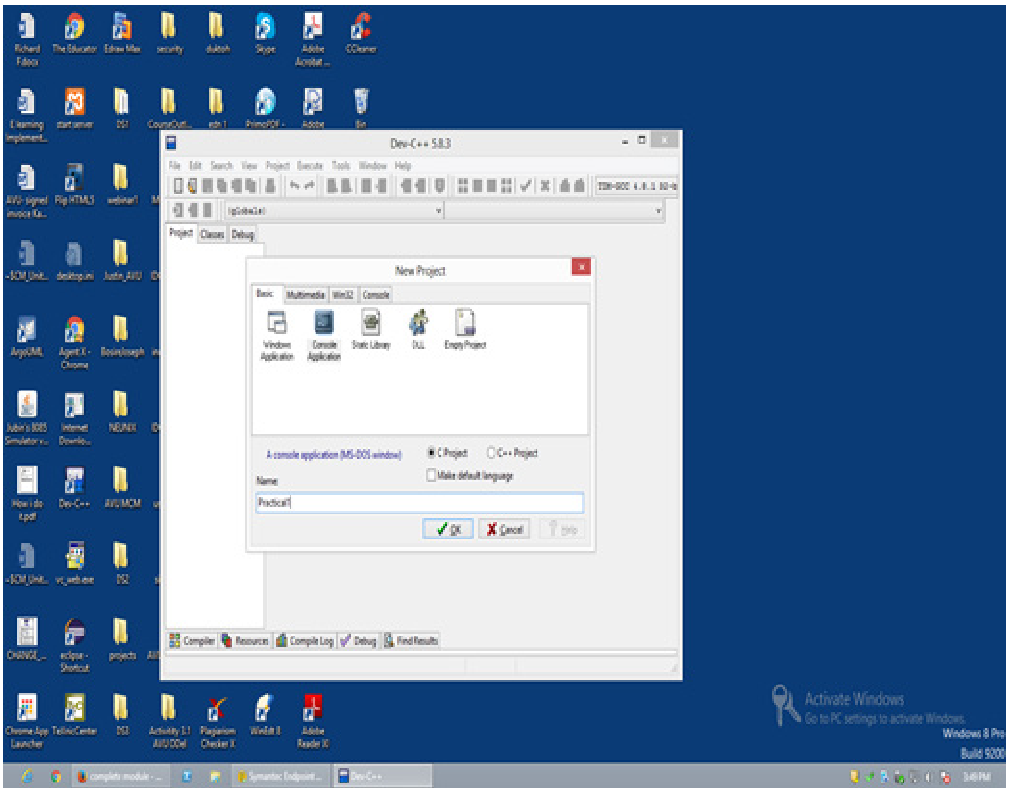

Dev-C++ 5.8.3
File Edit Search View Project Execute Tools Window Help
TDM-GCC 4.8.1 32-bit
(globals)
Project Classes Debug
Compiler Resources Compile Log Debug Find Results
Activate Windows
Go to PC settings to activate Windows.
Windows 8 Pro
Build 9200

Figure 3.1 DEV C++ console window

The next activity is to create a new source file from the FILE menu then save it with a dot c extension (.c) or from the FILE menu, create a new project, select console application then C project and name your project (for example as practical1). See figure 3.2. Click OK button.


Dev-C++ 5.8.3
File Edit Search View Project Execute Tools Window Help
Project Classes Debug
New Project
Basic Multimedia Win32 Console
Windows Application
Console Application
Static Library
DLL
Empty Project
OK
Cancel
Help
Compiler Resources Compile Log Debug Find Results
Activate Windows
Go to PC settings to activate Windows.
Windows 8 Pro
Build 9200

A console application (MS-DOS window)

[✓ C Project] C++ Project

Make default language

Name: Practical1

Figure 3.2 sample C new project

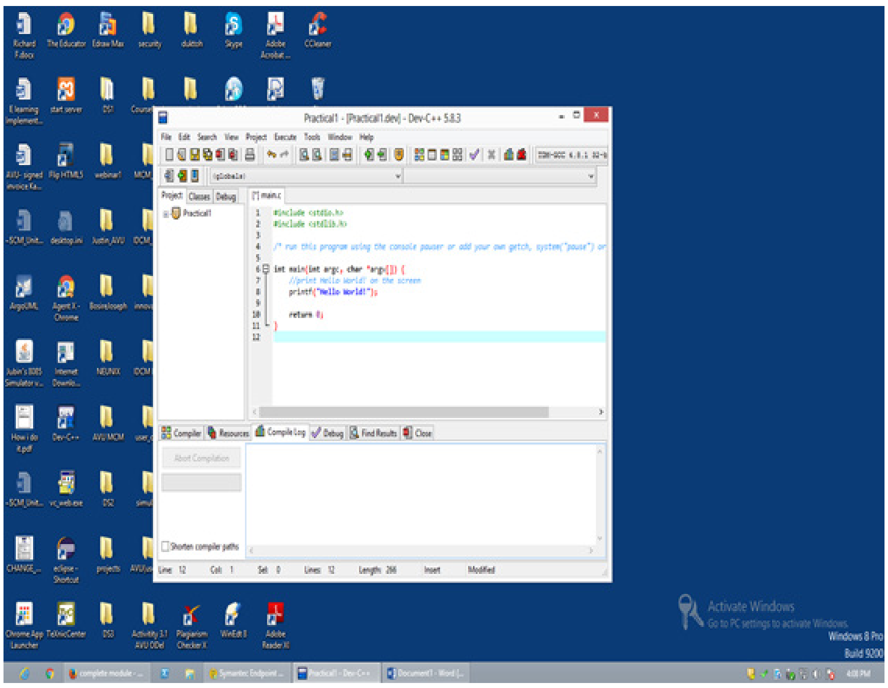
Now, select the folder to store the files in the next window. And after indicating the folder where the project configuration file (.dev) will be saved, the IDE generates a basis source code file (by default, main c). The IDE window includes three sub-windows: the Project Files Explorer, the Result Tabs, and the Source Code Editor. These windows can be resized and minimized.

Figure 3.3 IDE window subsections

The Files Explorer window shows the name of the project and the included files. The Project tab usually contains a single file with the source code of the program. In this pane, we can find two additional tabs: Classes and Debug. Classes tab shows the functions of the program.

Debug tab shows watched variables in the debugging process. The Results window is used to present the results of the actions of the IDE: compilation errors, compiling directives, debugging commands. The Source code editor shows the code of the program. Once the project has been created, we can start writing then execute our C program. For example the simple hello world program, see figure 3.4 below.

Figure 3.4



```
#include <stdio.h>
#include <stdlib.h>

/* run this program using the console pauser or add your own getch, system("pause") or

int main(int argc, char *argv[]) {
    //print Hello World! on the screen
    printf("Hello World!");

    return 0;
}
```

Practical1 - [Practical1.dev] - Dev-C++ 5.8.3
File Edit Search View Project Execute Tools Window Help
Project Classes Debug
main.c
Practical1
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Line: 12 Col: 1 Sel: 0 Lines: 12 Length: 266 Insert Modified
Activate Windows
Windows 8 Pro

Figure 3.5 A sample of a C program

The process of executing a program in C involves the following steps:

1. Creating the program
2. Compiling the program
3. Linking the program with functions that are needed from the C library
4. Executing the program

Note:

The above content was borrowed from http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-013/IntroductiontoDevCIDE.pdf. Therefore we recommend you to visit it for further reading. It introduces Dev C++ in a simple, clear manner (step by step) that will guide you to easily grasp and get started. Do not worry yourself so much with debugging part as we shall introduce it in the next unit.

A summary of these steps can be represented on a chart as shown below;

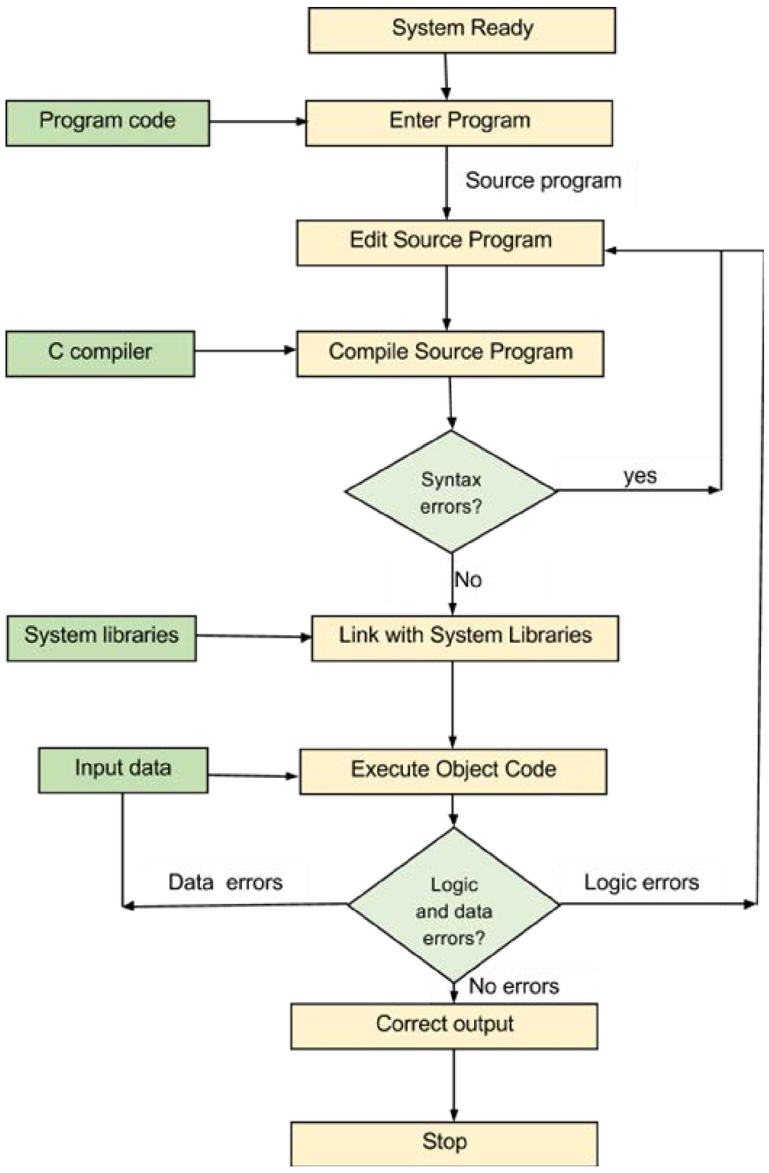


Figure 3.6 Process of executing C program source: ISRD

Creating the program: The program that is to be created must be entered into a file. The file name can consist of letters, digits and special characters, followed by the extension .C. Example of a valid file names are;

Hello.c

Pract1.c

Compiling and linking: During compilation process the source program instructions are translated into a form that is suitable for execution by the computer. The translation process checks each and every instruction for correctness and if no errors are reported then it generates the object code.

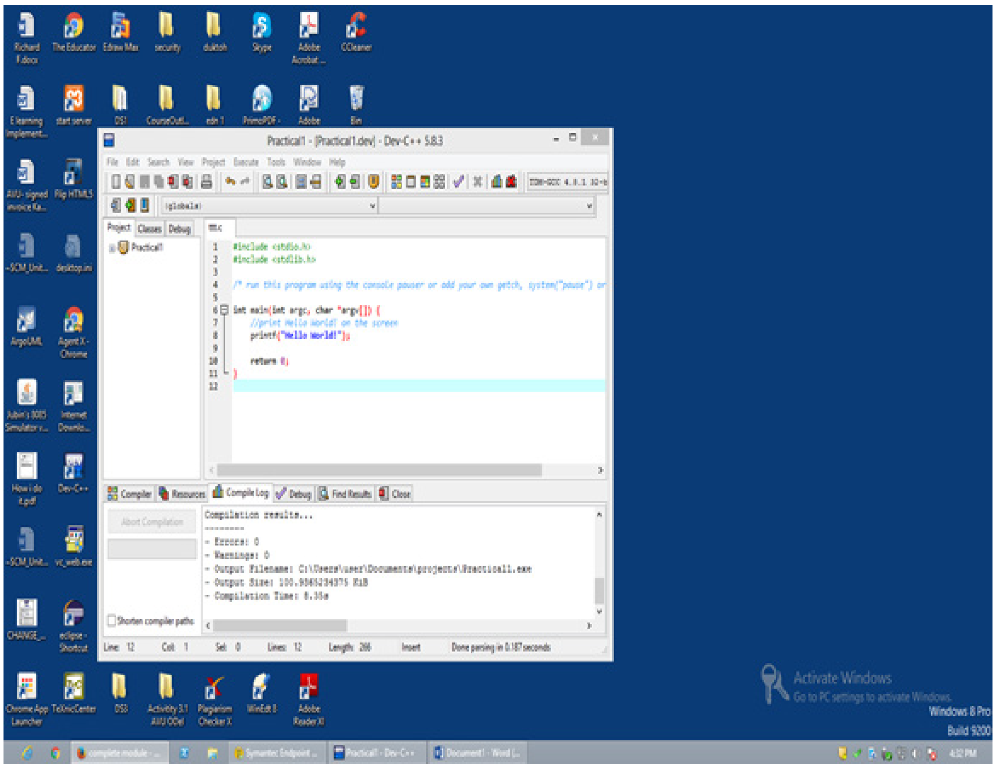
During the linking stage the other program files and functions that are required by the program are put together with it if mistakes in the syntax and semantics of the language are discovered, they are listed out and the compilation process ends there. The errors should be corrected in the source program with the help of an editor and the compilation is done again.

Executing the program: During execution, the executable object code in the computer's memory is loaded and the instructions are then executed. During execution, the program may request for some data through the keyboard.

An example describing basic parts of a C program using Dev C++;

Figure 3.7 Parts of a C program

See figure 3.8 for simple C program developed using Dev C++ version 5.8.3 (TDM-GCC 4.8.1). The program is executed by compiling then running it from EXECUTE menu.



Practical1 - [Practical1.dev] - Dev-C++ 5.8.3
File Edit Search View Project Execute Tools Window Help
TDM-GCC 4.8.1 32-bit
(globals)
Project Classes Debug
Practical1
main.c
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Activate Windows
Go to PC settings to activate Windows.
Windows 8 Pro
Build 9200

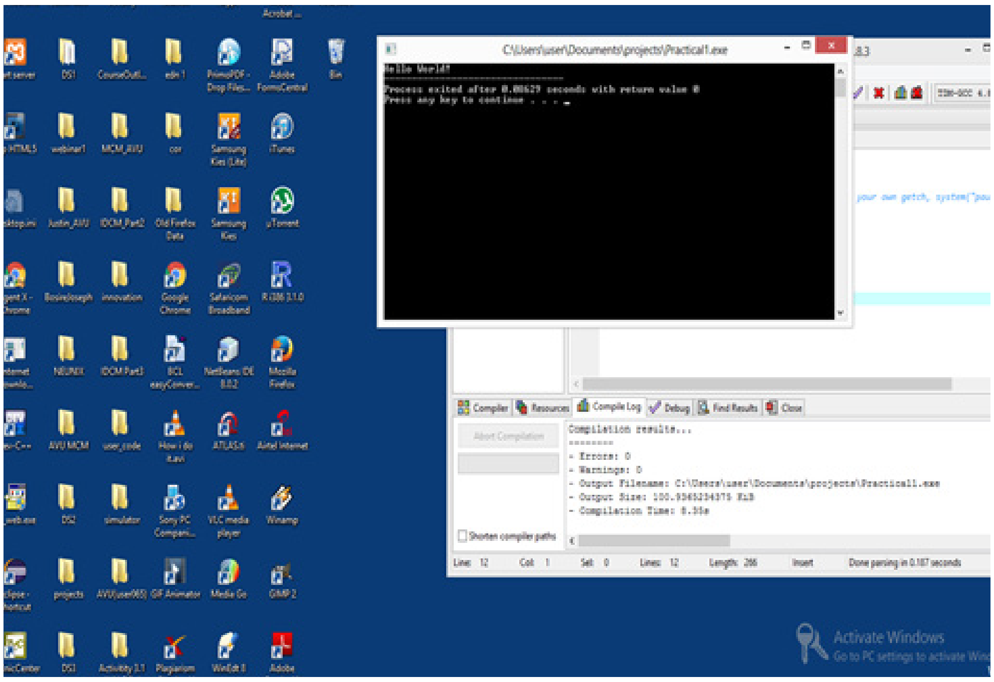
```
#include <stdio.h>
#include <stdlib.h>

/* run this program using the console pauser or add your own getch, system("pause") or

int main(int argc, char *argv[]) {
    //print Hello World! on the screen
    printf("Hello World!");

    return 0;
}
```

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\user\Documents\projects\Practical1.exe
- Output Size: 100.9365234375 KiB
- Compilation Time: 0.35s
```

```
Line: 12    Col: 1    Sel: 0    Lines: 12    Length: 266    Insert    Done parsing in 0.187 seconds
```

Figure 3.8 Compiled C sample program

When you select run from EXECUTE menu, you obtain the following output, see figure 3.9 (dark screen or DOS like screen):



C:\Users\user\Documents\projects\Practical1.exe
Activate Windows
Go to PC settings to activate Windows

```
Hello World!
--------------------------------
Process exited after 0.00629 seconds with return value 0
Press any key to continue . . .
```

Figure 3.9 Output of compiled C sample program

print () and scan () functions

Print () function is used to write information to a file, screen or any other output device. While scanf() function is used to read in data that the program manipulates or writes on the screen. These functions are supported by the header file or preprocessor directive <stdio.h>, meaning standard input and output header files. For example a program that prints a value that is already assigned to a variable and a value (age) read to the system by the user. The program prompts the user to key in age;

Remember to use formatting specifiers when reading and writing values. Also, use address '&' operator when reading in values. Apply escape characters where applicable to make your code more readable. You are required to read the textbook by ISDR (Programming and problem solving using C) page 48 on character constants to learn more on escape sequence. Also, read Priya and Ranjeet (2006) (Programming and problem solving through "C" language) page 82-83-section 3.7.1 on I/O functions to know more about scanf () function including the use of the address operator '&'.

Data types

A program stores or accepts different kinds of data including; integers, characters, and real values with the following keywords; int, char and float respectively. All these are defined as data types. Different data types consume different sizes of the memory. For example a character occupies one byte of the memory space, integers occupy four bytes e.t.c. as shown in the following table but memory requirements may vary from compiler to compiler. Therefore, it important to choose the correct data type to avoid wastage of the computer's vital resource (memory)-See table 3.1.

Table 3.1 Data types with their memory requirements and properties

Data type	Memory requirements	Properties of data values
Void	0 byte	Nothing, a non existing object
Char	1 byte	Holds character variables like, 'a', '&', etc
Int	2 bytes	Integral values, e.g. 12, 15
Float	4 bytes	Floating point values; single precision value e.g. 13.7, 78.0
Double	8 bytes	Floating point values; double precision e.g. 17E+21

Apart from these data types, C supports other data types known as secondary data types. The figure below gives a summary of the data types;

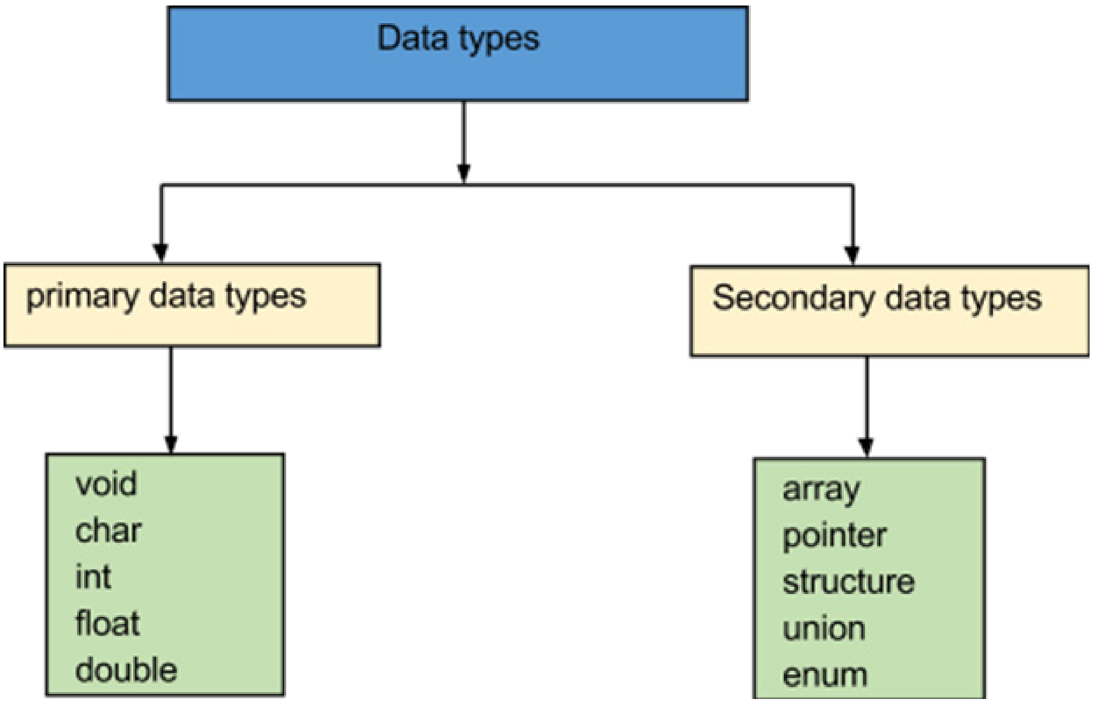


Figure 3.10 Categories of data types

The primary data types are also known as primitive data types. Secondary data types can be further divided into user-defined data types and derived data types. For example, structure, union and enum are user defined data types while arrays and pointers are derived data types.

Variables

Data is stored in memory location called a variable. This means a programmer names a location in the memory and specifies the amount of space required for storing data by defining the data type. The name of the memory location is known as an identifier. This name gives a user-friendly access to memory location. For example; consider a situation where the programmer needs to store the details of a student which include, age, gender and fee. Age is a whole number with no fraction. Therefore, int is an appropriate data type for age. Fee can have fractional values thus float variable is appropriate for fee. Gender can be represented as single character; 'F' or 'M' meaning char variable is appropriate. Thus, appropriate identifiers and memory requirements include the following declarations;

```
int age;

float fee;

char gender;
```

Each language has rules for inventing variable names or identifiers. C follows the following rules;

1. C is a case sensitive language thus ‘Age’ and ‘age’ are different
2. Variable names should not start with a number e.g 2014fee not allowed
3. Special characters like ‘%’, ‘@’ e.t.c except an underscore ‘_’ are not allowed in a variable name. e.g. –fee Balance not allowed but fee Balance is allowed
4. variable should start with an alphabet or the special character ‘_’. (just as a convention, it is not advisable to use variable names starting with underscore ‘_’ as library functions follow this convention)

<u>Constants</u>

C supports a number of constants which include;

1. Integer constants, for example 234, -78 e.t.c, consist of a set of digits 0 to 9 and can be represented assigned digits using + (positive) or – (negative) values
2. Real constants, for example -0.98, 10.78 e.t.c, consist of fraction part
3. Character constants, for example ‘v’, ‘j’, ‘y’, ‘n’ etc, are enclosed within single quotes (‘.’)
4. String constants, for example “Noela”, “Student”, “home” etc, are enclosed within double quotation marks (“...”).

Constants can be defined using an assignment operator (=), or a const keyword or # define directive.

Other character constants (escape sequence)

There are many other non-printable characters and C represents these characters using an escape sequence (\). For example ‘\n’ represents new line. It advances information to a new line.

Operators

Operators can directly operate on data or variables. There are two major types of operators which include; unary operators and binary operators. Unary operators operate on only one (or single) operand, for example +67, -5, ++x, --y, e.t.c. While binary operators operate on two or more operands, for example 2+3 (+ is an addition symbol and operates on two operands (values)), d/x (/ is division and operates on two operands) producing a different value. Operators can also be classified into various categories namely;

Arithmetic operators

OPERATOR	MEANING
+	Addition
-	Subtraction
/	Division
*	Multiplication
%	modulo

Relational operators

OPERATOR	MEANING
==	Equality
!=	Not equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

logical operators

OPERATOR	MEANING
&&	AND
||	OR
!	Not

Assignment operators

Represented using an '=' operator for example x=6;

Increment and decrement operators

OPERATOR	MEANING
++	Increment
--	Decrement

Conditional operators

It is a ternary operator that contains three operands and its syntax looks like this;

exp1? exp2: exp3;

For example we want to write a program that returns the value of an if the value in c is less than five (5). Otherwise display or return the value in b.

(c<5)? a: b

Comma operator

It is represented using ',' operator and is used to separate expressions.

Size of operator

It is used to return the number of bytes the operand is occupying in the memory. for example;

```
char y;

y=size of (char);

print ("&d", y);
```

Output is: y= 1

Bitwise operators

OPERATOR	MEANING
&	Bitwise AND
|	Bitwise inclusive OR
^	Bitwise exclusive OR
<<	Left shift
>>	Right shift
~	One's complement

Compound operators

These are operator that is used to manipulate the contents of a variable then assign results to the same variable. They include;

OPERATOR	MEANING
+=	Add and assign to
-=	Subtract and assign to
/=	Divide and assign to
*=	Multiply and assign to
%=	Find remainder and assign to

For example;

a+=b; which is the same as a = a+b. Meaning, add the value in a and b and store results in a.

Expressions

Variables must be assigned values before evaluation of expressions. For example;

variable=expression;

R= ((a+b/d) * (h*b));

Type casting

Forcing a data of a particular type to be returned as another type. The syntax is;

(desired-type) exp;

For example 8/7 will output 1 but when forced to return a real value as;

```
int x;

x= (float) 8/7;
```

outputs: 1.14

Activity Details

You are required to read more on printf(), scanf(), variable, constants, escape sequence and operators. The activity also involves practical. After reading, you will be required to write simple programs that will help you understand the principle concepts of programming. You are advised to use Fundamentals of programming languages by Dipali P. Baviskar and Programming and PROB solving using C. by ISRD. These books will guide you in learning and understanding variables, constants and operators. you can also search for relevant material on these topics from the internet. After reading you are required to:

- Make brief notes on data types, variable, different types of constants and operators.

Hands-on activity

1. Download and install C compiler in your computer (if you do not have one yet). You can ask for assistance on this activity.
2. Familiarize yourself with C language environment by writing a simple program that outputs your name
3. Learn how to define and declare identifiers.

a. declare a variable for storing your middle-name initial

b. declare a variable for storing your age

c. declare a variable for storing your height

d. declare a constant that holds interest (0.12)

e. declare a variable that holds principal amount (real number)

4. Write simple programs that help you learn how to use variables, assign values to variables, use operators and output results.
5. Using assignment operator, initialize variables i through v and print the values
6. Write a single statement that;

   - reads in an integer x
   - reads in a double y
   - reads in a character i
   - prints integer x
   - prints double y
   - prints character i
   - increments x by 1

7. Using appropriate data types, identifiers/constants, and operators, write a C program that reads in two values, adds the values, gives the product of two values, divides sum of the two values by 3, Increments the value obtained by adding two values (using prefix method), decrements the value obtained from the product of the two values by 2 then outputs the results on the screen.

8. Write a program that implements the following code. Show the output

```
int a, b, c, d=5;

 a = ++d;

 b = a++;

 c = b--;

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

Conclusion

This activity has taught you how to run a program using Dev C++. You have also known how to define and declare variables using various data types and appropriately apply all the operators mentioned in this unit.

Assessment

1. Define the following terms;
   - i. a data type
   - ii. A variable
   - iii. an operator
   - iv. an identifier
   - v. a program keywords
   - vi. derived data type
   - vii. header file
2. Name the rules that are followed when coming up with an identifier
3. Why is it important to understand how to use data types?
4. Various programming languages C included support a number of operators. List these operators; give examples, their associativity and how they are applied.
5. What is a ternary operator? Show its syntax.
6. List any five primary data types and any four secondary data types
7. What is the difference between postfix increment operator and a prefix increment operator?
8. Discuss the four types of constants.
9. Describe three ways of defining constants. Using C, write a program to illustrate how constants are declared with the use of the three methods.

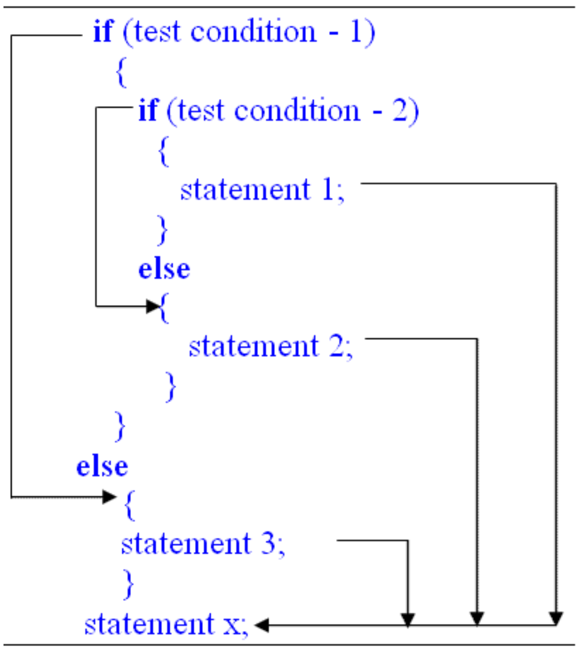
Control structures and functions

Introduction

A program is not limited only to sequential flow of control because at some point, a program may decides to branch or use a particular path of instructions or may repeat a statement a given number of times as long as condition is true. Therefore, other styles of handling or representing flow of control in programming include; branching and looping. Branching structures are also known as conditional or decisional structures. Looping structures repeat a command while condition is fulfilled/true. These two control structures are part of structured programming (discussed in unit 2. remember!) elements which make a program more readable and understandable.

Branching control structures

When dealing with branching or conditional structures also known as selection statements, the programmer is required to specify a condition that needs to be evaluated by the program, together with the statements to be executed by the program. If the condition is evaluated to true the if statement or block of statements are executed. Another selection structure is if/else statement (see figure 3.11A). If/else statement evaluates a condition and if it is true it executes the if block of statements but if the condition is evaluated to false it executes another block of statements (See figure 3.11 B).



```
if (test condition - 1)
 {
   if (test condition - 2)
    {
      statement 1;
    }
   else
    {
      statement 2;
    }
 }
else
 {
 statement 3;
 }
statement x;
```

Figure 3.11 if and if/else statement

Unlike figure 3.11A, figure 3.11 B tells what we want to happen if condition is false. In figure A, if condition is false, control is passed to the statement below if block statements without giving another option. There is also nested if statement where you can have if statement inside (executed by another if statement) another if statement. The syntax for the nested if statement looks as follows;

```
if (condition)

if (condition) {

Statements;

}
```

The if block statement/s will be executed only when both if statements are true. See figure 3.12

We can also have an if statement followed by else if statements. Its syntax;

Question: Design a flowchart for the above nested if and if/else statement. Study the syntax. Submit your solution to your instructor.

(We believe you have done it well. Now, let us proceed...)

Switch statement is a multi-decision control structure which allows values to be tested against a list of constant values until it finds a match then returns results. It is a cheaper way of representing nested if/else statements. It is more readable and easier to understand than nested if/else statement. Its syntax;

The? operator (conditional operator). This operator operates like the if/else statement. We mentioned it in first activity of this unit. Its syntax is as shown below;

exp 1 ? exp 2 : exp 3;

if exp 1 is true the exp 2 is executed but if exp 1 if false then exp 3 is executed.

Note: if statement is not terminated with (;) because it is executing the statement/s below it. Also, if the statement is executing more than one instruction, then it is important to present it as a block of code using the { }. For example;

```
if (condition)

{

statement;

statement;

}
```

Loops/repetition control structures

There are situations where an instructions needs to be executed repeatedly and a loop statement for the execution of a statement or statements a number of times as desired. The following loop statements can be used to repeatedly execute an instruction.

While statement: This statement repeats an instruction a number of times while the condition is true. If the condition evaluates to false, the while statement exits and the statement below it are executed (statements that are not within the while statement's parentheses). See figure 1.3. Its syntax is;

```
while (condition)

{

statement/s;

}
```

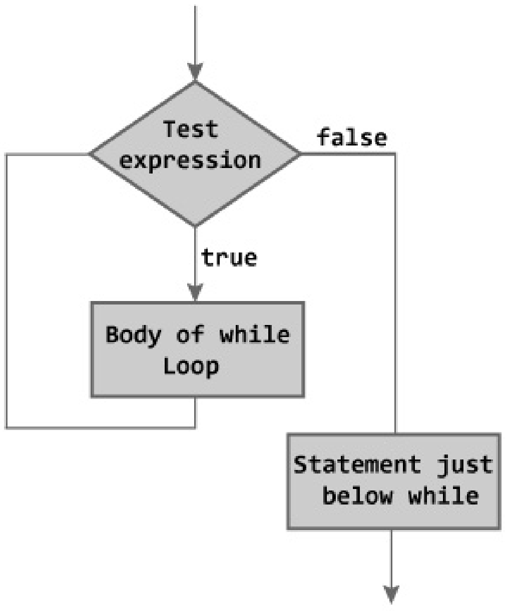


Figure 3.13 While statement flowchart

For example;

do/while statement: See figure 3.14. Its form looks like this;

```
do {

       statement/s;

}while(condition);
```

For example;

for statement: Just like the while statement, it repeats a loop a number of times. See figure 3.15. Its general form;

```
for (initializer; condition; increment)

        {

                statement/s;

        }
```

For example;

Note:

There is a small difference between while statement a do/while statement. The while statement first tests the condition before executing the statements. While the do/while statement executes the statements at least once before evaluating the condition – meaning it returns a value at least once whether the condition is true or false. Again, in do/while statement, the while statement is terminated; simply because it is not executing the statements below it.

Jump statements

There are two important jump statements in programming. These are; Break statement and continue statement. Break statement terminates a loop while continue statement exits the current loop then executes the next loop or iteration.

Functions

In the previous unit we discussed problem solving techniques which include top down and bottom up strategies. These techniques involve dividing a program into subprograms or subroutines and later all the subprograms are integrated to form one program. Do you still remember these strategies? If not, re-read unit 2. Anyway, these techniques lead to modular way of programming or creation of functions. Now, we can say that a function is a group of instructions that perform a particular task. In simple terms, all instructions in a specific group work together to achieve one goal.

Majority of the programming languages have at least one function known as main () function and this identifies where a program execution begins. Programmers are also able to invent or add more functions to their programs. Whenever a program encounters a function name, control is then passed to that particular function whose name was encountered. A function has this form;

return-type function-name (list of parameters) //function header

```
{
statement/s; //body of the function or function definition
}
```

Return type define the data type of the value that the function returns. Other functions do not return any value after executing an instruction. Such functions are defined with the return type using void keyword. Just like an identifier, function name is the actual name of the function that is used by other programs whenever they need services. A parameter is a placeholder for the values that are passed from the calling environment. Other functions may contain no parameters and the parameter list can be defined as void. That is, it is not receiving any values from other programs. The Body of the function contains statements that define what the function does.

For example a function that returns greeting to another function;

```
void    greeting (void)
        {
            printf("Good morning!");
        }
```

When another function needs to output this statement ("Good morning!") then the programmer includes the name of the function (greeting ()) in order to invoke or call that function. The function that invokes another function forms the calling environment. Now, see the program below;

Note:

We have defined our function (greeting ()) before our main function. This is because the compiler will not be able to see it if we placed it after the main function. To avoid restricting yourself to where to define a function, you can introduce a function prototype. A function prototype declares a function and tells the compiler that a particular function exists in the program. It gives the details of that function; return-type, name and parameters. For example;

Scope of rules

variables can be declared as local or global variables. Local variables are defined inside or within the block of code of a particular function. Such variables can be used only by statements that are inside that function or block of code simply because they are not known to functions outside them. Global variables are defined outside of all other functions in the program. As a convention, they are declared on top of the program after the preprocessor directive but before the main () function. They hold values throughout the lifetime of the program and they can be accessed by other functions that are defined for the program. Let us learn scope of variables using the following program;

Activity Details

Just like the first activity in this unit, you are required to read more on control structures and functions. This activity also involves practical. After reading, you will be required to write simple programs that will help you understand the principle concepts of programming. You are required to use Fundamentals of programming languages by Dipali P. Baviskar, Programming and PROB solving using C. by ISRD and Programming and problem solving through 'C' languages by Priya and Ranjeet. After reading the said topics, this activity then requires you to;

1. Write brief notes describing the difference between branching control structures and repetition structures.
2. Describe the significance of branching and looping structures
3. According to your own understanding, giving examples and probably illustrating, explain jump statements and their applications. Discuss the importance of learning and understanding how to use the jump statements.
4. Write brief notes on the advantages of using functions in a program

Hands-on activity (practical)

Using a software (preferably Microsoft word), draw a flowchart to perform the steps indicated in the statements 1 through X then gives the corresponding C statement. Assume they are properly declared

1. If age is equal to 20, increment age by 1. Output "next year you will be___" (output the incremented age on the blank line)
2. If grade is greater than 70, print the statement "excellent"
3. If grade is less or equal to 60, print the statement "Average", otherwise output "Get serious nyana!"
4. If height is greater or equal to 53.9 feet, decrement weight by 20.4 kg. Otherwise print "maintain weight at 55 kg".
5. While x is less or equal to 15, print the sum of values.

6. Convert statement 5 to a for statement
7. Convert statement 5 to a do/while statement
8. Write simple programs implementing the design (flowcharts 1 to 7); assume you are required to declare them.
9. Also, write a program that reads in a small integer n, passes it to a function which iterates n times returning "God is good".

Conclusion

This was a very interesting unit where you learnt and implemented some basic principles of programming. By now you must have appreciated programming concepts like data types, variables, operators, control structures and functions. These constructs are emphasized because they are common to most programming languages.

Assessment

1. Show the general form for;
    - i. if statement
    - ii. if/else statement
    - iii. nested if statement
    - iv. nested if/else statement
    - v. conditional statement
    - vi. switch statement
    - vii. while statement
    - viii. do/while statement and
    - ix. a for statement
    - x. a function
2. In some situations, control structures execute more than one statement. Why is it important to block statements when executing more than one instruction?
3. Explain the difference between do/while statement and a while statement
4. Usually, control structures statements for example if(x>y) are not terminated at the end. Why?
5. What is the difference between a formal parameter and an actual parameter?
6. What is a function prototype? what is its significance in a program?
7. Differentiate between local and global variables.
8. What are some of the advantages of using global variables in a program?
9. What is a function header?
10. Give the advantages of using functions in a program

Arrays and strings

Introduction

Array is a data structure that stores data items of the same type. For example, to store five integers in the memory, you are required to declare five variables of type int (e.g. int a, b, c, d, e;) but an array comes in handy for this case. Using an array, you invent one identifier, then define the number of elements or the size of the array. Just like all other variables, an array must be declared before use. Its general form looks like this;

type array_name[size];

The type specifies the data type of the elements that will be stored in the array which includes; int, float, char, double e.t.c. While the size indicate the maximum number of elements that can be stored inside the array. For example;

```
int abcde[5];

float balance[10];
```

Array elements are accessed using the array name and its subscript or index. Array elements are stored in a contiguous manner and the first element of the array starts with a subscript zero(0), the last element of the array ends with size of the array minus one ([size-1]). In our example, the first element is abcde[0]....[4]. See the figure 3.16;

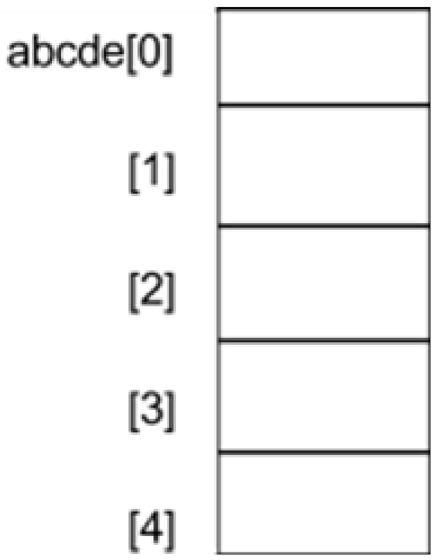


Figure 3.16 illustration of an array declaration

An array can be initialized or assigned values as;

type array_name[ ]={values};

For example;

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

    or;

int abcde[5];

abcde[0]=10;

abcde[1]=20;

abcde[2]=30;

abcde[3]=40;

abcde[4]=50;
```

See figure 3.17;

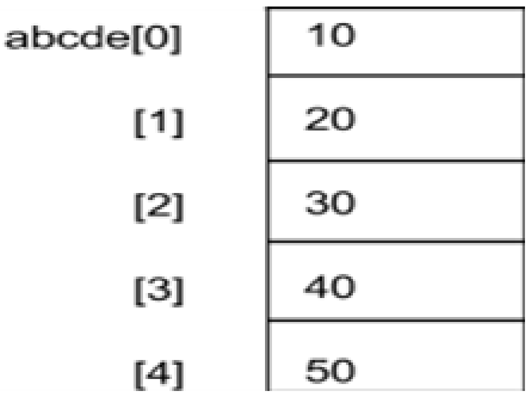


Figure 3.17 Sample of array initialization

For statement is used to manipulate the contents of the array. To illustrate this, lets us write this simple program;

Output:

```
abcde[0]=10;

abcde[1]=20;

abcde[2]=30;

abcde[3]=40;

abcde[4]=50;
```

Here, x is our control variable which we use to access or manipulate the contents of the array.

We have other types of arrays known as multidimensional arrays. Multidimensional array is an array of arrays. For example a two dimensional array is a multidimensional array that tries to represent elements inform of a table or matrices. It has two subscripts; one for the row and the other for the column. For example, below is a two dimensional array with three rows and 3 columns. It stores nine values (i.e. 3x3=9) or can be termed as a three by three table storing nine values, see figure 3.18.

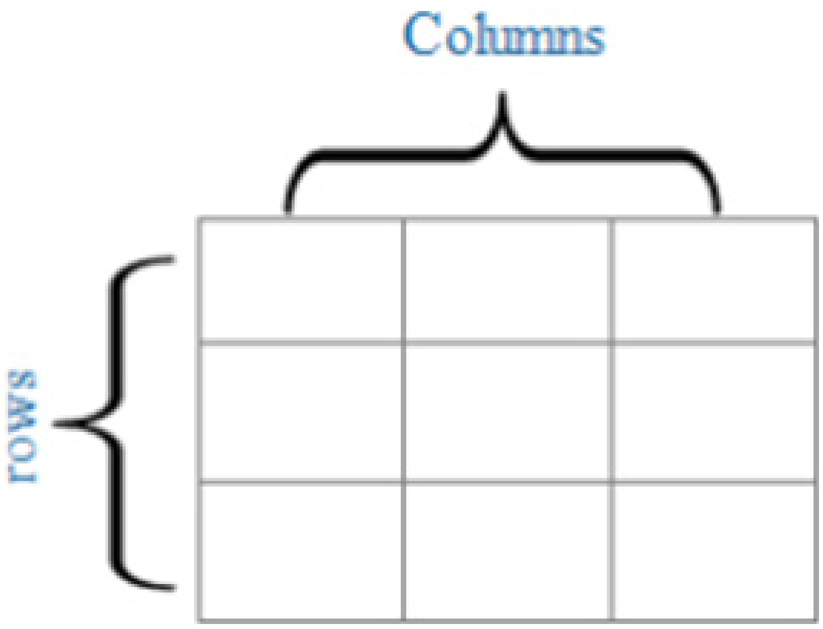


Figure 3.18 A three by three array

General form of a two dimensional array;

type array_name[row_size][column_size];

For example,

Figure 3.18 shows an array of three rows and three columns. Figure 3.19 shows an array named marks which is declared as a matrix having 4 rows and 5 columns. It can be declared as;

```
int marks[4][5];  or

float marks[4][5];
```

The first element of the array is marks[0][0] and last element is marks[3][4]. Figure 3.19 below illustrates a two dimensional array;

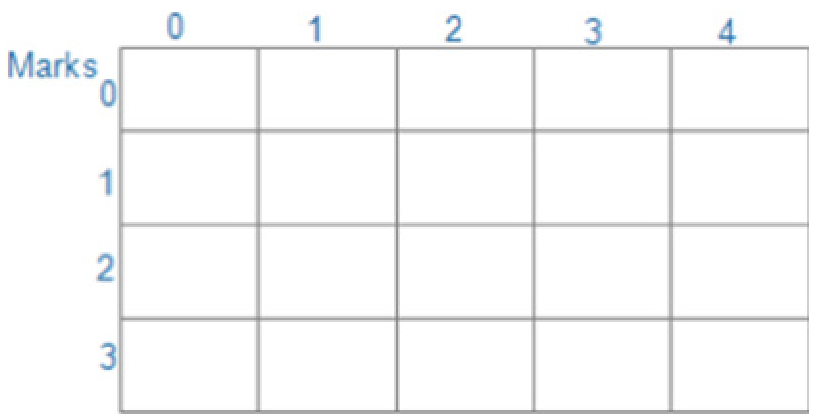


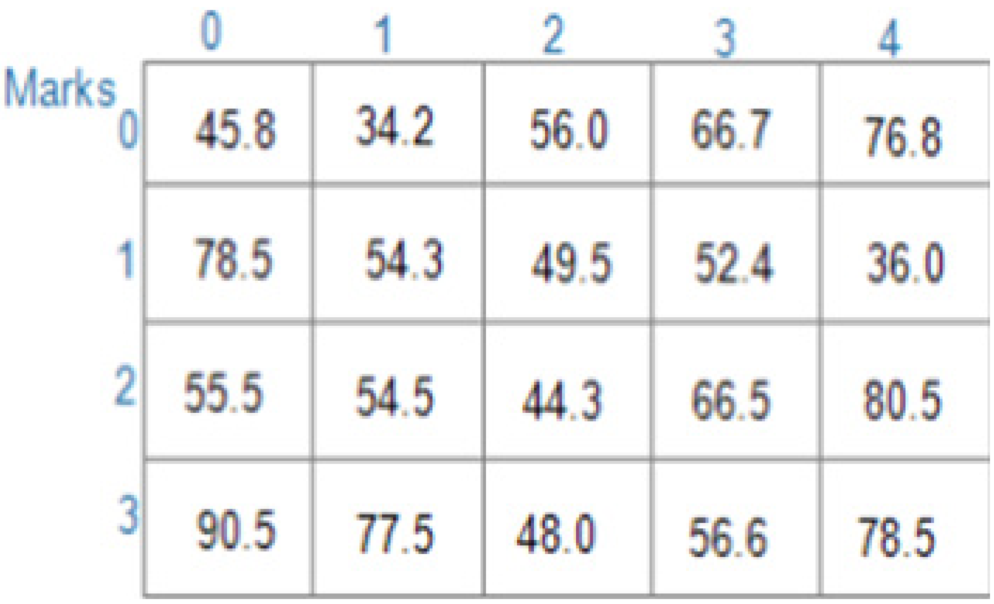
Figure 3.19 A sample of two dimensional array

Like the above one dimensional array, this array can be initialized as (see also figure 3.20);

data_type array_name[ ][ ]={values};

For example;

```
float marks[4][5]={{45.8, 34.2,56.0, 66.7,76.8}, {78.5, 54.3, 49.5,
52.4, 36.0},
        {55.5, 54.5, 44.3, 66.5, 80.5}, {90.5, 77.5, 48.0, 56.6, 78.5} };
```



Marks	0	1	2	3	4
0	45.8	34.2	56.0	66.7	76.8
1	78.5	54.3	49.5	52.4	36.0
2	55.5	54.5	44.3	66.5	80.5
3	90.5	77.5	48.0	56.6	78.5

Figure 3.20 An illustration of marks array

A simple program to illustrate the two dimensional array

Strings

A string is an array of characters that can be terminated with a null value '\0'. Here we have defined a string using two words which we have encountered in this unit; as an array and character (char). Meaning a string is declared using the same format as an array. Here;

type string_name[size];

Type is char because it is an array of characters, string name denotes the identifier that marks the location in the memory for storing characters. Size denotes the number of characters that can be contained in the string. For example;

"Noelajemu" is a string enclosed in double quotation mark while 'y' is a character enclosed in a single quotation mark. A string takes the form;

```
char first_Name[10];

char last_Name[10];
```

A string can be initialized as;

```
char first_Name[10]= {'N', 'o', 'e', 'l', 'a', 'j', 'e', 'm', 'u','\0'};

            or;

char first_Name[10]="Noelajemu";
```

Activity Details

You are required to read more on arrays and strings. After reading, you will be required to write programs that illustrate the concept of arrays and strings. You are advised to use Fundamentals of programming languages by Dipali P. Baviskar, Programming and PROB solving using C. by ISRD and Programming and problem solving through 'C' languages by Priya and Ranjeet. You can also obtain relevant information from the internet. For example; http://www.tutorialspoint.com/cprogramming/index.htm. This activity requires you to do a small research. After reading the said topics, you will be required to

- write a program that finds the sum of values stored in a one dimensional array. Come up with size of the array , data type and values
- write a program that reads in values to a two dimensional array, then finds the average. Invent your own array type, size and values. Print all the elements of the array.
- Sometimes it is necessary to limit the use multidimensional arrays in our programs. (effects of multidimensional arrays on our system). Research on this.

Conclusion

We completed this unit with two important concepts in programming; arrays and strings. As you can see, these two constructs are almost inevitable when developing a program or designing a solution. One, we always deal with series of data items, probably of the same type and two we use strings. Therefore, it is important that you understand these concepts. Finally, practice will help you appreciate programming.

<u>Assessment</u>

1. Define the following terms;

- i. an array
- ii. a string
-

2. Why do we use a nested for statement when dealing with a two dimensional array?
3. What is wrong with the following statement
4. char name[5]={'n', 'o', 'e', 'l', 'a' };. Explain your answer

Unit Summary

This unit entailed both theory and hands-on experience of programming principles. You have been introduced to problem solving using C language. Therefore, the unit started with C environment including simple programs and the use of printf and scanf() functions. It also looked at data types, variables, constants, operators, control structures, functions, arrays and strings. By now, you should be able to write programs using all the constructs that were taught in this unit.

Unit Assessment

1.	Declare and initialize an integer variable called total to 0	(2marks)
2.	Declare and initialize a character called initial to the letter K	(2marks)
3.	use const keyword to declare a constant PI and assign 3.1	(3marks)
4.	Define constant PI which has the value 3.14	(3marks)
5.	Assign the value of the integer variable a to variable b	(2marks)
6.	Store the sum of two variables c and e in sum	(2marks)
7.	Add the value in a to the value b and leave the answer in	(3marks)
8.	Divide the value in sum by 9 and leave the answer in sum	(2marks)
9.	Read in an integer to y	(3marks)
10.	Read in a character to c	(3marks)

11. Assign the result of dividing the integer variable total by 5 into the float variable balances. Use type casting to ensure that the remainder is also held by the float variable. (3marks)

12. Write a while loop statement to print values 1 to 15 on the screen (4marks)

13. Write a for loop statement to print the values 3 to 7 (4marks)

14. Write a for loop statement which adds all values between 50 and 100 into a variable called sum. (5marks)

15. Write an if statement that compares the value of an integer called product against the value 15, and if it is greater, print the text string "you exceeded". (5marks)

16. If the variable x is equal to y, print the value of x, else print the value of y. (5marks)

17. Convert the following expression to an if/else statement (4marks)

```
v=(a<b) ? a : b;
```

18. Declare a one dimensional array called limits that stores six integers (3marks)

19. Assign 23,46,67,87,53,44 to the array elements of limits (see 18 above). (3marks)

20. Assign 56 to array element four of limits (2marks)

21. Assign 34 to the fifth element of array limits. (2marks)

22. print the value in the first element of the array limits. (3marks)

23. Declare a character based array called names of 20 elements. (2marks)

24. Assign character 'S' to the array element 10 of names. (3marks)

25. Write a statement for reading in values to an array limits (see 18). (4marks)

26. Write a statement for reading in characters to array names (see 23). (4marks)

27. Rewrite the following program code using a switch statement (4marks)

```
if( c == 'A' )
                sum = 0;
        else if ( c == 'B' )
                sum+= 1;
        else if( c == 'Z' )
                sum = 26;
        else
                printf("Unknown character %c\n", c );
```

Instructions

You are required to write single statements for all the questions 1 through 27.

Questions cover everything that was presented in this unit in order to assess overall understanding. Answer them carefully and if your score falls

- below 40%, redo the readings and activities.
- between 40% and 60%, redo the readings on your weak points
- above 60%, you have substantial amount of knowledge

Grading Scheme

The unit assessment (formative assessment) has been assigned marks. The instructor will assign marks to activity assessment then compute all. At the end of unit four all the assessments must be computed to 10% of the total examination marks.

Answers

Formative Assessment Answers

```
1.   int total=0;

2.   char initial='K';

3.   float const PI = 3.14;

4.   #define PI 3.14

5.   int a, b; b=a;

6.   sum=c+e;

7.   a+=b; or a=a+b;

8.   sum/=9;

9.   scanf("%d", &y);

10.  scanf("%c", &c);

11.  balances=(float) total/5;

12.  int values=1;while (values<=15) {printf("%d\n",
values);++values  }

13.  for(int values=3; values<=7; ++values)

14.  for(int values=51, sum=0; values<100; sum+=values,
++values) or int sum=0,              values; for(values=51;
values<100; ++values) sum+=values;
```

15. `if(product>15) printf("you exceeded");`

16. `if(x==y) printf("%d", x); else printf("%d", y);`

17. `if (a<b)    v=a; else      v=b;`

18. `int limits [6];`

19. `limits [6]= { 23,46,67,87,53,44};`

20. `limits[4]= 56;`

21. `limits[4]=34;`

22. `printf("%d",limits[0]);`

23. `char names [20];`

24. `names[10]='S';`

25. `for (x=0; x<6; ++x) scanf("%d" &limits[x]);`

26. `scanf("%s", names);`

27.
```
switch (c) {
case 'A':
            sum = 0;
            break;
          case 'B':
            sum+= 1;
            break;
          case 'C':
            sum = 26;
            break;
    default:
printf("Unknown character %c\n", c );    }
```

Unit Readings and Other Resources

- Fundamentals of programming languages. Dipali P. Baviskar, Technical Publication Pune.
- Priya H., and Ranjeet, R., (2006). Programming and problem Solving Through "C" Language. Firewall Media.
- Programming and PROB solving using C. ISRD, Tata McGraw-Hill education.

Optional readings and other resources:

Programming: Grade in Industrial Technology Engineering. Creative

Commons. On: http://ocw.uc3m.es/ingenieria-informatica/programming-in-

c-language-2013/IntroductiontoDevCIDE.pdf

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

Unit 4. Program Testing, Debugging and Documentation.

Unit Introduction

This unit discusses the different types of errors that emerge during program development and various ways of correcting. It will also, in a very basic way look at documentation in software development. This unit is very essential to the learner as the learner gets to know of these three vital concepts (testing, debugging and documentation) which are common to all programming languages and paradigms. The contents of this unit were obtained from the listed reference books. Therefore, you are highly advised to access these resources for further reading and be able to work on the tasks/activities.

Unit Objectives

- Upon completion of this unit you should be able to:
- explain program testing and debugging.
- apply basic types of program testing.
- differentiate the basic types of program testing.
- debug a program.
- document a program using comments and explain other types of software documentation

Key Terms

Testing: It is a process of executing a program with the intent of finding an error.

Debugging: It is a methodological process of finding and reducing the number of bugs or defects in a computer program thus making it behave as expected

Documentation: It consists of instructions for using a program

Debuggers: These are programs which allow you to execute your program in a controlled manner, so that you can look inside your program to find a logic or run-time bug.

Test plan: A Software Test Plan is a document describing the testing scope and activities. It is the basis for formally testing any software/product in a project. It describes how you intend to test your program, that is, which inputs you plan to test it on and why those are good.

A test case: It is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.

A bug: A software bug is a problem causing a program to crash or produce invalid output. The problem is caused by insufficient or erroneous logic. A bug can be an error, mistake, defect or fault, which may cause failure or deviation from expected results. Most bugs are due to human errors in source code or its design. A program is said to be buggy when it contains a large number of bugs, which affect program functionality and cause incorrect results.

Learning Activities: Program Errors and Testing

Introduction

There are three basic types of errors which include; syntax or compilation errors, run time exception errors and logical errors. A program must be tested for all these types of errors. This ensures the reliability, trustworthiness, effectiveness, performance among other attributes of a good program. Testing also is primary divided into two basic types namely; white box testing and black box testing. Using white box testing methods a programmer can test cases which guarantee that every individual part in a program has been exercised at least once, all logical decisions on their true and false sides are executed, all loops at their boundaries within their operational bounds are executed and internal data structures to ensure their validity are exercised.

Black box testing attempts to find errors in the following categories; 1. incorrect or missing functions, 2. interface errors, 3. errors in data structures or external database access, 4. performance errors, 5.initialization and termination errors. Black box testing is a complementary approach that is likely to uncover a different class of errors that white box testing does not.

It is practically impossible to prove that the program is correct on all inputs. Instead, skeptics need to be convinced that it mostly works right by testing the program on various inputs and documenting what has been done. This document is called a test plan and one must be provided for each program.

Activity Details

This activity therefore, involves reading and writing a simple program that will be used to reinforce white box and black box testing concepts. You are required to read the topics on types of errors that emerge during program development and testing techniques which include white box and black box testing. The readings can be obtained from Programming and problem solving through 'C' languages by Harsha Priya, r. Ranjeet and Fundamentals of programming languages book by Dipali P. Baviskar. These reference books are listed in the reference section for this unit. To accomplish the task bullet three in this activity, the learner is required to consult the reading materials provided in the reference section for unit three and read on functions.

In this activity you are required to;

- Differentiate between syntax, logical and runtime errors.
- Describe how whitebox and black box testing is done
- Using C language, write a simple program that asks the user for a value (reads in an integer n), then passes it to a function that returns the sum of the values from 1 to n.
- Test the above program using whitebox and black box testing. Describe how you managed to test your program using the two techniques. For black box testing, you are required to use different types of data. How does the program respond to this input?
- In a very simple manner, describe the test case that you used to accomplish the above task.

Conclusion

At this point of study, the learner should be well equipped with the knowledge on the types of programming errors and error testing techniques which include white box and black box testing.

Assessment

1. Name and briefly describe three types of program errors. Give an example for each type of error.
2. Why is it necessary to test a program even if we know it works well?
3. Discuss two types of program testing.
4. Name any one simplest way of performing white box testing.

Program Debugging

Introduction

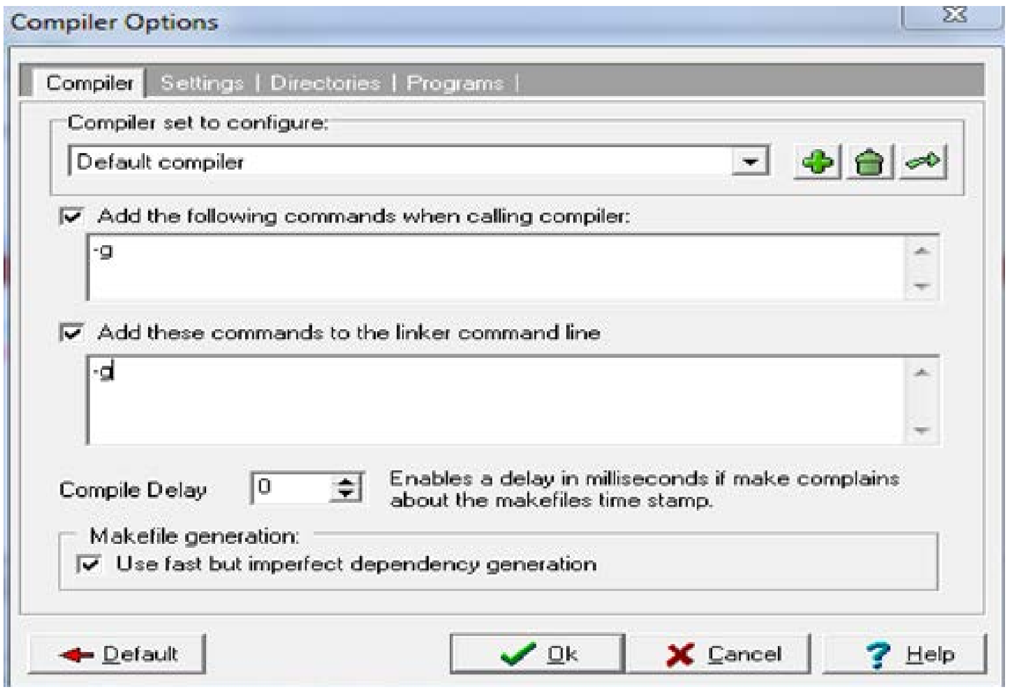
Debugging involves a process of finding and reducing the number of errors making sure the program does exactly that which is expected to do. This process tends to be harder when various subsystems/subprograms/subroutines highly depend on each other or are tightly coupled, as changes in one may impact other dependent subprograms thus errors emerging in other subprograms.

Table 4.1 The difference between testing and debugging: source (Baviskar, 2009)

Testing	Debugging
Testing finds cases where a program does not meet its specification	The process of debugging involves analyzing and possibly extending the given program that does not meet the specifications in order to find a new program that is close to the original and does satisfy the specification
Its objective is to demonstrate that the program meets its design specification	The objective is to detect the exact cause and remove known errors in the program
Testing is complete when all desired verifications against specifications have been performed	Debugging is complete when all known errors in the program have been fixed
Testing can begin in the early stages of software development	Debugging can begin only after the program is coded
Testing is the process of validating the correctness of the program	Debugging is the process of eliminating the errors in a program

Logical errors can be detected by doing hand simulation of the program code and by putting the print statements in the program code. Another approach for detecting logical errors is by use of a debugger. This is the most commonly used technique. Debugger is the software program that assist a programmer in following a program's execution step-by-step allowing the display of intermediate calculation results.

The IDE includes a debugger, which is a supporting tool to find runtime errors. To use the debugger, it is necessary to modify the compiler options to include debugging information in the object and the executable files (Tools > Compiler Options > Compiler). This is done by adding the –g parameter to the calls to the compiler and the linker.


Compiler Options

Compiler set to configure: Default compiler

[✓] Add the following commands when calling compiler:

```
-g
```

[✓] Add these commands to the linker command line

```
-g
```

Compile Delay: 0 — Enables a delay in milliseconds if make complains about the makefiles time stamp.

Makefile generation:

[✓] Use fast but imperfect dependency generation

Figure 4.1 Debugger tool source:

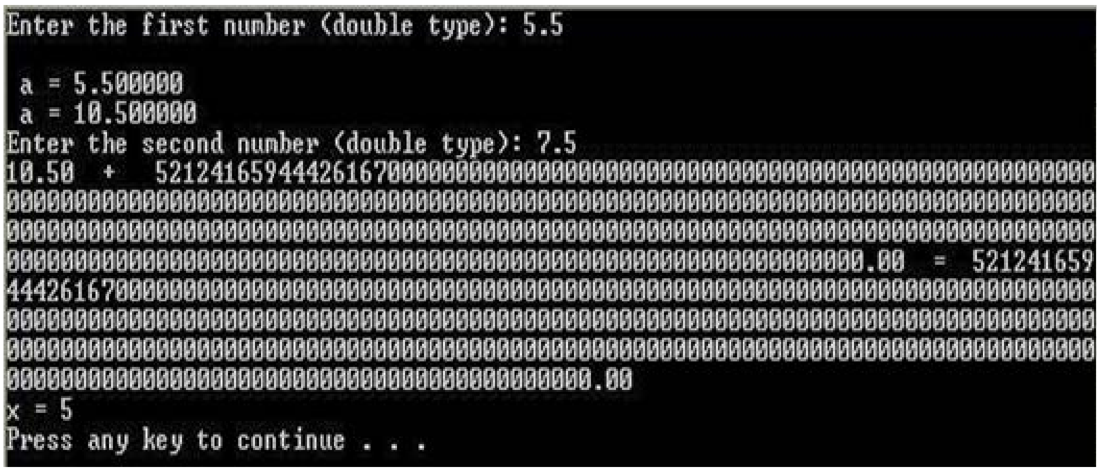
(http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf)

When the program is compiled and linked with these options, we can run the program in debug mode by clicking on the Debug button (F8). This mode allows running the program step by step (instructions are executed one-by-one), stopping the execution at breakpoints, or watch the value of a variable at any time of the execution.

The Debug mode allows us to consult or watch the value of a variable at any time during the execution of the program. To watch a variable, the program must be stopped (with a breakpoint or after using Run to Cursor).

For example, when the following code is compiled, and then executed in C program: Obtained from; (http://faculty.kfupm.edu.sa/ICS/said/ics103_101/Lab03%20Supplement%20-%20How%20to%20use%20Dev%20C++%20Debugger.doc).

The following output is obtained for input values a = 5.5 and b = 7.5:



```
Enter the first number (double type): 5.5

 a = 5.500000
 a = 10.500000
Enter the second number (double type): 7.5
10.50  +   521241659444261670000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000.00  =  521241659
44426167000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000.00
x = 5
Press any key to continue . . .
```

Figure 4.2 Output for debugger sample output

The output indicates that the program has one or more logic errors. The Dev C++ debugger can be used to find the error(s).

Activity Details

To perform this activity, the learner is required to read on program debugging; That is, how to debug syntax and logical errors using Fundamentals of programming languages book by Dipali P. Baviskar and http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf to learn how to use a debugger; creating breakpoints and watches. These materials will help you understand program debugging and guide you step by step to learn how to debug a program. In this activity you are required to:

- Develop a C program that asks the user for two integer values, calculates modulo, and prints the result on the screen. Add breakpoints and watches to all the variables of the program. Run the program step by step and check how the values are changed by the instructions. What is the value of the variables before reading them from the keyboard?
- Describe how you accomplished the above task.
- Using a C program of your own, create a new project in Dev C++ IDE Program. Add breakpoints and watches.

Conclusion

This activity has been an exciting one and so far we have learned how to debug program errors by doing hand simulation of the program code by use of a debugger which is the most commonly used technique. Debugger is the software program that assist a programmer in following a program's execution step-by-step allowing the display of intermediate calculation results. We learned how to add breakpoints and watches to our code with the use of a debugger.

Assessment

1. What do you understand with the term a bug?
2. List the differences between testing and debugging?
3. Describe how to debug syntax and logical errors
4. Software Documentation

Introduction

Documentation is a very important aspect of programming, because in your programming career, you will read a lot more code, than you write. If documentation for a program is proper, then understanding the program is good as reading the programmer wrote the program- Harsha Priya, r. Ranjeet. Documentation is often categorized into; installation, reference and tutorial. Documentation has many advantages which include; easy usage of a program, easy maintenance, easy modification, easy to understand the logic of a program from documented records rather than its code, flowcharts or comments used within the program are very helpful in understanding the functionality of the whole program, documented records are quite helpful in restarting a software project that was postponed due to some reason or the other, and probably, the job need not be started from scratch and the old ideas may still be easily recapitulated which saves lot of time and avoids duplication of work (reusability). Various forms of documentation include; comments, system manual and user manual.

Activity Details

This also is a reading activity and the learner is required to read on program/software documentation using Fundamentals of programming languages book by Dipali P. Baviskar. We shall practice the concept of documentation in a simple way by use of comments. The learner can visit http://en.wikibooks.org/wiki/C_Programming/Structure_and_style to read on documentation styles by use of comments. In this activity, you are required to;

- Briefly describe a self documenting code. Use a simple example.
- Describe user and system manuals
- Using comments, document the following program;
- Using C program, write a program of your own then document it using both single line and multi-line commenting style. In your program, show the concept of program self documentation

Conclusion

In this activity you have been introduced to software documentation; an activity that is essential and is practiced by programmers. The major point at this level was to help you understand the meaning of self documenting code and how to document your own code using comments. You also learned about user and systems manuals as part of software documentation. This far, we believe you appreciate the benefits of software documentation.

Assessment

1. What do you understand by the term software/program documentation?
2. Discuss the benefits of program documentation
3. Discuss the three categories of software documentation
4. Briefly explain the forms of program/software documentation

Unit Summary

This unit looked at three major programming concepts that are important and a new student of programming needs to know. These concepts include; program testing, debugging and documentation.

Unit Assessment

Check your understanding!

Formative assessment

1. Define the following

   - A bug____ (2marks)
   - Program testing__________________________________ (2marks)
   - Debugging____ (3marks)

2. Compilation errors are detected by_________________________ (1mark)

3. Execution errors are also known as_____________________________ (1mark)

4. A program statement was written;

   x=d\y; which error does it generate? (2marks)

5. Match the following; (4marks)

White box testing	test cases that guarantee that every individual part in a program has been exercised at least once, all logical decisions on their true and false sides are executed, all loops at their boundaries within their operational bounds are executed and internal data structures to ensure their validity are exercised
b. Black Box testing	ii. a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.
c. Test plan	iii. document describing the testing scope and activities
d. Test case	iv. attempts to find errors like incorrect or missing functions, interface errors, errors in data structures or external database access, performance errors, and initialization and termination errors by inputting data.

6. To watch a variable, the program must be stopped with a _______________ (2marks)

7. Logical errors can be detected by___________________and _____________ (2marks)

8. Documentation can be grouped into three categories. Name the three categories of software documentation. (3marks)

Instructions

Questions cover everything that was presented in this unit in order to assess overall understanding. Answer them carefully and if your score falls

- below 40%, redo the readings and activities.
- between 40% and 60%, redo the readings on your weak points
- above 60%, you have substantial amount of knowledge

Grading Scheme

The unit assessment (formative assessment) has been assigned marks. The instructor will assign marks to activity assessment then compute all. At the end of unit four all the assessments must be computed to 10% of the total examination marks.

Answers

Formative assessment answers

1. i. A bug is a problem causing a program to crash or produce invalid output. The problem is caused by insufficient or erroneous logic. A bug can be an error, mistake, defect or fault, which may cause failure or deviation from expected results.

   Testing is a process of executing a program with the intent of finding an error.

   Debugging is a methodological process of finding and reducing the number of bugs or defects in a computer program thus making it behave as expected

2. the compiler
3. run time errors
4. syntax
5. and I

   b iv

   c iii

   d. ii
6. breakpoint
7. hand simulations and using a debugger
8. installation, reference and tutorial

Unit Readings and Other Resources

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

Course Summary

The journey to programming has been long but exciting--we believe so. Did you enjoy it---you said 'yes', right? We started with the history and paradigms of programming. Here, we learned that software industry is dynamic--ever changing and growing. New programs (or compilers) as well as paradigms are continuously developed or improved. The old ones disappear or are improved to meet the current market demand or technology. The next step helped us to appreciate computers as problem solving tools. We looked at various problem solving techniques which include top-down analysis, bottom-up analysis and the steps that are followed when designing solutions using a computer (of course we are talking about a digital computer). Then came the moment that we all were waiting for---we know you were eagerly waiting too. Coding! We implemented our design solutions (flowcharts/algorithms) using C language. Where we started from C environment all the way to strings. At that point, we needed to understand programming best practices that will help us develop efficient, effective, reliable, dependable, user acceptable and maintainable software. We accomplished this by learning three things; testing, debugging and documentation techniques.

Course Assessment 1

Review Exercise: Descriptive

1. Define the following; (7marks)
    i. Programming
    ii. Syntax
    iii. Semantics
    iv. Program code
    v. Algorithm
    vi. Flowchart
    vii. Bug
2. Differentiate between; (8marks)
    i. Unary and binary operators
    ii. Compiler and interpreters
    iii. Low level and high level languages
    iv. testing and debugging
    v. Top down design and bottom up design techniques
    vi. Structured programming paradigm and object oriented programming paradigm

3. Programming languages have come a long way in terms of levels, generations and paradigms. List any three factors that contribute to the design of new languages (3marks)
4. List all the steps involved in problem solving (5marks)
5. An algorithm is language independent. Discuss (2marks)
6. Give any two advantages of flowcharts over algorithms (2marks)
7. List any four properties of a good algorithm (4marks)
8. Structured programming is a paradigm that revolutionized programming industry. Name three advantages of structured programming (3marks)
9. Name any two errors that emerge during program execution (2marks)
10. Discuss two major methods of testing a program (4marks)
11. Programming languages support various data types. Name any four primary data types used in C language (4marks)
12. What is the importance of learning and understanding data types? (1mark)
13. Elaborate on the steps followed when executing a C program. (4marks)
14. What do you understand by the terms; (5marks)
    i. A keyword
    ii. A variable
    iii. An identifier
    iv. Operator associativity
15. Programming languages support various types of operators. List any five types of operators (5mark)

Instructions

Answer all the questions within two hours

Grading Scheme

The course assessment (review exercise-descriptive assessment) has been assigned marks.

Therefore you are required to apply the assigned marks for grading this assessment.

Answers

Solutions to descriptive assessment

Q1.

i. Programming is the activity of writing instructions to be executed by the computer with aim of solving problems. These instructions tell the computer to do something.

ii. Programming syntax is the set of rules that defines the combinations of symbols that are considered to be a correctly structured document or fragment in that language

iii. Semantics define the meaning of syntactically legal strings defined by a specific programming language, showing the computation involved

iv. Program code a set of instructions meant to execute a particular task also known as source code

v. Algorithm is set of steps generated by the programmer to help in solving a particular problem.

vi. Flowchart is a pictorial representation of an algorithm

vii. Bug is a problem causing a program to crash or produce invalid output

Q2.

i. unary operator operates on single operand whereas binary operators operate on two operands

ii. Compiler analyses the entire program and reports all the errors at once whereas interpreter 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.

iii. Low level is machine dependent whereas high level languages are machine independent

iv. testing is a process of executing a program with the intent of finding an error whereas debugging is a methodological process of finding and reducing the number of bugs or defects in a computer program thus making it behave as expected

v. Top down is a design technique where a program is divided into a number of smaller and simpler tasks which can be tackled separately. In this technique, the main program is written and tested first before subprograms are written. Thus, it looks at the solution in a wider picture first design whereas bottom up design techniques is a design technique where a program is divided into a number of smaller and simpler tasks which can be tackled separately. In this technique, subprograms are written and tested first before the main program is written.

vi. Structured programming paradigm is function oriented, it emphasizes on procedure, data and functions are handled separately whereas object oriented programming paradigm emphasis is on data rather than procedure, divides a program into what are known as object entities (classes), Functions and data are bound together

Q3

i. Computer capabilities

ii. Applications

iii. Programming methods

iv. Implementation methods

v. Theoretical studies

vi. Standardization

Q4.

i. Requirements specification

ii. Requirements analysis

iii. iii.Software design

iv. Implementation

v. Testing

vi. Deployment

Q5.

An algorithm is language independent because the programmer may be independent from the designer hence the developer can choose any language of preference

Q6.

i. Seeing the logic visually/graphically in front makes the steps much simpler to follow.

ii. Less intimidating than pseudo code, which itself is also very easy to understand but, still consists of statements which must be read. With flowcharting applications like Microsoft's Visio, even people with the weakest of artistic skill can put together a flowchart quickly and efficiently.

Q7.

i. Finiteness

ii. Definiteness/simple

iii. Generality

iv. Effectiveness

v. unambiguous

vi. effective

vii. Complete

viii. Input/output

Q8.

i. Allows several programmers to code simultaneously

ii. Decreases code complexity

iii. Decreases debugging time

iv. Allows for reuse

Q9.

i. Syntax errors

ii. run time errors (logic errors)

Q10.

i. Black box testing

ii. white box testing

Q11.

i. integer (int)

ii. character (char)

iii. float

iv. Void

Q12.

To know the amount of space a particular type uses hence its helps to select appropriate type to avoid memory wastage

Q13.

Student to elaborate on;

compiling, linking/loading and execution

Q14.

i. A keyword is a program's reserved word that has a special meaning to the compiler

ii. A variable is memory storage location

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

Q15.

i. arithmetic

ii. assignment

iii. logical

iv. relational

v. increment and decrement operators

Course Assessment 2

Review Exercise: Practical

1. Write an algorithm that;

   i. Evaluates factorial of a number
   (5marks)

   ii. Evaluates average of a number
   (5marks)

   iii. Finds the smallest number from an array of integers (5marks)

   iv. Converts 12010 to binary number system
   (5marks)

2. Write a C program that;

   i. displays "I am a guru in programming" on the screen.
   (3marks)

   ii. reads in two integers, two real numbers, a character and displays them on the Screen
   (5marks)

   iii. divides 7 by 5 then prints the answer including the decimal part on the screen (4marks)

   iv. accepts your name and age then computes your age in the next birth day (using an operator) and displays on screen as follows;
   (5marks)

   Name:

   Age:

   Next B/day:

   v. reads in a value then passes it to a function that returns factorial
   (5marks)

   vi. reads in 10 values to an array, evaluates sum and prints sum on the screen (5marks)

3. Using Ms-word, draw a flowchart that calculates sum of a value greater than 0 and less or equal to n (5marks)

Instructions

Answer all the questions within two hours. Submit this assessment to your instructor.

Grading Scheme

The course assessment (review exercise: Practical) has been assigned marks.

Therefore you are required to apply the assigned marks for grading this assessment. The instructor will be required to distribute marks accordingly (for example, for partly correct answered questions)

Answers

The instructor will be required to test students' feedback then provide appropriate solutions.

Course References

1. Baviskar, D. P. (2009). Fundamentals of programming languages. 1st edition. Technical Publication Pane.
2. Priya and Ranjeet, (2006). Programming and problem solving through "C" languages. Firewall media.
3. ISRD, (2008). Programming and PROB solving using C. Tata McGraw-Hill education.
4. Programming: Grade in Industrial Technology Engineering. On: http://ocw.uc3m.es/ingenieria-informatica/programming-in-c-language-2013/IntroductiontoDevCIDE.pdf.
5. 'Regan, G. (2012). A brief history of computing. 2nd edition, Springer London.
6. Mitchell, J. C. (2003). Concepts in programming Languages. Cambridge University Press. Chapter 1.
7. Gary. M., (2009). Fundamentals of Programming: with Object Oriented Programming. Gary Marrel.
8. Farrell, J. (2012). Programming Logic and design, Comprehensive. 7th edition. Cengage Learning.
9. Hanly, J. R. and Koffman, E. F. (2009). Problem solving and program design in C. 6 illustrated. Addison-wesley.
10. http://archive.oreilly.com/pub/a/oreilly//news/languageposter_0504.html
11. http://www.softpanorama.org/History/lang_history.shtml#General

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