CSC 212: COMPUTER PROGRAMMING II

Published by the Centre for Open and Distance Learning,
University of Ilorin, Nigeria

E-mail: codl@unilorin.edu.ng
Website: https://codl.unilorin.edu.ng

Course Development Team

Content Authors

Idowu Dauda, Oladipo (Ph.D.)

Content Editors

Mrs. Bankole

Instructional Design

Miss. Damilola Adesodun

Mr. Olawale S. Koledafe

From the Vice Chancellor

Courseware development for instructional use by the Centre for Open and Distance Learning (CODL) has been achieved through the dedication of authors and the team involved in quality assurance based on the core values of the University of Ilorin. The availability, relevance and use of the courseware cannot be timelier than now that the whole world has to bring online education to the front burner. A necessary equipping for addressing some of the weaknesses of regular classroom teaching and learning has thus been achieved in this effort.

This basic course material is available in different electronic modes to ease access and use for the students. They are available on the University's website for download to students and others who have interest in learning from the contents. This is UNILORIN CODL's way of extending knowledge and promoting skills acquisition as open source to those who are interested. As expected, graduates of the University of Ilorin are equipped with requisite skills and competencies for excellence in life. That same expectation applies to all users of these learning materials.

Needless to say, that availability and delivery of the courseware to achieve expected CODL goals are of essence. Ultimate attention is paid to quality and excellence in these complementary processes of teaching and learning. Students are confident that they have the best available to them in every sense.

It is hoped that students will make the best use of these valuable course materials.

Professor S. A. Abdulkareem
Vice Chancellor

Foreword

Courseware is the livewire of Open and Distance Learning. Whereas some institutions and tutors depend entirely on Open Educational Resources (OER), CODL at the University of Ilorin considered it necessary to develop its materials. Rich as OERs are and widely as they are deployed for supporting online education, adding to them in content and quality by individuals and institutions guarantees progress.

Pursuing this goal has brought the best out of the Course Development Team across Faculties in the University. Despite giving attention to competing assignments within their work setting, the team has created time and eventually delivered. The development of the courseware is similar in many ways to the experience of a pregnant mother eagerly looking forward to the delivery date.

As with the eagerness for a coming baby, great expectation pervaded the air from the University Administration, CODL, Faculty and the writers themselves. Careful attention to quality standards, ODL compliance and UNILORIN CODL House Style brought the best out from the course development team. Response to quality assurance with respect to writing, subject matter content, language and instructional design by the authors, reviewers, editors and designers, though painstaking, has yielded the course materials now made available primarily to CODL students as open resources.

Aiming at parity of standards and esteem with regular university programmes is usually an expectation from students on open and distance education programmes. The reason being that stakeholders hold the view that graduates of face-to-face teaching and learning are superior to those exposed to online education. CODL has the dual mode mandate. This implies a combination of face-to-face with open and distance education.

With this in mind, the Centre has developed its courseware to combine the strength of both modes to bring out the best from the students. CODL students and other categories of students of the University of Ilorin and similar institutions will find the courseware to be their most dependable companion for the acquisition of knowledge, skills and competences in the respective courses and programmes.

Activities, assessments, assignments, exercises, reports, discussions and projects at various points in the courseware are targeted at achieving the Outcomes of teaching ang learning. The courseware is interactive and directly points the attention of students and users to key issues helpful to their particular learning. The student's understanding has been viewed as a necessary ingredient at every point. Each course has also been broken into modules and their component units in an ordered sequence. In it all, developers look forward to successful completion by CODL students.

Courseware for the Bachelor of Science in Computer Science housed primarily in the Faculty of Communication and Information Science provide the foundational model for Open and Distance Learning in the Centre for Open and Distance Learning at the University of Ilorin.

Henry O. Owolabi
Director, CODL

COURSE GUIDE

CSC 212 (Computer Programming II) is a three (3) credit unit. The course deals with the fundamental concepts of Programming Languages with specific focus on C++ programming language. The course is organized into seven (7) distinct modules with 22 course unit, with each module addressing at least three study units.

The course begins by introducing you to the history of computer programming languages, history and fundamentals of C++ and how C++ is being complied. The second module discusses the basic program structure of C++ program where the basic Concept of C++ Programming Language was treated in details. In the next module we introduce you control structure in C++ where selection control, Loop/iteration control and break/termination control structure was discussed fully.

Function definition, function call and types of function were enumerated in module four. Furthermore, Composite data types were treated in module five where array, string, structure and union were discussed. The last two modules discuss the concept of object oriented programming (OOP) in C++ which is one of the important features of C++ programming language. Classes, objects and constructor were treated. Finally, advance OOP in C++ was treated which involves inheritance, data hiding and

Course Goal

The goal of this course is to expose students to the concept and fundamental syntax and structure of C++ language focusing on major aspects of programming ranging from variables, datatypes, operators, expression, statement, control structures, functions and composite types. This course will serve as a solid foundation for learning programming languages. Object Oriented Programming (OOP) Language and its fundamentals was also covered which differentiates C++ with the popular C language.

The course also provides student with the basic knowledge of the underpinning principles behind the Computer Programming Language and provides background knowledge for C++ Programming Language..

Related Courses
CSC 111 – Introduction to Computer Science I
CSC 112 – Introduction to Computer Science II
CSC231 – Computer Programming I

WORK PLAN

Learning Outcomes

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

I.Trace the Historical Development of Computer Systems;

Week 01

II. Mention the advantages, disadvantages and characteristics of Computer

Week 02

III.Differentiate among Computer Software, Hardware and Humanware in relation to their functions.

Week 03

IV.Perform various calculations in number systems (binary, decimal, octal decimal, and hexadecimal)

Week 04

V. Explain Boolean algebra, Logic Gates and form various logic circuits.

Week 05

VI. Discuss generations of programming languages

Week 06

Course Guide

Module 1

History and Program Development

Unit 1: History of Computer Programming Language

Unit 2: History and Fundamentals of C++

Unit 3: Compiled Languages and C++

Module 2

Program Structure in C++

Unit 1: Statement in C++ Program

Unit 2: Structure of C++ Program

Unit 3: Input and Output Standard

Module 3

Control Structures

Unit 1: Selection Control Structures

Unit 2: Loop/Iteration Control Structures

Unit 3: Break/Termination Control Structures

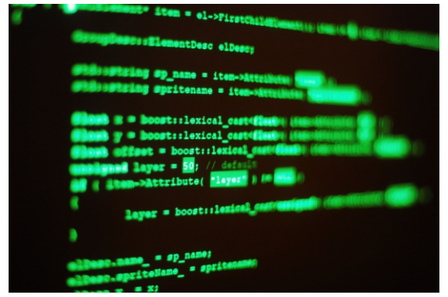
Module 4

Function

Unit 1: Function Definition

Unit 2: Function Call & Prototype (declaration)

Unit 3: Type of Function



Module 5

Composite Data Types

Unit 1: Array

Unit 2: String

Unit 3: Structure

Unit 4: Union

Module 6

OOP in C++

Unit 1: OOP Concept

Unit 2: Classes and Objects

Unit 3: Constructor in C++

Module 7

Advance OOP in C++

Unit 1: Inheritance

Unit 2: Data Hiding

Unit 3: Polymorphism

Course Requirements

Requirements for success

The CODL Programme is designed for learners who are absent from the lecturer in time and space. Therefore, you should refer to your Student Handbook, available on the website and in hard copy form, to get information on the procedure of distance/e-learning. You can contact the CODL helpdesk which is available 24/7 for every of your enquiry.

Visit CODL virtual classroom on http://codllms.unilorin.edu.ng. Then, log in with your credentials and click on CSC 111. Download and read through the unit of instruction for each week before the scheduled time of interaction with the course tutor/facilitator. You should also download and watch the relevant video and listen to the podcast so that you will understand and follow the course facilitator.

At the scheduled time, you are expected to log in to the classroom for interaction.

Self-assessment component of the courseware is available as exercises to help you learn and master the content you have gone through.

You are to answer the Tutor Marked Assignment (TMA) for each unit and submit for assessment.

Embedded Support Devices

Support menus for guide and references

Throughout your interaction with this course material, you will notice some set of icons used for easier navigation of this course materials. We advise that you familiarize yourself with each of these icons as they will help you in no small ways in achieving success and easy completion of this course. Find in the table below, the complete icon set and their meaning.

		
Introduction	**Learning Outcomes**	**Main Content**

 Summary	 Tutor Marked Assignment	 Self Assessment
 Web Resources	 Downloadable Resources	 Discuss with Colleagues
 References	 Futher Reading	 Self Exploration

Grading and Assessment

TMA

CA

Exam

Total

Picture:
Data science programming
Photo: freepik.com

Module 8

History and Program Development

Units

Unit 1 - History of Computer Programming Language
Unit 2 - History and fundamentals of c++
Unit 3 - compiled Language and c++

UNIT 1

History of Computer Programming Language

Introduction

In this unit, we will be discussing various type of computer programming language, the concept in computer program, we would also outline the characteristics of a good Program. Finally, we will analyze the Phases of Program Development. Let's get started!

A program is a set of instructions following the rules of the chosen language. It is important for you to note that without programs, computers are useless. A program is like a recipe. It contains a list of ingredients (called variables) and a list of directions (called statements) that tell the computer what to do with the variables.

Learning Outcomes

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

1. Explain computer programming language.
2. Define a program
3. Identify types of computer programming language
4. Explain concepts in computer Program.
5. Explain Language Types.

Main Content

Computer Programme

1 min

Let us further understand what computer programs are. The first concept I want you see here is programming language; a programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks. We often refer *programming*

language as high-level languages, some of which are BASIC, C, C++, COBOL, Java, FORTRAN, Ada, and Pascal. Each programming language has a unique set of keywords (words that it understands) and a special syntax for organizing program instructions.

Computer Programming language

I want you to know that there are various languages for expressing a set of detailed instructions for a digital computer. These instructions can be executed directly when they are in the computer manufacturer-specific numerical form known as machine language, after a simple substitution process when expressed in a corresponding assembly language, or after translation from some "higher-level" language. Although we have many computer languages, relatively few are widely used.

SAQ 1

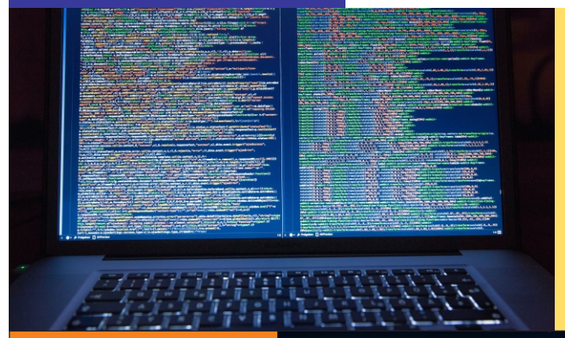
Machine and assembly languages are "low-level," requiring a programmer to manage explicitly all of a computer's characteristic features of data storage and operation. Unlike what we have in low-level languages,

high-level languages shield a programmer from worrying about such considerations and provide a notation that is more easily written and read by programmers.

Moving on, computer language or programming language (you can call it any of the two), is a coded syntax used by computer programmers to communicate with a computer. Computer language establishes a flow of communication between software programs. The language enables a computer user to dictate what commands the computer must perform to process data. These languages can be classified into following categories

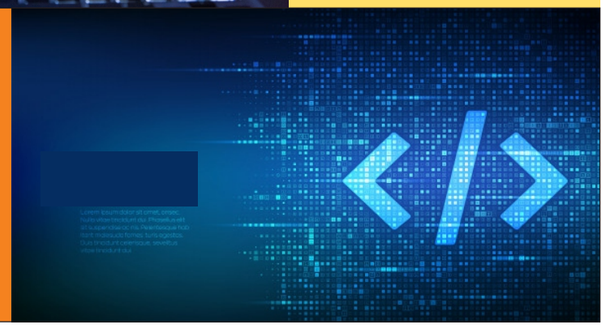
SAQ 3

8. Machine language



2. Assembly language

3. High level language



Machine Language

Let us talk about machine language briefly, Machine language or machine code is the native language directly understood by the computer's central processing unit or CPU. This type of computer language is not easy to understand, the reason is that it only uses a binary system, an element of notations containing only a series of numbers consisting of one and zero, to produce commands.

Disadvantages of Machine Language

Let us talk about some of the disadvantages of Machine Language.

- They are very bulky.
- They require much time for writing and reading.
- They are prone to error which is difficult to be detected and corrected.
- Very difficult to learn.
- Can only run on the computer it is designed for (machine dependent)

Assembly Level Language

Next to machine language, let us look at Assembly Language. Assembly Language uses MNEMONICS (symbols) to represent data and instructions. Such program eliminates problems associated with machine language. Let me point out that computer cannot execute directly a program written in assembly language, it requires a translator called assembler. Assembler is a special program designed to translate a program content in assembly language to a machine language equivalent.

Assembly Level Language is a set of codes that can run directly on the computer's processor. You should note that this type of language is most appropriate in writing operating systems and maintaining desktop applications. With the assembly level language, it is easier for a programmer to define commands. It is easier to understand and use as compared to machine language.

Disadvantages of Assembly Language

Let us consider some of the shortcoming of the assembly language.

- It is machine dependent; programmer has to be knowledgeable in both subject area and the operations of the machine.
- It is cumbersome though less cumbersome than that of machine language.
- Very expensive to develop and
- It consumes time

High Level Language

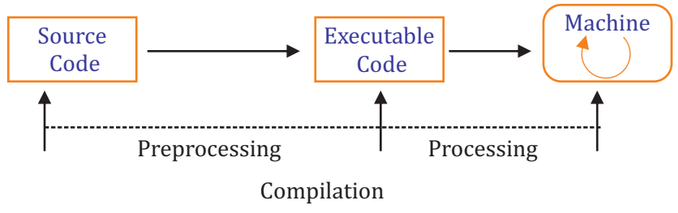
Let us discuss what we can consider to be the best of them all; High level languages. High Level Languages are user-friendly languages which are similar to English with vocabulary of words and symbols. These are easier to learn and require less time to write. They are problem oriented rather than 'machine' based. Another great thing we should note about program written in high level language is that it can be translated into many machine languages and therefore can run on any computer for which there exists an appropriate translator, isn't that excellent?

These are the programs that execute instructions written in a high-level language. There are two ways to run programs written in a high-level language. The most common is to compile the program; the other method is to pass the program through an interpreter. Let us talk on them briefly.

a. Compiler

A compiler is a special program that processes statements written in a particular programming language called as source code and converts them into machine language or “machine code” that a computer's processor uses.

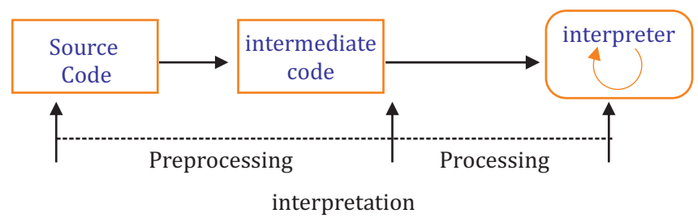
Compiler translates high level language programs directly into machine language program. We can refer to this process involved as *compilation*. (Fig 1.11)



b. Interpreter

Asides compilation, another method of running high-level programs is what we call Interpreter. Interpreter translates high-level instructions into an intermediate form, which it then executes.

If we would compare both methods; Compiled programs generally run faster than interpreted programs. The advantage of an interpreter, however, is that it does not need to go through the compilation stage during which machine instructions are generated. This process can be time-consuming if the program is long. (Fig 1.12)



Advantages of High-Level Language

Let us consider some of the reasons why High-Level Language is a better choice.

- The person writing the program does not need to know anything about the computer in which the program will be run (Machine Independent).
- The programs are portable.
- Very easy to learn and write

Features of High Level Language

- Machine independent
- Problem oriented
- Ability to reflect clearly the structure of program written in it.
- Readability
- Programs are portable.

Examples of High level Languages are FORTRAN, COBOL, QBASIC, VISUAL BASIC, JAVA, ADA, and PASCAL e.t.c.

Definition of Concepts in computer Program

I would briefly break down some of the concepts you should be familiar with when it comes to computer program.

SAQ 2

A program is a set of instructions written in a language (such as C programming) understandable by the computer to perform a particular function on the computer. A well written program could be packed well to form an application package customized for solving specific type of problem on the computer system.

SAQ 4

A computer programmer is computer scientist (a professional) skilled in using constructs of programming languages to develop executable and acceptable computer programs.

A software developer is a programmer. Programmers often work hand in hand with system analysts on large projects.

Programming languages are artificial notational languages created or developed to be used in preparing coded instructions on the computer for later execution by the computer. They are usually composed of series of usage rules (syntax) that determine the meaning (semantics) of expressions written in the

language. You should note that each programming language comes handy with its own translator i.e., interpreter or compiler as the case may be.

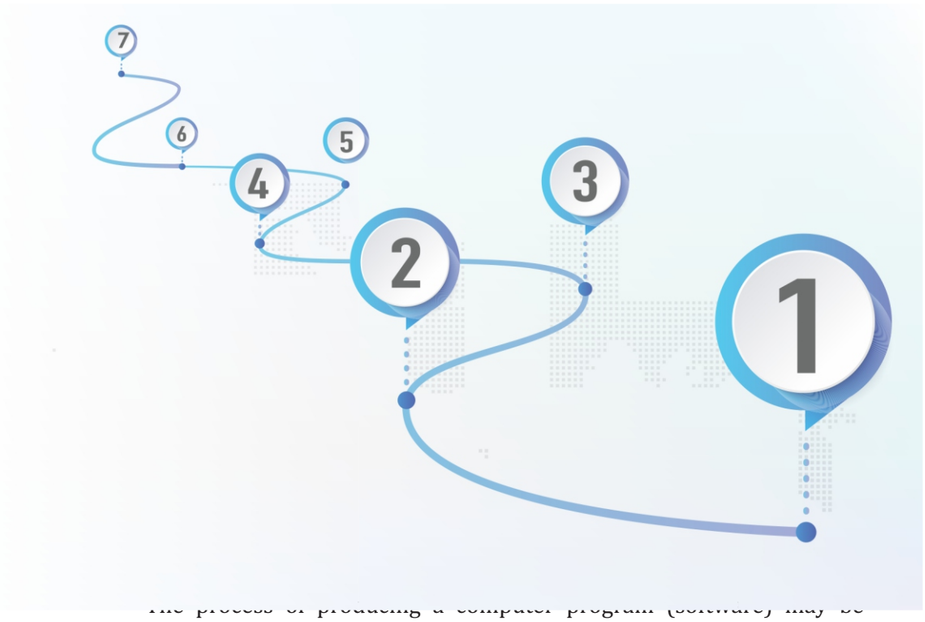
Programming is the art of developing computer programs with the aid of selected programming language by a computer programmer. It is a special skill whose quality is tested by the quality of the resulting program or software. In programming, we must properly follow programming stages, i.e., from problem definition to maintenance and review

Characteristics of a Good Program

Let us briefly point out what a good program should look like

- Transferability- Must be able to work on any computer machine.
- Reliability- It can be relied upon to do what it is expected to do.
- Efficiency / cost saving- It must not cost more than its benefits and enables problem to be solved appropriately, quickly and efficiently.
- Simplicity- It should be as simple as possible to understand.
- Understandability / Readability- It must be readable and understandable by other programmers and end users.
- Flexibility / Adaptability / Maintainability- A good program must be flexible adaptable and maintainable in order to suit user's need. Modification must be possible and very easy.

Phases of Program Development (Programming)



The process of producing a computer program (software) may be divided into eight phases or stages:

1. Problem definition / Analysis
2. Selection or development of an algorithm
3. Designing the program
4. Coding the programming statements
5. Compiling / Compilation stage
6. Testing / Running and Debugging the program
7. Documentation.

1. **Problem Definition / Analysis Stage:** in this stage, you need to understand the problem that requires a solution. We need to determine the data to be processed, from of the data, volume of the data, what to be done to the data to produce the expected / required output.

2. **Selection or development of an algorithm:** An algorithm is the set of steps required to solve a problem written down in English language.

3. **Designing the program:** In order to minimize the amount of time to be spent in developing the software, the programmer makes use of flowchart. Have it at the back of your mind that flowchart is the pictorial representation of the algorithm developed in step 2 above. Pseudocode IPO chart (input processing output) and Hipo chart (Hierachical- input-processing and output) may be used in place of flowchart or to supplement flowchart.

4. **Coding the statement:** This stage involves writing the program statements. The programmer uses the program flow chart as a guide for coding the steps the computer will follow.

5. **Compiling:** There is need to translate the program from the source code to the machine or object code if it is not written in machine language. A computer program is fed into the computer first, then as the source program is entered, a translated equivalent (object program) is created and stored in the memory.

6. **Running, Testing and Debugging:** When the computer is activated to run a program, it may find it difficult to run it because many syntax errors might have been committed. Manuals are used to debug the errors. A program that is error free is tested using some test data. If the program works as intended, real required data are then loaded.

7. **Documentation:** This is the last stage in software development. This involves keeping written records that describe the program, explain its purposes, define the amount, types and sources of input data required to run it. List the Departments and people who use its output and trace the logic the program follows.

Language Types

Machine and assembly languages

SAQ 5

We have earlier briefly talked about what machine language is. Furthermore, a machine language consists of the numeric codes for the operations that a particular computer can execute directly. The codes are strings of 0s and 1s, or binary digits ("bits"), which are frequently converted both from and to hexadecimal (base 16) for human viewing and modification. Machine language instructions typically use some bits to represent operations, such as addition, and some to represent operands, or perhaps the location of the next instruction. Don't forget as we mentioned earlier that Machine language is difficult to read and write, since it does not resemble conventional mathematical notation or human language, and its codes vary from computer to computer.

Assembly language is one level above machine language. It uses short mnemonic codes for instructions and allows the programmer to introduce names for blocks of memory that hold data. One might thus write "add pay, total" instead of "0110101100101000" for an instruction that adds two numbers.

May I let you know that Assembly language is designed to be easily translated into machine language? Although blocks of data may be referred to by name instead of by their machine addresses, assembly language does not provide more sophisticated means of organizing complex information. Like machine language, assembly language requires detailed knowledge of internal computer architecture. It is useful when such details are important, as in programming a computer to interact with input/output devices (printers, scanners, storage devices, and so forth).

Algorithmic languages

Let us discuss algorithmic languages for a while, Algorithmic languages are designed to express mathematical or symbolic computations. They can express algebraic operations in notation similar to mathematics and allow the use of subprograms that package commonly used operations for reuse. Bear in mind that algorithmic languages were the first high-level languages.

FORTRAN

It may interest you to know that the first important algorithmic language was FORTRAN (*for*mula *trans*lation), designed in 1957 by an IBM team led by John Backus. It was intended for scientific computations with real numbers and collections of them organized as one- or multidimensional arrays. Its control structures included conditional IF statements, repetitive loops (so-called DO loops), and a GOTO statement that allowed non-sequential execution of program code. FORTRAN made it convenient to have subprograms for common mathematical operations, and built libraries of them.

You should note that FORTRAN was also designed to translate into efficient machine language. It was immediately successful and continues to evolve.

ALGOL

You should note that ALGOL (*algo*rithmic *l*anguage) was designed by a committee of American and European computer scientists during 1958–60 for publishing algorithms, as well as for doing computations. Like LISP (we will describe this in the next section), ALGOL had recursive subprograms—procedures that could invoke themselves to solve a problem by reducing it to a smaller problem of the same kind. ALGOL introduced block structure, in which a program is composed of blocks that might contain both data and instructions and have the same structure as an entire program. Block structure became a powerful tool for building large programs out of small components.

You should note that ALGOL contributed a notation for describing the structure of a programming language, Backus–Naur Form, which in some variation became the standard tool for stating the syntax (grammar) of programming languages. ALGOL was widely used in Europe, and for many years it remained the language in which computer algorithms were published. Many important languages, such as Pascal and Ada (both described later), are its descendants.

LISP

I will like you to know that LISP (*lis*t *p*rocessing) was developed about 1960 by John McCarthy at the Massachusetts Institute of Technology (MIT) and was founded on the mathematical theory of recursive functions (in which a function appears in its own definition). A LISP program is a function applied to data, rather than being a sequence of procedural steps as in FORTRAN and ALGOL. LISP uses a very simple notation in which operations and their operands are given in a parenthesized list. Let us consider this example, (+ *a* (* *b c*)) stands for *a* + *b***c*. Although to you, this may appear awkward, the notation works well for computers. LISP also uses the list structure to represent data, and, because programs and data use the same structure, it is easy for a LISP program to operate on other programs as data.

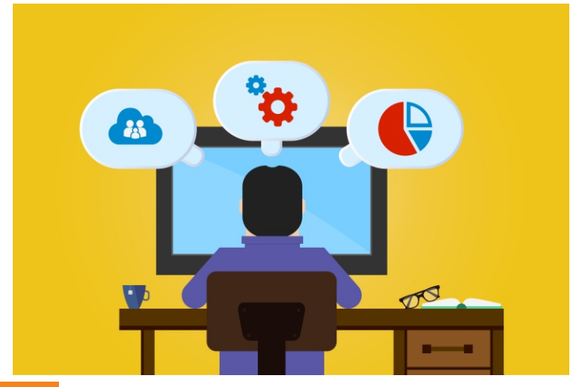
LISP became a common language for artificial intelligence (AI) programming, partly owing to the confluence of LISP and AI work at MIT and partly because AI programs capable of "learning" could be written in LISP as self-modifying programs. LISP has evolved through numerous dialects, such as Scheme and Common LISP.

C

You must have heard of the C programming language prior to this time, the C programming language was developed in 1972 by Dennis Ritchie and Brian Kernighan at the AT&T Corporation for programming computer operating systems. Its capacity to structure data and programs through the composition of smaller units is comparable to that of ALGOL. It uses a compact notation and provides the programmer with the ability to operate with the addresses of data as well as with their values. This ability is important in systems programming, and C shares with assembly language the power to exploit all the features of a computer's internal architecture. C, along with its descendant C++, remains one of the most common languages.

Business-oriented languages

We would discuss two commonly known languages under the business-oriented languages, which are COBOL and SQL. Let us get started on this.



COBOL

It may interest you to know that COBOL (*co*mmon *b*usiness-oriented *l*anguage) has been heavily used by businesses since its inception in 1959. A committee of computer manufacturers and users and U.S. government organizations established CODASYL (*Co*mmittee on *Da*ta *Sy*stems and *L*anguages) to develop and oversee the language standard in order to ensure its portability across diverse systems.

Let me mention that COBOL uses an English-like notation—novel when introduced. Business computations organize and manipulate large quantities of data, and COBOL introduced the record data structure for such tasks. A record clusters heterogeneous data such as a name, ID number, age, and address into a single unit. This contrasts with scientific languages, in which homogeneous arrays of numbers are common. Records are an important example of "chunking" data into a single object, and they appear in nearly all modern languages.

SQL

SQL is another business-oriented language I want you to learn. SQL (structured query language) is a language for specifying the organization of databases, that is collections of records. I want you to observe that Databases organized with SQL are called relational because SQL provides the ability to query a database for information that falls in a given relation. Let us take a look at this example, a query might be “find all records with both last_name Smith and city New York.” Commercial database programs commonly use a SQL-like language for their queries.

Education-oriented languages

So far, we have talked about a number of languages, another one we would consider is the education-oriented languages, we will briefly discuss four languages under this.

BASIC

The first language we want to look at under education-oriented language is BASIC. BASIC (beginner's all-purpose symbolic instruction code) was designed at Dartmouth College in the mid-1960s by John Kemeny and Thomas Kurtz. It was intended to be easy to learn by novices, particularly non-computer science majors, and to run well on a time-sharing computer with many users. It had simple data structures and notation and it was interpreted: a BASIC program was translated line-by-line and executed as it was translated, which made it easy to locate programming errors.

You should bear in mind that because of its small size and simplicity, BASIC became a popular language for early personal computers. Its recent forms have adopted many of the data and control structures of other contemporary languages, which makes it more powerful but less convenient for beginners.

Pascal

Sometime around 1970, Niklaus Wirth of Switzerland designed Pascal to teach structured programming, which emphasized the orderly use of conditional and loop control structures without GOTO statements. Although Pascal resembled ALGOL in notation, it provided the ability to define data types with which to organize complex information, a feature beyond the capabilities of ALGOL as well as FORTRAN and COBOL. User-defined data types allowed the programmer to introduce names for complex data, which the language translator could then check for correct usage before running a program.

During the late 1970s and '80s, Pascal was one of the most widely used languages for programming instruction. It was available on nearly all computers, and, because of its familiarity, clarity, and security, it was used for production software as well as for education.

Logo

Logo originated in the late 1960s as a simplified LISP dialect for education; Seymour Papert and others used it at MIT to teach mathematical thinking to schoolchildren. It had a more conventional syntax than LISP and featured “turtle graphics,” a simple method for generating computer graphics. (The name came from an early project to program a turtlelike robot.) Turtle graphics used body-centred instructions, in which an object was moved around a screen by commands, such as “left 90” and “forward,” that specified actions relative to the current position and orientation of the object rather than in terms of a fixed framework. Together with recursive routines, this technique made it easy to program intricate and attractive patterns.

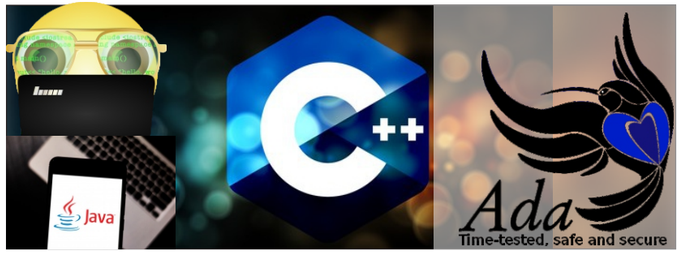
Hypertalk

It will interest you know that Hypertalk was designed as “programming for the rest of us” by Bill Atkinson for Apple's Macintosh. Using a simple English-like syntax, Hypertalk enabled anyone to combine text, graphics, and audio quickly into “linked stacks” that could be navigated by clicking with a mouse on standard buttons supplied by the program. Hypertalk was particularly popular among educators in the 1980s and early '90s for classroom multimedia presentations. Although Hypertalk had many features of object-oriented languages (described in the next section), Apple did not develop it for other computer platforms and let it languish; as Apple's market share declined in the 1990s, a new cross-platform way of displaying multimedia left Hypertalk all but obsolete (*see* the section World Wide Web display languages).

object-oriented languages

We will observe that Object-oriented languages help to manage complexity in large programs. Let me explain that; Objects package data and the operations on them so that only the operations are publicly accessible and internal details of the data structures are hidden. This information hiding made large-scale programming easier by allowing a programmer to think about each part of the program in isolation. Let me also add that objects may be derived from more general ones, “inheriting” their capabilities. Such an object hierarchy made it possible to define specialized objects without repeating all that is in the more general ones.

You should pay attention to the fact that Object-oriented programming began with the Simula language (1967), which added information hiding to ALGOL. Another influential object-oriented language was Smalltalk (1980), in which a program was a set of objects that interacted by sending messages to one another.



C++

Here is another commonly used language you must have heard. The C++ language, developed by Bjarne Stroustrup at AT&T in the mid-1980s, extended C (you remember we talked about this earlier) by adding objects to it while preserving the efficiency of C programs. It has been one of the most important languages for both education and industrial programming. It should interest you to know that large parts of many operating systems, such as the Microsoft Corporation's Windows 98, were written in C++.

Ada

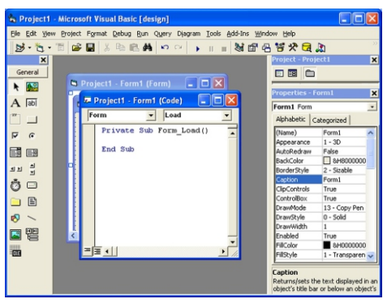
Does the name of this language sound like someone's name to you? Well, Ada was named for Augusta Ada King, countess of Lovelace, who was an assistant to the 19th-century English inventor Charles Babbage, and is sometimes called the first computer programmer. Ada, the language, was developed in the early 1980s for the U.S. Department of Defense for large-scale programming. It combined Pascal-like notation with the ability to package operations and data into independent modules. Its first form, Ada 83, was not fully object-oriented, but the subsequent Ada 95 provided objects and the ability to construct hierarchies of them. You should note that even though Ada is no longer mandated for use in work for the Department of Defense, Ada remains an effective language for engineering large programs.

Java

Like C, C++, Java is another commonly used and familiar language you must have heard or even attempted before. Let us do a little background on Java, in the early 1990s, Java was designed by Sun Microsystems, Inc., as a programming language for the World Wide Web (WWW). Although it resembled C++ in appearance, it was fully object-oriented. In particular, Java dispensed with lower-level features, including the ability to manipulate data addresses, a capability that is neither desirable nor useful in programs for distributed systems. In order to be portable, Java programs are translated by a Java Virtual Machine specific to each computer platform, which then executes the Java program. Let me also mention that to adding interactive capabilities to the Internet through Web "applets," Java has been widely used for programming small and portable devices, such as your mobile telephones.

Visual Basic

The last one we will look into here under object-oriented language is Visual Basic. Visual Basic was developed by Microsoft to extend the capabilities of BASIC by adding objects and "event-driven" programming: buttons, menus, and other elements of graphical user interfaces (GUIs). Visual Basic can also be used within other Microsoft software to program small routines.



Declarative languages

Next language we will look into are the Declarative languages. Declarative languages, also called nonprocedural or very high level, are programming languages in which (ideally) a program specifies what is to be done rather than how to do it. You should note that in these languages there is less difference between the specification of a program and its implementation than in the procedural languages described so far. The two common kinds of declarative languages are logic and functional languages.

Let us talk briefly on logic programming languages. You should note that in Logic programming languages, PROLOG (*pro*gramming in *log*ic) is the best known; this programming language states a program as a set of logical relations (e.g., a grandparent is the parent of a parent of someone). Such languages are similar to the SQL database language. Let us break this down, a program is executed by an "inference engine" that answers a query by searching these relations systematically to make inferences that will answer a query. PROLOG has been used extensively in natural language processing and other AI programs.

I want you to note that functional languages have a mathematical style. A functional program is constructed by applying functions to arguments. Functional languages, such as LISP, ML, and Haskell, are used as research tools in language development, in automated mathematical theorem provers, and in some commercial projects.

Scripting languages

Do you know that Scripting languages are sometimes called little languages? This is because they are intended to solve relatively small programming problems that do not require the overhead of data declarations and other features needed to make large programs manageable. Let me explain further, scripting languages are used for writing operating system utilities, for special-purpose file-manipulation programs, and, because they are easy to learn, sometimes for considerably larger programs.

PERL (practical extraction and report language) was developed in the late 1980s, originally for use with the UNIX operating system. You should take note of the fact that PERL was intended to have all the capabilities of earlier scripting languages. PERL provided many ways to state common operations and thereby allowed a programmer to adopt any convenient style. In the 1990s it became popular as a system-programming tool, both for small utility programs and for prototypes of larger ones. Together with other languages discussed below, it also became popular for programming computer Web "servers."

Document formatting languages

Just as we see the name implies; document formatting languages specify the organization of printed text and graphics. They fall into several classes: text formatting notation that can serve the same functions as a word processing program, page description languages that are interpreted by a printing device, and, most generally, markup languages that describe the intended function of portions of a document. Let us further discuss them one after the other briefly.

TeX

The first we want to talk about here is TeX. TeX was developed during 1977–86 as a text formatting language by Donald Knuth, a Stanford University professor, to improve the quality of mathematical notation in his books. Text formatting systems, unlike WYSIWYG ("What You See Is What You Get") word processors, embed plain text formatting commands in a document, which are then interpreted by the language processor to produce a formatted document for display or printing. TeX marks italic text, take a look at this example carefully, as {\it this is italicized}, it is then displayed as *this is italicized*.

It may interest you know that TeX largely replaced earlier text formatting languages. Its powerful and flexible abilities gave an expert precise control over such things as the choice of fonts, layout of tables, mathematical notation, and the inclusion of graphics within a document. It is generally used with the aid of "macro" packages that define simple commands for common operations, such as starting a new paragraph; LaTeX is a widely used package. TeX contains numerous standard "style sheets" for different types of documents, and these may be further adapted by each user. There are also related programs such as BibTeX, which manages bibliographies and has style sheets for all of the common bibliography styles, and versions of TeX for languages with various alphabets.

PostScript

I will like you to learn that PostScript is a page-description language developed in the early 1980s by Adobe Systems Incorporated on the basis of work at Xerox PARC (Palo Alto Research Center). Such languages describe documents in terms that can be interpreted by a personal computer to display the document on its screen or by a microprocessor in a printer or a typesetting device.

PostScript commands can, for example, precisely position text, in various fonts and sizes, draw images that are mathematically described, and specify colour or shading. PostScript uses postfix, also called reverse Polish notation, in which

an operation name follows its arguments. Thus, “300 600 20 270 arc stroke” means: draw (“stroke”) a 270-degree arc with radius 20 at location (300, 600). Although PostScript can be read and written by a programmer, it is normally produced by text formatting programs, word processors, or graphic display tools.

You should take note of this, the success of PostScript is due to its specification's being in the public domain and to its being a good match for high-resolution laser printers. It has influenced the development of printing fonts, and manufacturers produce a large variety of PostScript fonts.

SGML

Let us start from knowing and noting that SGML (standard generalized markup language) is an international standard for the definition of markup languages; that is, it is a meta language. Markup consists of notations called tags that specify the function of a piece of text or how it is to be displayed. SGML emphasizes descriptive markup, in which a tag might be “<emphasis>.” Such a markup denotes the document function, and it could be interpreted as reverse video on a computer screen, underlining by a typewriter, or italics in typeset text.

Furthermore, SGML is used to specify DTDs (document type definitions). A DTD defines a kind of document, such as a report, by specifying what elements must appear in the document—e.g., <Title>—and giving rules for the use of document elements, such as that a paragraph may appear within a table entry but a table may not appear within a paragraph. A marked-up text may be analyzed by a parsing program to determine if it conforms to a DTD. Another program may read the markups to prepare an index or to translate the document into PostScript for printing. Yet another might generate large type or audio for readers with visual or hearing disabilities.

World Wide Web display languages

The World Wide Web is commonly identified as www, does that make it more relatable to you? The word Wide Web is a system for displaying text, graphics, and audio retrieved over the Internet on a computer monitor. Each retrieval unit is known as a Web page, and such pages frequently contain “links” that allow related pages to be retrieved.

HTML

I will like you to note what HTML is all about. HTML (*hyper*t*ext* *m*arkup *l*anguage) is the markup language for encoding Web pages. Le us do a background study of HTML, It was designed by Tim Berners-Lee at the CERN nuclear physics laboratory in Switzerland during the 1980s and is defined by an SGML DTD. HTML markup tags specify document elements such as headings, paragraphs, and tables. They mark up a document for display by a computer program known as a Web browser. The browser interprets the tags, displaying the headings, paragraphs, and tables in a layout that is adapted to the screen size and fonts available to it.

You should also take note that HTML documents contain anchors, which are tags that specify links to other Web pages. An anchor has the form <A HREF= “http://www.britannica.com”> Encyclopædia Britannica</A>, where the quoted string is the URL (universal resource locator) to which the link points (the Web “address”) and the text following it is what appears in a Web browser, underlined to show that it is a link to another page. What is displayed as a single page may also be formed from multiple URLs, some containing text and others graphics.

XML

We earlier discussed HTML, let me at this point mention that HTML does not allow one to define new text elements; that is, it is not extensible. Unlike HTML, XML (extensible markup language) is a simplified form of SGML intended for documents that are published on the Web. Like SGML, XML uses DTDs to define document types and the meanings of tags used in them. XML adopts conventions that make it easy to parse, such as that document entities are marked by both a beginning and an ending tag, such as <BEGIN>...</BEGIN>. XML provides more kinds of hypertext links than HTML, such as bidirectional links and links relative to a document subsection.

Because an author may define new tags, an XML DTD must also contain rules that instruct a Web browser how to interpret them—how an entity is to be displayed or how it is to generate an action such as preparing an e-mail message.

Web Scripting

You should take note that web pages marked up with HTML or XML are largely static documents. I want you to understand that web scripting can add information to a page as a reader uses it or let the reader enter information that may, for example, be passed on to the order department of an online business. CGI (common gateway interface) provides one mechanism; it transmits requests and responses between the reader's Web browser and the Web server that provides the page.

The CGI component on the server contains small programs called scripts that take information from the browser system or provide it for display. A simple script might ask the reader's name, determine the Internet address of the system that the reader uses, and print a greeting. Scripts may be written in any programming language, but, because they are generally simple text-processing routines, scripting languages like PERL are particularly appropriate.

Another approach is to use a language designed for Web scripts to be executed by the browser. JavaScript is one such language, designed by the Netscape Communications Corp., which may be used with both Netscape's and Microsoft's browsers. JavaScript is a simple language, quite different from Java. A JavaScript program may be embedded in a Web page with the HTML tag <script language="JavaScript">. JavaScript instructions following that tag will be executed by the browser when the page is selected. In order to speed up display of dynamic (interactive) pages, JavaScript is often combined with XML or some other language for exchanging information between the server and the client's browser. In particular, the XML Http Request command enables asynchronous data requests from the server without requiring the server to resend the entire Web page. This approach, or "philosophy," of programming is called Ajax (*a*synchronous *J*avaScript *a*nd *X*ML).

VB Script is a subset of Visual Basic. Originally developed for Microsoft's Office suite of programs, it was later used for Web scripting as well. Its capabilities are similar to those of JavaScript, and it may be embedded in HTML in the same fashion.

Behind the use of such scripting languages for Web programming lies the idea of component programming, in which programs are constructed by combining independent previously written components without any further language processing. JavaScript and VB Script programs were designed as components that may be attached to Web browsers to control how they display information.

Summary

Since we have learnt a lot in this unit, let us outline some of the specific points:

- A program is a set of instructions following the rules of the chosen language.
- A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.
- Machine and assembly languages are “low-level,” requiring a programmer to manage explicitly all of a computer's characteristic features of data storage and operation.
- Machine language or machine code is the native language directly understood by the computer's central processing unit or CPU.
- Transferability, Reliability, Efficiency, Simplicity, Understandability / Readability and Flexibility / Adaptability / Maintainability are characteristic of good program
- Problem definition / Analysis, Selection or development of an algorithm, Designing the program, Coding the programming statements, Compiling / Compilation stage, Testing / Running and Debugging the program, Documentation and Maintenance are Phases of Program Development (Programming)

Self-Assessment

SAQ

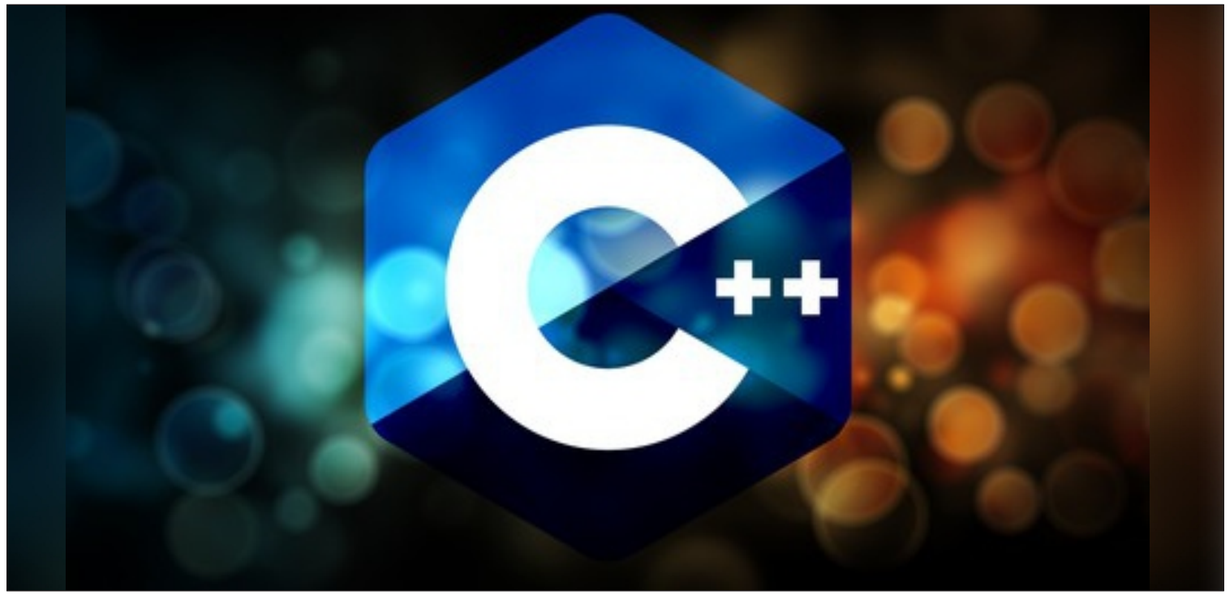
- What is a computer programming language?
- Define a program
- Identify three (3) types of computer programming language
- Briefly explain the concepts in computer Program.
- What is Language Types.

Tutor Marked Assessment

- Define a program
- List the stages in program development
- State the characteristics of a Good programming Language

Further Reading

- https://www.coursera.org › learn › introduction-to-computer-programming
- https://www.researchgate.net › publication › 317182495_INTRODUCTION_...
- https://www.freetechbooks.com › introduction-to-programming-f105
- https://www.pearsonhighered.com › assets › samplechapter



UNIT 2

History and Fundamentals of C++

Introduction

Welcome to another unit of this course, we would be looking at the concept of programming using C++ language. We will discuss in details the history, features/characteristics, and compare the two foundation modern programming languages i.e C Vs C++. We will also state the similarities between C and C++ and detailed differences between C and C++. In the end, we will highlight the supported paradigms of C++ programming language.

Learning Outcomes

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

1. Give account of the historical background of C++ language
2. List at least 5 similarities of C and C++ language
3. State four advantages of C++ over C language
4. Highlight the feature of C++

Main Content

History and Fundamentals of C++

1 min

It will interest you to know that the C++ programming language has a history going as far back as 1979, when Bjarne Stroustrup was doing work for his Ph.D. thesis. One of the languages Stroustrup had the opportunity to work with was a language called Simula, which as the name implies is a language primarily designed for simulations. The Simula 67 language - which was the variant that Stroustrup worked with - is regarded as the first language to support the object-oriented programming paradigm. Stroustrup found that this paradigm was very useful for software development, however the Simula language was far too slow for practical use.

You should also note that not long after that time, he began work on "C with Classes", which as the name implies was meant to be a superset of the C language. His goal was to add object-oriented programming into the C language, which was and still is a language well-respected for its portability without sacrificing speed or low-level functionality. His language included classes,

SAQ 1

basic inheritance, inlining, default function arguments, and strong type checking in addition to all the features of the C language.

The first C with Classes compiler was called Cfront, which was derived from a C compiler called CPre. It was a program designed to translate C with Classes code to ordinary C. let me mention an interesting point about this, the Cfront was written mostly in C with Classes, making it a self-hosting compiler (a compiler that can compile itself). Cfront would later be abandoned in 1993 after it became difficult to integrate new features into it, namely C++ exceptions. Nonetheless, Cfront made a huge impact on the implementations of future compilers and on the Unix operating system.

Let us go further in our background study, in 1983, the name of the language was changed from C with Classes to C++. The ++ operator in the C language is an operator for incrementing a variable, which gives some insight into how

Stroustrup regarded the language. Many new features were added around this time, the most notable of which are virtual functions, function overloading, references with the & symbol, the const keyword, and single-line comments using two forward slashes (which is a feature taken from the language BCPL).

In 1985, Stroustrup's reference to the language entitled The C++ Programming Language was published. That same year, C++ was implemented as a commercial product. The language was not officially standardized yet, making the book a very important reference. The language was updated again in 1989 to include protected and static members, as well as inheritance from several classes.

In 1990, The Annotated C++ Reference Manual was released. The same year, Borland's Turbo C++ compiler would be released as a commercial product. Turbo C++ added a plethora of additional libraries which would have a considerable impact on C++'s development. Although Turbo C++'s last stable release was in 2006, the compiler is still widely used.

In 1998, the C++ standards committee published the first international standard for C++ ISO/IEC 14882:1998, which would be informally known as C++98. The Annotated C++ Reference Manual was said to be a large influence in the development of the standard. The Standard Template Library, which began its conceptual development in 1979, was also included. In 2003, the committee responded to multiple problems that were reported with their 1998 standard, and revised it accordingly. The changed language was dubbed C++03.

In 2005, the C++ standards committee released a technical report (dubbed TR1) detailing various features they were planning to add to the latest C++ standard. The new standard was informally dubbed C++0x as it was expected

to be released sometime before the end of the first decade. Ironically, however, the new standard would not be released until mid-2011. Several technical reports were released up until then, and some compilers began adding experimental support for the new features.

In mid-2011, the new C++ standard (dubbed C++11) was finished. The Boost library project made a considerable impact on the new standard, and some of the new modules were derived directly from the corresponding Boost libraries. Some of the new features included regular expression support (details on regular expressions may be found here), a comprehensive randomization library, a new C++ time library, atomics support, a standard threading library (which up until 2011 both C and C++ were lacking), a new for loop syntax providing functionality similar to foreach loops in certain other languages, the auto keyword, new container classes, better support for unions and array-initialization lists, and variadic templates.

SAQ

Now that we have discussed about the background study of C++, briefly explain historical background of C++ language.

C vs. C++

1 min

What is C programming language?

I want you to note that C is middle-level programming language which was developed at Bell Lab in 1972 by Dennis Ritchie. C language combines the features of Low level as well as High-level Language. Hence its considered a middle-level Language.

You should also know that C is a high-level classical type programming language that allows you to develop firmware and portable applications. The C

language was developed with an objective of writing system software. It is an ideal language for developing firmware systems.

Let us point out the difference here, C is a system programming language, whereas C++ is a general-purpose programming language commonly used in **embedded systems.** C is procedural, so it doesn't support classes and objects like C++ does (although, despite being object-oriented, C++ can be procedural like C, making it a bit more hybrid).

What is C++ programming language?

Let us discuss the C++ extensively here, C++ is a computer programming language that contains the feature of C programming language as well as Simula67 (a first object Oriented language). C++ introduced the concept of Class and Objects.

C++ is everything C is, and more. It may interest you to know that C++ is not new, either, and has itself been the inspiration for many languages that have come behind it like Python, Perl, and **PHP**. It does however add in a few modern elements that make it a step up from C.

You should note that C++ encapsulates high and low-level language features. So, it is seen as an intermediate level language. It would interest you to know that C++ was earlier called "C with classes" as it had all the properties of the C language.

Comparing two Foundations of Modern Programming

Let us compare the programming languages C and C++, it would look a bit like comparing a traditional typewriter with an electric typewriter. That's because C++ is a direct descendent of C, the “grandfather” of many modern programming languages, just with more under the hood. C++ boasts better efficiency and productivity; you should note however, with more bells and whistles comes more responsibility.

As we have earlier mentioned, C is generally considered to be the foundation of many modern high-level programming languages like C# and Java. C++ language is one of those—an enhanced version of the language that adds an object-oriented layer, which definitely boosts developer speed and productivity. C++ is also one of the foundation languages for the MongoDB database and the Apache HTTP server.

For us to get a better understanding of how C++ builds upon C and adds new features, here's a look at these two closely related programming languages.

Generally, you'd opt to use C over C++ if you didn't want the extra overhead of C++—however you can always just pick the features of C++ you want to use and exclude the others.

Similarities between C and C++ are:

Since we have mentioned the relationship between C and C++, let us point out in specifics what they have in common

(i) Both the languages have the same syntax.

(ii) Code structure of both the languages are same.

(iii) the compilation of both the languages is about the same (not 100% sure, but it should be given that syntax & grammar are similar)

(iv) They share the same basic syntax. Nearly all of C's operators and keywords are also present in C++, and do the same thing.

(v) grammar, although C++ has a slightly extended grammar than Ruby that is all the OOP features introduce a few new operators but they both are similar else.

(vi) Basic memory model of both is very close to the hardware.

(vii) Same notions of stack, heap, file-scope and static variables are present in both the languages.

You see, for a C++ developer to know the language, they'll also know C—and quite a bit more, which can make it difficult to learn. C++ was created in the 1980s and has been used in the creation of desktop and web applications, although it's most popular for applications such as games, operating systems, and low-level hardware programming for a PC or server.

- C++ is directly derived from the C language. What this should mean to you is that it shares some properties with C while also adding some improvements.
- C++ is object-oriented. This translates to productivity and organization of code, which is a boon for more complex applications. It's great for fast applications and server-side software.
- C++ is lightweight and compiled. You should understand from this that before a C++ application is launched on a PC or the server, the code is converted into a binary file, or an executable .EXE file. C++ compiled files are pretty lightweight vs. files with more overhead, like C#. With C++, you can code for any platform including Mac, Windows and Linux.
- It has benefits of both high-level and low-level programming languages. This makes it more of a mid-level language.
- The power of C++ lies in its performance and speed. This makes it ideal for complex, large applications that require a lot of speed at scale. It's super efficient where higher level languages might not be as efficient, making it a better solution for applications where performance is important. We'll get more into some of the features that enable this below, but this is a big win for the language.
- C++ plays well with other languages. Because it can interface with nearly any other language, C++ is a great option. And, almost any system can

compile and run C++ code.

- Pointers equal productivity. A “pointer” is a feature of C++ (and other C-based languages) that allows developers to simplify code. A pointer represents an “address” where a piece of data exists, so you code the *location* of a variable, not the whole variable. Think of it this way: Instead of personally handing out newsletters to everyone in your company, you put the newsletters in a mailbox and tell everyone where the mailbox is located. Or, if you're dealing with a large bit of data, think of a pointer like giving someone your address, rather than giving them your whole house. It's a logic for computing—one we use every day as humans.

SAQ 1

Enumerate four (4) similarities of C and C++ language

Main Differences:

I want you to understand that C++ can be said a superset of C. Mainly added features in C++ are Object Oriented Programming, Exception Handling and Richer Library.

You should also note that the major difference between C and C++ is that C is a procedural programming language and does not support classes and objects, while C++ is a combination of both procedural and object oriented programming language; therefore C++ can be called a hybrid language. The following table presents differences between C and C++ in detail.

Detailed Differences between C and C++ are:

Basis of distinction	C	C++
Programming type	It is a Procedural Oriented language.	It is an Object Oriented Programming language.
Approach	C language follows Top Down programming approach	C++ follow bottom up programming approach.
File extension	The file extension of a C program is .c	The file extension of a c+ + program language is.cpp
Program division	In C programming language, a big program code is divided into small pieces which is called functions.	In C++ programming language, a big program code is divided into Objects and Classes.
Structure	Structure in C not provide the feature of function declaration.	Structure in C++ provides the feature of declaring a function as a member function of the structure.
Inline function	It does not allow inline function.	It supports inline function.
Standard I/O operations	In C scan and print f are used for the standard input and output	In C++ cin» and cout« are given for standard input and output operations.
Data Security	In C language the data is not secured.	Data is secure, so it can't be accessed by external functions. (Using Encapsulation concept of OOPs)

Detailed Differences between C and C++ are:

Basis of distinction	C	C++
Ease of Coding	C is an older programming language that is described as Hands on. In this language, you must tell the program to do everything. Moreover, this language will let you do almost anything.	C++ is an extension language of C. It allows for the highly controlled object oriented code.
Compatibility with other languages	C is not compatible with another language.	C++ is compatible with the other generic programming languages.
Pointer	C supports only Pointers.	C++ supports both pointers and references.
Variable	In C, the variable should be defined at the beginning of the program.	C++ allows you to declare variables anywhere in the function.
Point of Focus	C focuses on the steps or procedures that are followed to solve a problem.	C++ emphasizes the objects and not the steps or procedures. It has higher abstraction level.
Function Overloading	C does not allow you to use function overloading.	C++ allows you to use function overloading.
Data Types	C language does not allows you to declare String or Boolean data types. It supports built in and primitive data types.	C++ supports String and Boolean data types.

Detailed Differences between C and C++ are:

Basis of distinction	C	C++
Exception Handling	C does not support Exception Handling. However, it can be performed using some work arounds.	C++ supports Exception handling. Moreover, this operation can be performed using try and catch block.
Functions	Does not allows functions with default arrangements	Allow functions with default arrangements.
Namespace	It is absent in C language.	It is present in the C++ language.
Source Code	Free format program source code.	Originally developed from the C programming language.
Relationship	C is a subset of C++. It cannot run C++ code.	C++ is a superset of C. C++ can run most of C code while C cannot run C++code.
Driven by	Function driven language	Object-driven language
Focus	Focuses on method or process instead of data.	Focuses on data instead of method or procedure.
Encapsulation	Does not support encapsulation. As Data and functions are separate and free entities.	Supports encapsulation. Data and functions are encapsulated together as an object.
Information hiding	C does not support information hiding. In this language, data are free entities and can be changed outside code.	Encapsulation hides the data. So that data structures and operators are used as per intention.

Detailed Differences between C and C++ are:

Basis of distinction	C	C++
Memory management	C provide malloc() and calloc() functions for dynamic memory allocation.	C++provides a new operator for this
Data Types	Supports built -in data types.	Supports built- in & user -defined data types.
Global Variables	Allows Multiple Declaration of global variables.	Multiple Declaration of global variables are not allowed.
Concept of Mapping	The mapping between Data and Function is very complicated.	The mapping between Data and Function can be easily established using "Classes and Objects."
Inheritance	Inheritance is not supported C	Inheritance is possible in C++ language.
Default header file	C used stdio.h header file.	C++ uses iosteam.h as default header file.
Virtual function	The concept of virtual Functions are present in C.	The concept of virtual Function is not used in C++.
Keywords	Contain 32 keywords.	Contains 52keywords.
Polymorphism	In C. Polymorphism is not possible	The concept of polymorphism is used inC++. Polymorphism is one of the most Important Features of OOPS.
GUI programming	C language offers GTK tool for GUI programming	C++ supports Qt tools for GUI programming

When should you use which language?

Let me tell you that for the most part, a developer will be able to guide you into which scenarios it is most appropriate to use one language over the other, and there's often healthy discussion over which excels in which situations.

If you want an application that works directly with computer hardware or deals with application development, C++ is a good option. C++ programs include server-side applications, networking, gaming, and even device drivers for your PC. However, if you need to code truly tiny systems, using C will result in a little less overhead than C++.

You should know that C++ is well-rounded in terms of platforms and target applications, so if your project is focused on extremely low-level processing, then you may want to use C++. One other instance you may consider C over C++ is when you need an application to be incredibly stable, and removing the abstractions of C++ can ensure air-tight code and control over every aspect. Also, if you don't have a C++ compiler on your platform, that's another common reason to go with C.

Type System

A type system refers to the rules that the different types of variables of a language have to follow. Some languages (including most assembly languages) do not have types and thus this section does not apply to them. However, as most languages (including C++) have types, this information is important.

Type Strength: Strong or Weak

Let us consider here a strong typing system and a weak typing system; you should note that a strong typing system puts restrictions on how different types of variables can be converted to each other without any converting statements. An ideal strong typing system would forbid implicit "casts" to types that do not make any sense, such as an integer to a Fruit object. A weak typing system would try to find some way to make the cast work.

Type Expression: Manifest or Inferred

We would understand here how the compiler/interpreter for a language infers the types of variables. Many languages require variables' types to be explicitly defined, and thus rely on manifest typing. Some however, will infer the type of the variable based on the contexts in which it is used, and thus use inferred typing.

Type Checking: Static or Dynamic

You would notice that if a language is statically typed, then the compiler/interpreter does the type checking once before the program runs/is compiled. On the other hand, if the language is dynamically type checked, then the types are checked at run-time.

Type Safety: Safe or Unsafe

I want you to know that type safety refers to the degree to which a language will prohibit operations on typed variables that might lead to undefined behavior or errors. A safe language will do more to ensure that such operations or conversions do not occur, while an unsafe language will give more responsibility to the user in this regard. These typing characteristics are not necessarily mutually exclusive, and some languages mix them.

Supported paradigms

I want you to understand that a programming paradigm is a methodology or way of programming that a programming language supports. Here is a summary of a few common paradigms:

Declarative

I want you to know that a declarative language will focus more on specifying what a language is supposed to accomplish rather than by what means it is supposed to accomplish it. Such a paradigm might be used to avoid undesired side-effects resulting from having to write one's own code.

Functional

I will like you to note that functional programming is a subset of declarative programming that tries to express problems in terms of mathematical equations and functions. It goes out of its way to avoid the concepts of states and mutable variables which are common in imperative languages.

Generic

I want you to pay attention to the fact that generic programming focuses on writing skeleton algorithms in terms of types that will be specified when the algorithm is actually used, thus allowing some leniency to programmers who wish to avoid strict strong typing rules. You should note that It can be a very powerful paradigm if well-implemented.

Imperative

I want you to know that the peculiarity of imperative languages unlike declarative programming is that it allow programmers to give the computer ordered lists of instructions without necessarily having to explicitly state the task. It can be thought of being the opposite of declarative programming.

Structured

just as the name implies, you need to understand that structured programming languages aim to provide some form of noteworthy structure to a language, such as intuitive control over the order in which statements are executed (if X then do Y otherwise do Z, do X while Y is Z). Such languages generally deprecate "jumps", such as those provided by the go to statement in C and C++.

Procedural

it is important for you to note that although this programming language is sometimes used as a synonym for imperative programming, a procedural programming language can also refer to an imperative structured programming language which supports the concept of procedures and subroutines (also known as functions in C or C++).

Object-Oriented

Object-Oriented programming (sometimes abbreviated to OOP) is a subset of structured programming which expresses programs in the terms of "objects", which are meant to model objects in the real world. Such a paradigm allows code to be reused in remarkable ways and is meant to be easy to understand.

Standardization

Think about this for a while; does a language have a formal standard? This can be very important to ensure that programs written to work with one compiler/interpreter will work with another. It may interest you to know that some languages are standardized by the American National Standards Institute (ANSI), some are standardized by the International Organization for Standardization (ISO), and some have an informal but de-facto standard not maintained by any standards organization.

Summary

Let us highlight some of the lessons we have learnt in this unit

- The first C with Classes compiler was called Cfront, which was derived from a C compiler called CPre. It was a program designed to translate C with Classes code to ordinary C.
- In 1983, the name of the language was changed from C with Classes to C++. The ++ operator in the C language is an operator for incrementing a variable, which gives some insight into how Stroustrup regarded the language.
- C is a system programming language, whereas C++ is a general-purpose programming language commonly used in embedded systems.
- C++ is everything C is, and more. It's not new, either, and has itself been the inspiration for many languages that have come behind it like Python, Perl, and PHP. It does however add in a few modern elements that make it a step up from C.
- The major difference between C and C++ is that C is a procedural programming language and does not support classes and objects, while

C++ is a combination of both procedural and object oriented programming language; therefore C++ can be called a hybrid language.

Tutor Marked Assessment

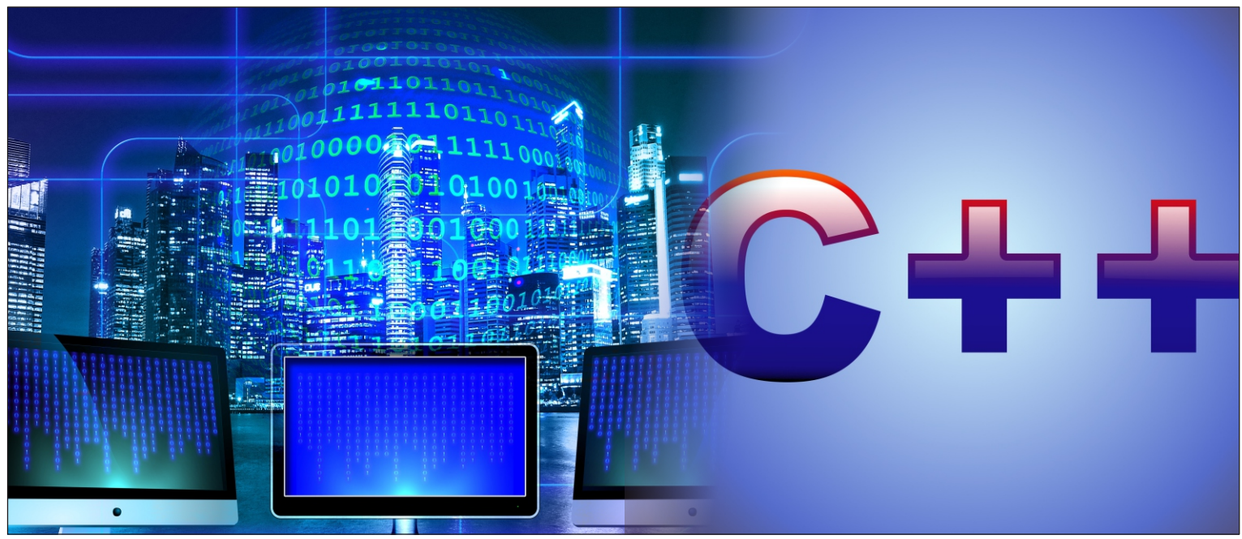
- List 5 feature of C++ programming language
- State five different between C and C++ programming language
- What is the main different between C and C++ programming language

Further Reading

- https://en.wikibooks.org › wiki › Visual_Basic › History
- https://www.ntu.edu.sg › home › ehchua › programming › cpp › cp1_Basics
- https://python.cs.southern.edu › cppbook › progcpp
- https://www.learncpp.com

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition:
  Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.



UNIT 3

Compiled Languages and C++

Introduction

You are welcome to another unit, I hope you have enjoyed the previous unit so far. In this unit, I will be taking you through the compilation process in which a program goes from text file (or source files) to processor instruction, fundamental components that make up the C++ language. I will bring to your knowledge concepts like tokens, types of token, purpose with example. I will also explain to you character set, comments and identifiers, keywords, datatypes, variables and constants that form the vocabulary of C++ language. Let me mention that these entire concepts are very significant as they form the solid foundations upon which all other concepts are made.

Learning Outcomes

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

1. Identify the elements of the two categories of character set used in C++
2. Enumerate rules for constructing user defined identifiers in C++
3. State the fundamental datatypes in C++ with their sizes in byte.
4. List at least the keywords in C++
5. Declare, Initialize and assign appropriate values to variables

Main Content

Why Use a Language Like C++?

10 min

You should bear in mind that at its core, a computer is just a processor with some memory, capable of running tiny instructions like "store 5 in memory location 23459." If you are curious, you may start to ask, why would we express a program as a text file in a programming language, instead of writing processor instructions?

The advantages:

1. **Conciseness:** programming languages allow us to express common sequences of commands more concisely. C++ provides some especially powerful shorthands.

2. **Maintainability:** modifying code is easier when it entails just a few text edits, instead of rearranging hundreds of processor instructions. C++ is object oriented.

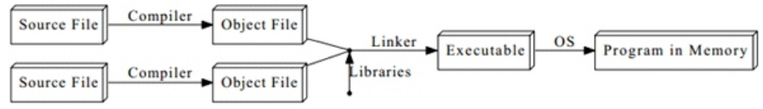
SAQ 1

3. **Portability:** different processors make different instructions available. Programs written as text can be translated into instructions for many different processors; one of C++'s strengths is that it can be used to write programs for nearly any processor.

C++ is a high-level language: when you write a program in it, the shorthands are sufficiently expressive that you don't need to worry about the details of processor instructions. C++ doesgive access to some lower-level functionality than other languages (e.g. memory addresses).

The Compilation Process

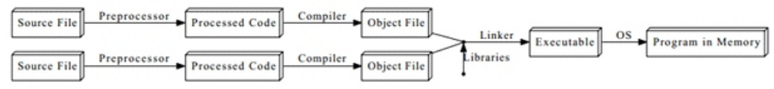
In the diagram below, you will observe how a program goes from text files (or source files) to processor instructions



You should also note that object files are intermediate files that represent an incomplete copy of the program: each source file only expresses a piece of the program, so when it is compiled into an object file, the object file has some markers indicating which missing pieces it depends on. The linker takes those object files and the compiled libraries of predefined code that they rely on, fills in all the gaps, and spits out the final program, which can then be run by the operating system (OS).

Bear in mind that the compiler and linker are just regular programs. The step in the compilation process in
which the compiler reads the file is called parsing. In C++, all these steps are performed ahead of time, before you start running a program.

I want you to understand that in some languages, they are done during the execution process, which takes time. This is
one of the reasons C++ code runs far faster than code in many more recent languages.
C++ actually adds an extra step to the compilation process: the code is run through a preprocessor, which applies some modifications to the source code, before being fed to the compiler. Thus, the modified diagram is:



General Notes on C++

You should be aware that C++ is immensely popular, particularly for applications that require speed and/or access to some low-level features. It was created in 1979 by Bjarne Stroustrup, at first as a set of extensions to the C programming language. C++ extends C; our first few lectures will basically be on the C parts of the language.

Though you can write graphical programs in C++, it is much hairier and less portable than
text-based (console) programs. We will be sticking to console programs in this course.
Everything in C++ is case sensitive: someName is not the same as SomeName.

Hello World

I want you to know that in the tradition of programmers everywhere, we'll use a “Hello, world!” program as an entry
point into the basic features of C++.

The code

```
1 // A Hello World program
2 # include <iostream >
3
4 int main () {
5 std :: cout << "Hello , world !\n";
6
7 return 0;
8 }
```


Tokens

Let us look at token briefly, Tokens are the minimal chunk of program that has meaning to the compiler – the smallest
meaningful symbols in the language. Our code displays all 6 kinds of tokens, though the
usual use of operators is not present here:

Token type	Description/Purpose	Examples
Keywords	Words with special meaning to the complier	int, double, for, auto
Identifiers	Names of things that are not built into the language	out, std, x, myFunction
Literals	Basic constant values whose value is specified directly in the source code	“Hello, world!, 24.3, 0, ‘c’
Operators	Mathematical or logical operators	+, - , &&, %, <<
Punctuations/ Separators	Punctuation defining the structure of a program	{ } ( ) , ;
Whitespace	Spaces of various sorts;	Spaces, tabs, newlines, comments

Rule for Constructing User defined Identifier

SAQ 1

I. An identifier can only contain a combination of letter, digit and underscore (_)
ii. Must not begin with a digit
iii. Must not contain a space
iv. Must not be any of the C++ keyword.

Line-By-Line Explanation

The Comment

I want you to know that once you see this sign //, it indicates that everything following it until the end of the line is a comment: it is ignored by the compiler. You should also learn that another way to write a comment is to put it between /* and */ (e.g. x = 1 + /*sneaky comment here*/ 1;). A comment of this form may

span multiple lines. Comments exist to explain non-obvious things going on in the code. Use them: document your code well!

Preprocessor commands

Another symbol I want you to learn is #. Lines beginning with # are preprocessor commands, which usually change what code
is actually being compiled. #include tells the preprocessor to dump in the contents of another file, here the iostream file, which defines the procedures for input/output.

int main()

The next one I want you to note is int main() {...}, it defines the code that should execute when the program starts up.
The curly braces represent grouping of multiple commands into a block. More about this syntax in the next few lectures.

cout <<

This is the syntax for outputting some piece of text to the screen.
We'll discuss how it works in later.

Namespaces: In C++, identifiers can be defined within a context – sort of a directory of names – called a namespace. When we want to access an identifier defined in a namespace, we tell the compiler to look for it in that namespace usingb the scope resolution operator (::). Here, we're telling the compiler to look for
cout in the std namespace, in which many standard C++ identifiers are defined.
A cleaner alternative is to add the following line below line 2:

Using namespace std

This line tells the compiler that it should look in the std namespace for any identifier we haven't defined. If we do this, we can omit the std:: prefix when writing cout. This is the recommended practice.

Strings: A sequence of characters such as Hello, world is known as a string. A string that is specified explicitly in a program is a string literal.

Escape sequences: T3he \n indicates a newline character. It is an example of an escape sequence – a symbol used to represent a special character in a text literal.

Escape Sequence	Represented Character
\a	System bell (beep sound)
\ b	Backspace
\ f	Formfeed (page break)
\ n	Newline (line break)
\ r	“Carriage return” (returns cursor to start of line)
\ t	Tab
\ \	Backslash
\ ,	Single quote character
\ ”	Double quote character
\ some integer *x*	The character represented

Return 0

This indicates that the program should tell the operating system it has completed successfully. This syntax will be explained in the context of functions; for now, just include it as the last line in the main block.

Note that every statement ends with a semicolon (except preprocessor commands and blocks using {}). Forgetting these semicolons is a common mistake among new C++ programmers.

Basic Language Features

So far our program doesn't do very much. Let's tweak it in various ways to demonstrate
some more interesting constructs.

Values and Statements

First, let us give a few definitions for proper understanding of some key words:

A *statement* is a unit of code that does something – a basic building block of a program.
An *expression* is a statement that has a value – for instance, a number, a string, the
sum of two numbers, etc. 4 + 2, x - 1, and "Hello, world!\n" are all expressions.
Not every statement is an expression. It makes no sense to talk about the value of an
#include statement, for instance.

Operators

I want you to know that we can perform arithmetic calculations with operators. Operators act on expressions to form
a new expression. Let us take a look at this example, we could replace "Hello, world!\n" with (4 + 2) / 3,
which would cause the program to print the number 2. In this case, the + operator acts on

the expressions 4 and 2 (its operands).

Operator types:

Mathematical: +, -, *, /, and parentheses have their usual mathematical meanings, including using - for negation. % (the modulus operator) takes the remainder of two numbers: 6 % 5 evaluates to 1.

Logical: used for “and,” “or,” and so on. More on those in the next lecture.
Bitwise: used to manipulate the binary representations of numbers. We will not focus
on these.

SAQ 2

Data Types

It may interest you to know that every expression has a type – a formal description of what kind of data its value is. Let us take a look at this instance, 0 is an integer, 3.142 is a floating-point (decimal) number, and "Hello, world!\n"
is a string value (a sequence of characters). Data of different types take a different amounts
of memory to store. Here are the built-in datatypes we will use most often:

Type Names	Description	Size	Range
char	Single text character or small integer. Indicated with single quotes (‘ a ’, ‘3’).	1 byte	signed: - 128 to 127
int	Large integer	4 bytes	signed: - 2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolen (true/false). Indicated with the keywords true and false	1 bytes	Just true (1) or false (0)
double	“Doubly” precise floating point number.	8 bytes	+/ - 1.7e +/ -308 (15 digits)

Notes on this table:

- You will observe that signed integer is one that can represent a negative number; an unsigned integer will never be interpreted as negative, so it can represent a wider range of positive numbers. Most compilers assume signed if unspecified.
- You should also note that there are actually 3 integer types: short, int, and long, in non-decreasing order of size (int is usually a synonym for one of the other two). You generally don't need to worry about which kind to use unless you're worried about memory usage or you're using really huge numbers. The same goes for the 3 floating point types, float, double, and long double, which are in non-decreasing order of precision (there is usually some imprecision in representing real numbers on a computer).

you must have seen that the sizes/ranges for each type are not fully standardized; those shown above are the
ones used on most 32-bit computers.

An operation can only be performed on compatible types. You can add 34 and 3, but you can't take the remainder of an integer and a floating-point number.

An operator also normally produces a value of the same type as its operands; thus, 1 / 4 evaluates to 0 because with two integer operands, / truncates the result to an integer. To get 0.25, you'd need to write something like 1 / 4.0.

A text string, for reasons we will learn in Lecture 5, has the type char *.

Variables

We might want to give a value a name so we can refer to it later. We do this using variable.

A variable is a named location in memory.

For example, say we wanted to use the value 4 + 2 multiple times. We might call it x and
use it as follows:

```
1 # include <iostream >

2 using namespace std ;

3

4 int main () {

5 int x;

6 x = 4 + 2;

7 cout << x / 3 << ' ' << x * 2;
```



```
8

9 return 0;

10 }
```

(Note how we can print a sequence of values by “chaining” the << symbol.) The name of a variable is an identifier token. Identifiers may contain numbers, letters, and underscores (_), and may not start with a number.

Line 5 is the declaration of the variable x. We must tell the compiler what type x will be
so that it knows how much memory to reserve for it and what kinds of operations may be
performed on it.

Line 6 is the initialization of x, where we specify an initial value for it. This introduces a
new operator: =, the assignment operator. We can also change the value of x later on in the
code using this operator.

We could replace lines 5 and 6 with a single statement that does both declaration and
initialization:

```
int x = 4 + 2;
```

This form of declaration/initialization is cleaner, so it is to be preferred.

5 Input

Now that we know how to give names to values, we can have the user of the program input
values. This is demonstrated in line 6 below:

```
1 # include <iostream >

2 using namespace std ;
```

```
3

4 int main () {

5 int x;

6 cin >> x;

7

8 cout << x / 3 << ' ' << x * 2;

9

10 return 0;

11 }
```

I want you to know that as cout << is the syntax for outputting values, cin >> (line 6) is the syntax for inputtingvalues.
Memory trick: if you have trouble remembering which way the angle brackets go for cout and cin, think of them as arrows pointing in the direction of data flow. cin represents the terminal, with data flowing from it to your variables; cout likewise represents the terminal, and your data flows to it.

Debugging

Let us take a look at the term debugging. There are two kinds of errors you'll run into when writing C++ programs: compilation errors and runtime errors. Compilation errors are problems raised by the compiler, generally resulting from violations of the syntax rules or misuse of types. These are often caused by typos and the like. Runtime errors are problems that you only spot when you run the program: you did specify a legal program, but it doesn't do what you wanted it to. These are usually more difficult to catch, since the compiler won't tell you about them.

Summary

So far in this unit, you have learned that:

- Conciseness, Maintainability and Portability are the advantages of C++ that makes it different from other programming languages.
- A program goes from text files (or source files) to processor instructions.
- An identifier can only contain a combination of letter, digit and underscore (_); must not begin with a digit; must not contain a space; and must not be any of the C++ keyword are rules for Rule for Constructing User defined Identifier.
- A *statement* is a unit of code that does something – a basic building block of a program.
- An *expression* is a statement that has a value.

Self-Assessment

- List the rules for constructing user defined identifiers in C++
- State 3 fundamental datatypes in C++ with their sizes in byte.
- List at least 5 keywords in C++

Tutor Marked Assessment

i)What are Expression and Statement?

ii)Write short note on all of the following terms in C++ programming language (i)Identifier (iii)Keywords (iii)Comment (iv)Constant (v)Symbolic constant

iii)Explain the meaning of following terms/symbols in C++ program (I)#include (ii)// (iii)main( ) (iv){ and } (v);

Further Reading

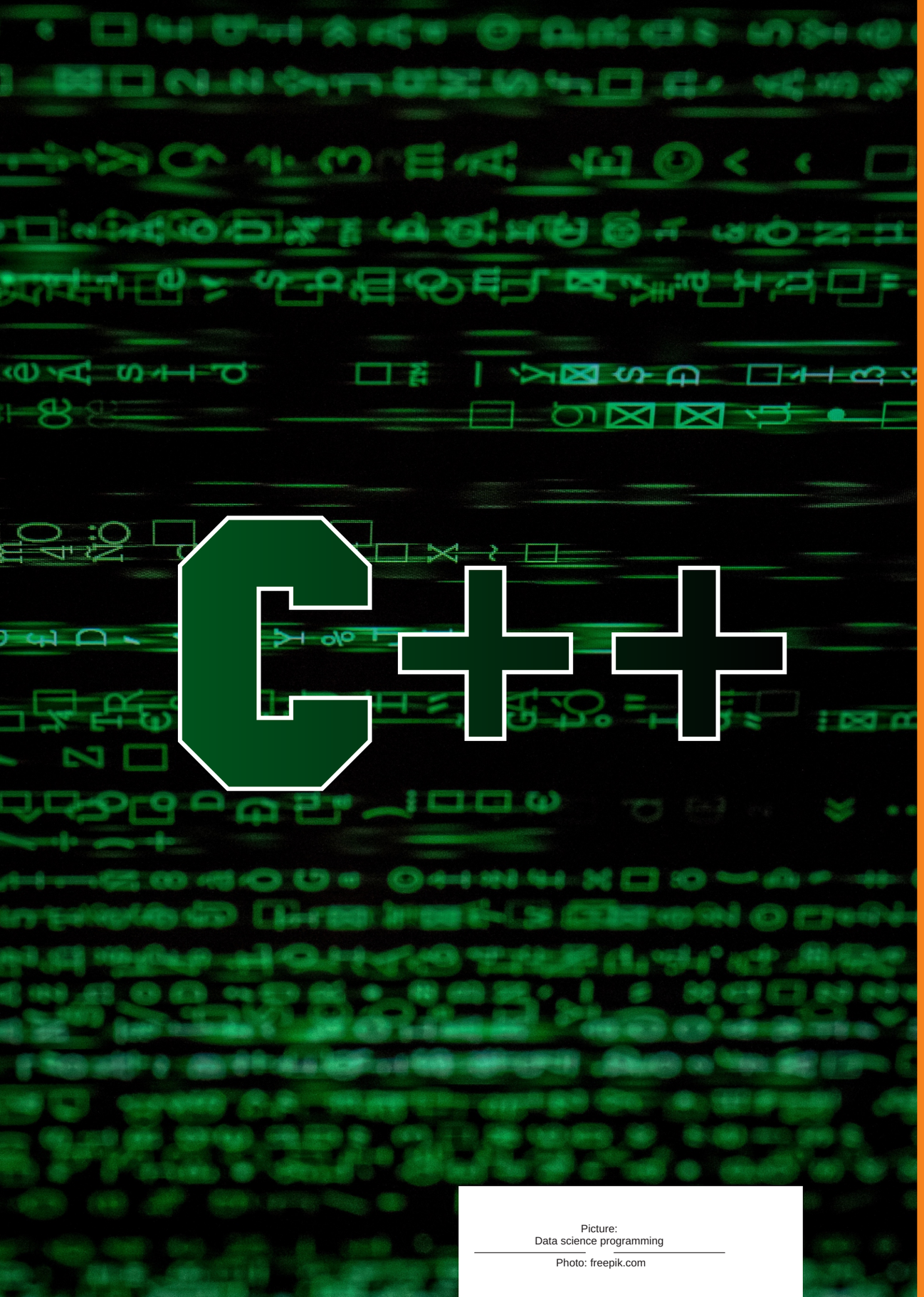
- https://hackr.io › blog › features-uses-applications-of-c-plus-plus-language
- https://www.quora.com › Why-should-I-learn-programming-languages-like-...
- https://www.geeksforgeeks.org › benefits-c-language-programming-langua...
- https://www.upwork.com › hiring › development › c-sharp-vs-c-plus-plus
- https://www.educba.com › ... › Blog › Programming Languages

References

(I)P. B. Shola (2002): Learn C++ Programming Language:
Reflect Publishers, Ibadan, Oyo State, Nigeria

(ii)John R Hubbard (2006): Programming with C++, Schaum's outlines
second edition: Tata McGraw-Hill Publishing Company Limited Newde

(iii)Paul Deitel & Harvey Deitel (2012): C++ How to program,
Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.



Picture:
Data science programming

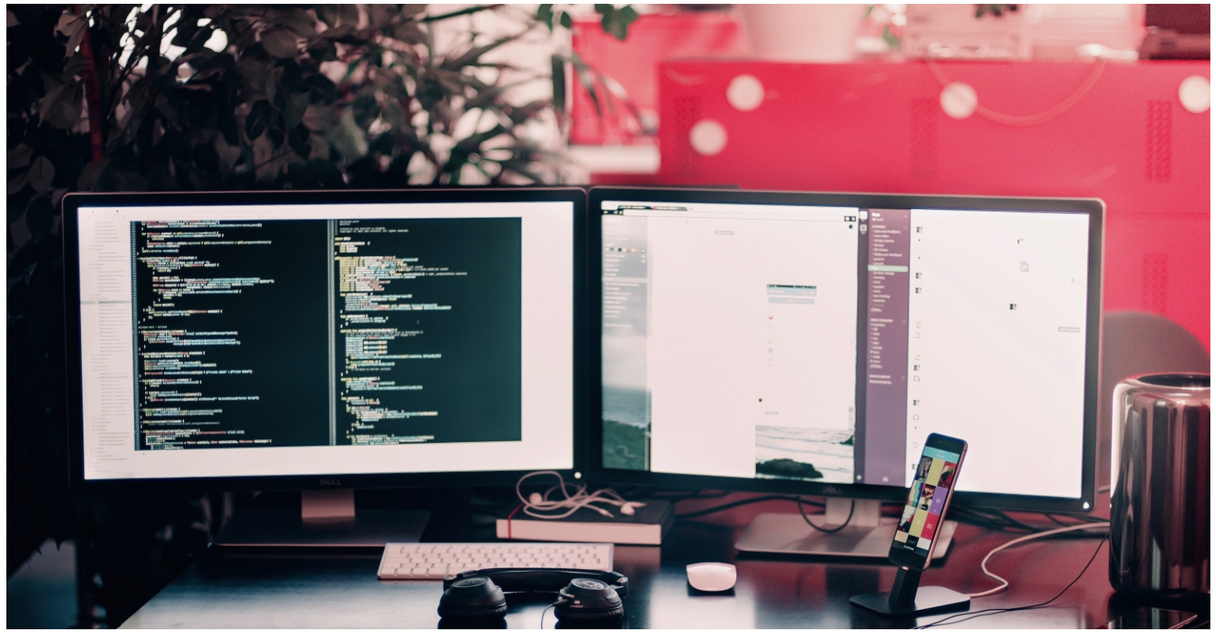
Photo: freepik.com

Module 2

Program Structure in C++

Units

Unit 1 - Statement in C++ Program
Unit 2 - Structure of C++ Program
Unit 3 - Basic Input and Output Operator in C++



UNIT 1

statement

Introduction

A **statement** in C++ is a block of code that does something. An **assignment** statement assigns a value to a variable. A **for statement** performs a loop. In C++ Statements are grouped together as one statement using curly brackets

In computer programming a statement can be thought of as the smallest standalone element of an imperative programming language. A program is formed by a sequence of one or more statements. A statement will have internal components (e.g., expressions). Finally, a brief description on process of compiling and running the program using C/C++ compliers was enumerated.

Learning Outcomes

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

1. Define a Statement
2. Differentiate between a statement and expression
3. State the 3 types of Statement with examples
4. Mention what is meant by input and output statement
5. Give two example of control statement

Main Content

Statement

5 min

SAQ 1

A statement causes an action to be taken in a program. In C++ statement are terminated by a semi-colon.

Expression

An expression is a combination of operator and variable or constant which can be evaluated to yield a value e.g a + 3, a + b + C, x = 5 e.t.c

Category of Statement in C++:

Basically, statement in C++ are grouped into three category namely: Null statement, Simple Statement and Compound Statement.

Null Statement: This is when noting is terminated by ;, it is used to provide an empty body to a **for** or **while** loop.

Simple statement; is a simple expression terminated by a semi-colon e.g x = 5;

Compound statement: A compound statement start and end with a curry bracket { & } e.g{ x = 5; a = (x + y)/5;}

Types of Statements in C++ Language

SAQ 3

C++ language Statements are fragments of the C++ program that are executed in sequence. The body of any function is a sequence of statements. For example: int main() {

```
int n = 1;                    // declaration statement

n = n + 1;                    // expression statement
```



```
std::cout << "n = " << n << '\n';          // expression statement
```

Expression Statements

C++ distinguishes between expressions and statements. Generally speaking, we could say that every expression becomes a statement if a semicolon is added.

However, we would like to discuss this topic a bit more.

SAQ 2

Expressions

Let us build this recursively from the bottom up. Any variable name (x, y, z, . . .), constant, or literal is an expression. One or more expressions combined by an operator constitute an expression, e.g., x + y or x * y + z. In several languages, such as Pascal, the assignment is a statement. In C++, it is an expression, e.g., x= y + z. As a consequence, it can be used within another assignment: x2= x= y + z. Assignments are evaluated from right to left. Input and output operations such as

std::cout ≪ "x is " ≪ x ≪ "\n"

are also expressions.

A function call with expressions as arguments is an expression, e.g., abs(x) or abs(x * y + z). Therefore, function calls can be nested: pow(abs(x), y). Note that nesting would not be possible if function calls were statements.

Since assignment is an expression, it can be used as an argument of a function: abs(x= y). Or I/O operations such as those above, e.g.:

print(std::cout ≪ "x is " ≪ x ≪ "\n", "I am such a nerd!");

Control Statement

A C++ control statement redirects the flow of a program in order to execute additional code. These statements come in the form of conditionals (if-else, switch) and loops (for, while, do-while). Each of them relies on a logical condition that evaluates to a boolean value in order to run one piece of code over another.

Examples are:

- **If-Else**

If-Else statements allow the program to execute different blocks of code depending on conditionals. All If statements have the following form:

if (condition) {//body}

- **While Loop**

A while loop is a simple loop that will run the same code over and over as long as a given conditional is true. The condition is checked at the beginning of each run through the loop (including the first one). If the conditional is false for the beginning, the while loop will be skipped all together.

while (conditional) {// loop body}

Declaration Statement

Let me get you familiar with the declaration statement. The basic elements of a program are the data declarations, functions, and comments. These can be organized into a simple C++ program. The heading comments tell the programmer all about the program. The *data declarations* describe the data that the program is going to use.

Example:

data declarations

int main( )

{ executable statements return(0); }

Our single function is named main. The name main is special, because it is the first function called. Any other functions are called directly or indirectly from main. The function main begins with:

int main( )

{

SAQ 3

Input and Output Statement in C++

Standard input (cin)

I want you to bear in mind that in most program environments, the standard input by default is the keyboard, and the C++ stream object defined to access it is cin . int age; cin >> age; The first statement declares a variable of type int called age , and the second extracts from cin a value to be stored in it.

Standard output (cout)

You should also note that on most program environments, the standard output by default is the screen, and the C++ stream object defined to access it is cout.

For formatted output operations, cout is used together with the insertion operator, which is written as << (i.e., two "less than" signs).

```
cout << "Output sentence";          // prints Output sentence on screen

cout << 120;                        // prints number 120 on screen

cout << x;                          // prints the value of x on screen
```

Standard input (cin)

In most program environments, the standard input by default is the keyboard, and the C++ stream object defined to access it is cin.

I want you to know that formatted input operations, cin is used together with the extraction operator, which is written as >> (i.e., two "greater than" signs). This operator is then followed by the variable where the extracted data is stored. For example:

```
int age;

cin >> age;
```

Compiling and Running a C++ program

Just before you move to look at the structure of C++ language, it is necessary for you to know that to be able to write and run a C++ program, a code editor with a compiler is required. There are various IDE applications available for editing, compiling and running a C or C++ program. Code Blocks IDE is recommended to compile and run all the code examples. Code blocks is an open source integrated development environment that contain the GNU GCC C/C++ compiler.

It was built to meet the most demanding needs of its users and designed to be very extensible and fully configurable. It is one of the most common C compilers available to programmers to properly edit, debug, compile, build and run C, C++ and FORTRAN programs.

The compiler is a free source and can be downloaded from https://sourceforge.net/projects/codeblocks/files/Binaries/17.12/Windows/codeblocks-17.12-setup.exe/download. The installation process would have being covered in Compute Programming I in the first semester.

Summary

- A statement causes an action to be taken in a program. In C++ statement are terminated by a semi-colon.
- An expression is a combination of operator and variable or constant which can be evaluated to yield a value e.g a + 3, a + b + C, x = 5 e.t.c
- Statement in C++ is grouped into three category namely: Null statement, Simple Statement and Compound Statement.
- A C++ control statement redirects the flow of a program in order to execute additional code.
- These statements come in the form of conditionals (if-else, switch) and loops (for, while, do-while).

Self-Assessment

SAQ

- Define a statement
- What is an expression?
- State three types of Statement in C++ Language
- What are input and output statements in C++?
- How do you write cout in C++?

Tutor Marked Assessment

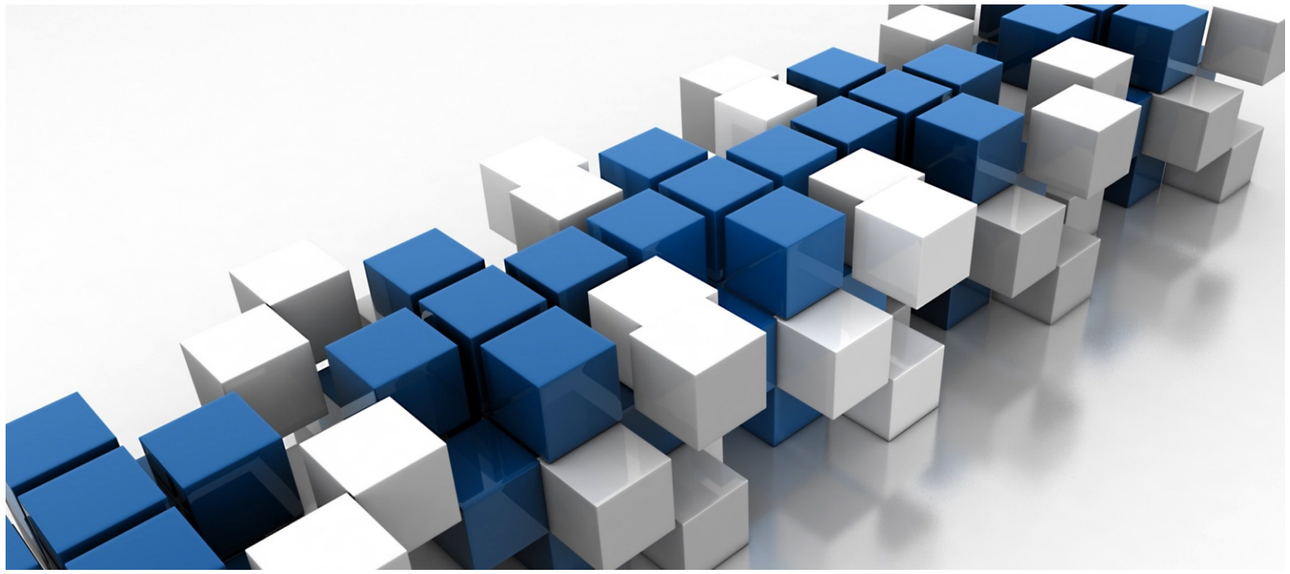
- With example state the different types of statement in C++ language
- State the difference between statement and expression
- What is the main use of a Null Statement?
- What is meant by control statement?
- What are the 3 types of loops?

Further Reading

- http://en.cppreference.com/w/cpp/string/basic_string:
- http://www.cplusplus.com/doc/tutorial/
- https://people.clarkson.edu › ~cashinpj › notes › control
- https://www.oreilly.com/library/view/practical-c-programming
- https://www.ntu.edu.sg › ehchua › programming › cpp › cp0_Introduction

References

- Peter Gottschling (2016): Discovering Modern C++:
  An Intensive Course for Scientists, Engineers and Programmers.
  Published Dec 17, 2015 by Addison-Wesley Professional.
  Part of the C++ In-Depth Series series.



UNIT 2

Structure of C++ Program

Introduction

You must have learnt in previous unit that undoubtedly, the best way to start learning a programming language is by writing a program. Therefore, this unit focuses fundamental components that every C++ program has. It also provides a comprehensive explanation on the structure of a complete C++ program. In addition we discussed and analyze the first program to be considered in this course.

Learning Outcomes

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

1. Write a simple program in C++ following the standard structure of C++ programs
2. Understand the key word in C++

Main Content

My First C++ Program

6 min

SAQ 2

Let me get you started this way by saying probably, the best way to start learning a programming language is by writing a program. Therefore, here is our first program:

Left Hand Side (LHS)	Right Hand Side (RHS)
//my first program in C++ #include <iotream> using namespace std; int main () {cout<<"Hello World!"; return 0;}	Hello World!

From our first program above the (LHS) shows the source code for our first program. The (RHS) shows the result of the program once compiled and executed. To the left, the double slash on the first line is for documentation - these are not part of the program, and are shown here merely for informational purposes.

I want you to learn that the way to edit and compile a program depends on the compiler you are using. Depending on whether it has a Development Interface or not and on its version. Consult the compilers section and the manual or help included with your compiler if you have doubts on how to compile a C++ console program.

You should be aware of the fact that the previous program is the typical program that programmer apprentices write for the first time, and its result is the printing on screen of the "Hello World!" sentence. It is one of the simplest programs that can be written in C++, but it already contains the fundamental components that every C++ program has. We are going to look line by line at the

// Comment Line

What I have shown you right there is refer to as a comment line. Let me explain further, all lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program. The programmer can use them to include short explanations or observations within the source code itself. In this case, the line is a brief description of what our program is.

#include <iostream>

The next thing I want you to know is that lines beginning with a hash sign (#) are directives for the preprocessor. They are not regular code lines with expressions but indications for the compiler's preprocessor. In this case the directive #include <iostream> tells the preprocessor to include the iostream standard file. This specific file (iostream) includes the declarations of the basic standard input-output library in C++, and it is included because its functionality is going to be used later in the program.

using namespace std;

All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name *std*. So in order to access its functionality we declare with this expression that we will be using these entities. This line is very frequent in C++ programs that use the standard library, and in fact it will be included in most of the source codes included in this lesson.

int main ()

This line corresponds to the beginning of the definition of the main function. The main function is the point by where all C++ programs start their execution, independently of its location within the source code. It does not matter whether there are other functions with other names defined before or after it - the instructions contained within this function's definition will always be the first ones to be executed in any C++ program. For that same reason, it is essential that all C++ programs have a *main* function.

The word main is followed in the code by a pair of parentheses (()). That is because it is a function declaration: In C++, what differentiates a function declaration from other types of expressions is these parentheses that follow its name. Optionally, these parentheses may enclose a list of parameters within them.

Right after these parentheses we can find the body of the main function enclosed in braces ({}). What is contained within these braces is what the function does when it is executed.

cout << "Hello World!";

This line is a C++ statement. A statement is a simple or compound expression that can actually produce some effect. In fact, this statement performs the only action that generates a visible effect in our first program.

cout is the name of the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters (in this case the Hello World sequence of characters) into the standard output stream (*cout*, which usually corresponds to the screen).

cout is declared in the *iostream* standard file within the *std* namespace, so that's why we needed to include that specific file and to declare that we were going to use this specific namespace earlier in our code.

At this point, I want you to notice that the statement ends with a semicolon character (;). This character is used to mark the end of the statement and in fact

it must be included at the end of all expression statements in all C++ programs (one of the most common syntax errors is indeed to forget to include some semicolon after a statement).

return 0;

The return statement causes the main function to finish. return may be followed by a return code (in our example is followed by the return code with a value of zero). A return code of *0* for the *main* function is generally interpreted as the program worked as expected without any errors during its execution. This is the most usual way to end a C++ console program.

You may have noticed that not all the lines of this program perform actions when the code is executed. There were lines containing only comments (those beginning by //). There were lines with directives for the compiler's preprocessor (those beginning by #). Then there were lines that began the declaration of a function (in this case, the main function) and, finally lines with statements (like the insertion into *cout*), which were all included within the block delimited by the braces ({}) of the main function.

The program has been structured in different lines in order to be more readable, but in C++, we do not have strict rules on how to separate instructions in different lines. For example, instead of

int main ( ) {Cout <<"Hello World!";return 0;}

We could have written: int main ( ) { cout<<"Hello World!; return 0;}

I want you to observe that if we had all in just one line, this would have had exactly the same meaning as the previous code.

you should know that this is because in C++, the separation between statements is specified with an ending semicolon (;) at the end of each one, so the separation in different code lines does not matter at all for this purpose. We can write many statements per line or write a single statement that takes many code lines. The division of code in different lines serves only to make it more legible and schematic for the humans that may read it.

Let us add an additional instruction to our first program:

Left Hand Side (LHS)	Right Hand Side (RHS)
// my second program in C++ #include <iostream> using namespace std;int main () {cout<<"Hello World!"; cout<<"I'm a C++ program"; return 0;}	*Hello World! I'm a C++ program*

In this case, we performed two insertions into cout in two different statements. Once again, the separation in different lines of code has been done just to give greater readability to the program, since *main* could have been perfectly valid defined this way:

```
int main () { cout <<"Hello World!"; cout<<"I'm a C++ program"; return 0;}
```

We were also free to divide the code into more lines if we considered it more convenient:

```
int main ()
{cout<<"Hello World!";
cout<<"I'm a C++ program";
return 0;}
```

And the result would again have been exactly the same as in the previous examples.

Preprocessor directives (those that begin by #) are out of this general rule since they are not statements. They are lines read and processed by the preprocessor and do not produce any code by themselves. Preprocessor

directives must be specified in their own line and do not have to end with a semicolon (;).

Comments

SAQ 1

Do you remember comments? Comments are parts of the source code disregarded by the compiler. They simply do nothing. Their purpose is only to allow the programmer to insert notes or descriptions embedded within the source code.

C++ supports two ways to insert comments:

```
//line comment
/*block comment*/
```

So let us talk more about comment. The first of them, known as line comment, discards everything from where the pair of slash signs (//) is found up to the end of that same line. The second one, known as block comment, discards everything between the /* characters and the first appearance of the */ characters, with the possibility of including more than one line.

We are going to add comments to our second program:

Source code (LHS)	Output (RHS)
/*my second program in C++ with more comment*/ #include <iostream> using namespace std; int main ()	Hello World! I'm a C++ program

```
{cout << "Hello World!";          // prints HelloWorld!
cout << "I'm a C++ program";     //prints I'm a C++ program
return 0;}
```

If you include comments within the source code of your programs without using the comment characters combinations //, /* or */, the compiler will take them as if they were C++ expressions, most likely causing one or several error messages when you compile it.

Summary

It has been an informing lesson, let us highlight some of the things we have learnt;

- **// Comment Line:** This is a comment line. All lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program.
- **#include <iostream> :** Lines beginning with a hash sign (#) are directives for the preprocessor.
- **using namespace std;** All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name *std*.
- **int main ():** This line corresponds to the beginning of the definition of the main function.
- **cout << "Hello World!";** This line is a C++ statement. A statement is a simple or compound expression that can actually produce some effect.
- **return 0;** The return statement causes the main function to finish.

Self-Assessment

- Write the two ways to include comment in C++
- Write a C++ program to display the word "I love C++ programming Language"

Tutor Marked Assessment

- Write a C++ Program to compute the sum of two numbers
- State the function of the following terms in C++ (a)comment line(b)return 0;(c) i n t m a i n ( ) (d) #include <iostream>

Further Reading

- https://www.ntu.edu.sg › ehchua › programming › cpp › cp0_Introduction
- http://www.cplusplus.com/doc/tutorial/
- https://people.clarkson.edu › ~cashinpj › notes › control
- https://www.oreilly.com/library/view/practical-c-programming

References

- (I)Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.
- (ii)Peter Gottschling (2016): Discovering Modern C++: An Intensive Course for Scientists, Engineers and Programmers. Published Dec 17, 2015 by Addison-Wesley Professional. Part of the C++ In-Depth Series series.
- (iii)P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- (iv)John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde

UNIT 3

Basic Input and Output Operator in C++

Introduction

Are you ready for the start of another unit? Here I will introduce to you the concepts of input and output techniques in C++. I want you to know that generally, input and output (I/O) operators are used to take input and display output. The operator used for taking the input is known as the extraction or get from operator (>>), while the operator used for displaying the output is known as the insertion or put to operator (<<).

Learning Outcomes

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

1. Describe the extraction and insertion operator in C++
2. Compute simple programs that read in data and display results on the screen
3. Compute programs using cascading of the input/output operator

Main Content

Input Operator

1 min

Let us start from here, the input operator, usually known as the extraction operator (>>), is used with the standard input stream, cin. It is important for you to note that, cin treats data as a stream of characters. These characters flow from cin to the program through the input operator. The input operator works on two operands, namely, the c in stream on its left and a variable on its right. Thus, the input operator takes (extracts) the value through cin and stores it in the variable.

Just so that you can understand the concept of an input operator, take a look at this example.

Example 1: Input Operator

```
#include<iostream>
using namespace, std;
int main ()
{
int a;cin>>a;
a = a+1; return 0;
}
```

From the example above, the statement cin>> a takes an input from the user and stores it in the variable a.

Output Operator

The output operator, also known as the insertion operator (<<), is used. The standard output stream cout Like cin, cout also treats data as a stream of characters. These characters flow from the program to cout through the output operator. The output operator works on two operands, namely, the cout stream on its left and the expression to be displayed on its right. The output operator directs (inserts) the value to count.

For you to understand the concept of an output operator, take a look at this example.

The program below demonstrates to you the working of an output operator.

Example 2: Output Operator

```
#include<iostream>

using namespace std;

int main ()

{

int a;cin>>a; a=a+1;cout<<a; return 0;

}
```

If you observe closely, you will see that this example 2 is similar to Example 1 above. The only difference is that the value of the variable a is displayed through the instruction cout << a .

Cascading of Input/Output Operators

The cascading of the input and output operators refers to the consecutive occurrence of input or output operators in a single statement.

In other to understand the concept of cascading of the input/output operator, let us consider these examples.

Example 3: Without Cascading of the Input/Output Operator

A program without cascading of the input/output operator.

```
#include<iostream>

using namespace std;

int main (){int a, b;cin>>a;

cin>>b;

cout<<"The value of a is

cout<<a;

cout<<"The value of b is

cout<<b;

return 0;

}
```

SAQ 3

Let us consider example 3, all cin and cout statements use separate input and output operators respectively.

However, these statements can be combined by cascading the input and

output operators accordingly as shown in this example.

A program with cascading of the input/output operator

Example 4: Another Example with cascading of the input/output operator

```
#include<iostream>

using namespace std;

int main ()

{

int a, b;

cin>>a>>b;

Cout<<"The value of b is : "<<b;

cout<<"The value of a is "<<a;

return 0;

}
```

When you consider example 4 above, the cascaded input operators wait for the user to input two values and the cascaded output operator first displays the message The value of a is: and then displays the value stored in a.

You have similar to that the case for the next statement.

We can observe that cascading of the input/output operator improves the readability and reduces the size of the program.

Summary

At the end of this unit, you have learned that;

- The input operator, usually known as the extraction operator (>>), is used with the standard input stream, cin.
- The output operator, also known as the insertion operator (<<), is used. The standard output stream cout Like cin, cout also treats data as a stream of characters.
- The cascading of the input and output operators refers to the consecutive occurrence of input or output operators in a single statement.

Self-Assessment

SAQ

- Describe the use of extraction operation in C++
- Write a simple C++ program to read in data and display results on the screen
- Write a program to describe input/output operator in C++ without cascading

Tutor Marked Assessment

I)Where does a cin stops its extraction of data ?

ii)What is the output of this program?

```
#include < iostream >
using namespace std;
int main ()
{
int i;
cout << "Please enter an integer value: ";
cin >> i + 4;
return 0;
}
```

iii)Write a program to describe input/output operator in C++ with cascading

Further Reading

I)Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.

ii)Peter Gottschling (2016): Discovering Modern C++: An Intensive Course for Scientists, Engineers and Programmers. Published Dec 17, 2015 by Addison-Wesley Professional. Part of the C++ In-Depth Series series.

(iii)P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria

(iv)John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde

References

http://ecomputernotes.com/cpp/introduction-to-oop/input-output-operator-in-cpp

http://www.cplusplus.com/doc/tutorial/basic_io/

https://www.geeksforgeeks.org › basic-input-output-c

http://www.allindiaexams.in/engineering/cse/c-p-p-multiple-choice-questions-answers/cpp-input-output-streams

https://www.programiz.com › library-function › iostream › cout

Picture:
keyboard

Photo: pixabay

Module 3

Selection Statements

Units

Unit 1 - Selection Statements
Unit 2 - Iteration or Loop Statement
Unit 3 - Break/Termination Control Structure

UNIT 1

Selection Statements

Introduction

Welcome to another unit where we will consider selection control statement in details. I want you to note that control structures are program statement which determines the order of execution of a program. Generally, the selection statement is the If statement, if-else statement and the switch statement.

I want you to bear in mind that in C++, a statement can be labeled using a valid identifier followed by colon and then the statement.

Learning Outcomes

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

1. Define selection control Statement
2. State the 3 types selection
3. Write a program on how to use the 3 types of selection control statement

Main Content

Selection Statements

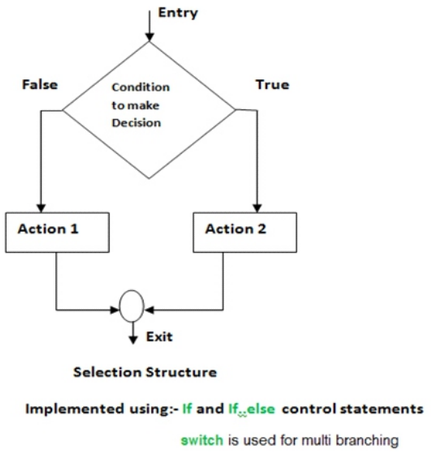
6 min

SAQ 1

Have you ever heard of a selection statement before and what it does? Let me tell you; a selection statement selects among a set of statements depending on the value of a controlling expression. The selection statements are the if statement, if else statement the switch statement, which are discussed in the following sections.

The diagram below shows selection structures implementation using If, If Else and Switch statements.

Selection statement structures



The if Statement

SAQ 2

The if statement allows conditional execution. It has the following syntax:

if (*condition*) *statement;*

where *condition* is an integral expression and *statement* is any executable statement. The statement will be executed only if the value of the integral expression is nonzero. Notice the required parentheses around the condition.

Example 3.1.1: This program tests if one positive integer is not divisible by another:

// This program tests if tests if one positive integer is not divisible by another:

```
int main ( )
{ int n. d:
cout <<"Enter two positive integers:":
cin >> n >>
if (n%d) cout << n <<"is not divisible by " d << endl:
```

On the first run, if we enter 66 and 7:

Enter two positive integers: 66 7

66 is not divisible by 7

I want you to take note that the value 66%7 is computed to be 3. Since that integral value is ***not zero,*** the expression is interpreted as a **true** condition and consequently the divisibility message is printed.

Let us observe another angle, if we *enter two positive integers: 56 7*

The value 56%7 is computed to be 0, which is interpreted to mean *"false,"* so the divisibility message is not printed.

In C++, whenever and integral expression is used as a condition, the value 0 means *"false"* and all other values means *"true."*

I want you to bear in mind that the program in example 3.1.1 above is inadequate because it provides no affirmative information when n is divisible by d. This fault can be remedied with an if else statement, which we shall discuss in section. 3.1.2

The if-else Statement

Let us now move on to the next statement. The if-else statement causes one of two alternative statements to execute depending upon whether the condition is true. Its has the following syntax;

If (condition) statement 1;

else statement 2;

where condition is an integral expression and *statement 1* and *statement 2* are executable statements. If the value of the condition is nonzero then *statement 1* will execute, otherwise *statement 2* will execute.

SAQ 3

Example: Using the example 3.1.1 program that test if one positive integer is not divisible by another:

Note that this program is the same as the program in example 3.1.1 above except that the *if statement* has been replaced by an if –else statement.:

// This program tests if tests if one positive integer is not divisible by another:

```
int main ( )

{ int n. d;

cout <<"Enter two positive integers:":

cin >> n >> d;

if (n%d) cout << n <<"is not divisible by
" d << endl;
```

Since when we enter 56 and 7, we get

an affirmative response

Enter two positive integers: 56 7

56 is divisible by 7

Since 56%7 is zero, the expression is interpreted as being a *false* condition and consequently the statement after the else is executed.

I want you to note that the if-else is only one statement, even though it requires two semicolons.

Consider the following example:

```
if (i < 1)

  funct(i);

else

  {

  i = x++;

  funct(i);

  }
```

Let us observe the example above, if the value of i is less than 1, then the statement funct(i) is executed and the compound statement following the keyword else is not executed. If the value of i is not less than 1, then only the compound statement following the keyword else is executed.

The control expression in a selection statement is usually a logical expression, but it can be any expression of scalar type.

When if statements are nested, an else clause matches the most recent if statement that does not have an else clause, and is in the same block. For example:

```
if (i < 1)

  {

  if (j < 1)

    funct(j);

  if (k < 1)        /* This if statement is associated with */

    funct(k);
```

```
else            /* this else clause.            */

   funct(j + k);

}
```

The switch control Statement

Next thing I want you to learn is the switch control statement. The switch control structure is a multiple, alternative decision control structure which is very similar to the else if clause when use with the else if clause.
The use of else if clause often generate a tedious if structure for a multiple alternative structure and thereby makes it hard to read and debug, this inform the implementation of the switch control structure.
Switch (expression)

```
{
        case const 1    :       stmt 1;
                                break;
        case const 2:           stmt 2;
                                break;
        :       :               :
        :       :               :
        :       :               :
        case const N:           stmt N;
                                break;
        default                 stmt D;
}
```

In the execution of these control structure the following step are taken

1.The expression is evaluated

2. The value of expression yielded in 1 is compared to the individual const 1, const 2,….. constN, if any of the const matches the value the statement associated with statement is executed.

3. Otherwise (if non of the const matches the value of the expression) the statement in the default part is executed (if present). The default part of the if statement is optional.

Example (a)char sex = 'F'switch (sex)

```
{
case 'F':  cout << '' \n FEMALE'';

   break;

case 'M':  cout'' \n MALE'';

   break;

default: cout'' \n invalid value'';

}
```

The use of the break stmt in the switch stmt is to avoid fall through.

The break transfer controls out of the switch stmt. It used to separate each of the case statement.

When the switch statement is executed, the following sequence takes place:

- The switch control expression is evaluated (and integral promotions applied) and compared with the constant expressions in the case labels.
- If the control expression's value matches a case label, control transfers to the statement following that label. If a break statement is encountered, the switch statement terminates; otherwise, execution continues into the following case or default statements until a break statement or the end of the switch statement is encountered.
- A switch statement can also be terminated by a return or goto statement. If a switch statement is inside a loop, the switch statement is terminated if a continue statement terminates the loop.

- If the control expression's value does not match any case label, and there is a default label, control is transferred to the statement following that label. If a break statement does not end the default statement, and a case label follows, that case statement is executed.

- If the control expression's value does not match any case label and there is no default label, execution of the switch statement terminates.

Example (b): Using switch to Count Blanks, Tabs, and New Lines

```
/*  This program counts blanks, tabs, and new lines in text *
 *  entered from the keyboard.                          */
#include <stdio.h>
main()
{
 int number_tabs = 0, number_lines = 0, number_blanks = 0;
  int ch;
  while ((ch = getchar()) != EOF)
    switch (ch)
      {
        case '\t': ++number_tabs;
                   break;
        case '\n':  ++number_lines;
                   break;
        case ' ' :  ++number_blanks;
                   break;
        default:;
```



```
      }
  printf("Blanks\tTabs\tNewlines\n");
  printf("%6d\t%6d\t%6d\n", number_blanks,
                    number_tabs,number_lines);
}
```

Key to 3.1.3 Example (b):

- A series of case statements is used to increment separate counters depending on the character encountered.

- The break statement causes control to return to the while loop. Control is passed to the while loop if the value of ch does not match any of the case constant expressions.

Without the break statements, each case would drop through to the next.

If variable declarations appear in the compound statement within a switch statement, initializers on auto or register declarations are ineffective. However, initializations within the statements following a case are effective. Consider the following example:

```
switch (ch)
  {
    int nx = 1;        /* Initialization ignored          */
    printf("%d", n);    /* This first printf is not executed */
    case 'a' :
     { int n = 5;      /* Proper initialization occurs     */
       printf("%d", n);
       break; }
    case 'b' :
```

```
    { break; }
  default :
    { break; }
}
```

In this example, if ch == 'a' , then the program prints the value 5. If the variable equals any other letter, the program prints nothing because the initialization occurs outside of the case label, and statements outside of the case label are ineffective.

Summary

- A selection statement selects among a set of statements depending on the value of a controlling expression.
- ·The selection statements are the if statement, if else statement the switch statement,
- ·The if statement allows conditional execution. It has the following syntax:

  if (condition) statement;
- ·The if-else statement causes one of two alternative statements to execute depending upon whether the condition is true. Its has the following syntax;

  If (condition) statement 1;

  else statement 2;
- ·The use of else if clause often generate a tedious if structure for a multiple alternative structure and thereby makes it hard to read and debug, this inform the implementation of the switch control structure.

Self-Assessment

SAQ

- Define selection control statement in C++
- State the 3 types of selection control statement in C++
- Write a program to tests if one positive integer is not divisible by another

Tutor Marked Assessment

- Why are selection statements used?
- How do you explain an if statement?
- How do you write an IF ELSE statement in C++?

Further Reading

- www.cplusplus.com › doc › tutorial › control
- ecomputernotes.com › ... › Control Structures
- https://www.w3adda.com › cplusplus-tutoria
- www.cs.iit.edu › lectures › Selection
- https://www.geeksforgeeks.org › decision-m

References

- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.
- Peter Gottschling (2016): Discovering Modern C++: An Intensive Course for Scientists, Engineers and Programmers. Published Dec 17, 2015 by Addison-Wesley Professional. Part of the C++ In-Depth Series series.
- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde

UNIT 2

Iteration or Loop Statement

Introduction

A warm welcome to another unit of this course. We will discuss the Iteration or Loop statement in C++ . It is one of the control structure in C++. The various iteration statement that we be covered in this unit are the for loop, while loop and the do while loop.

Learning Outcomes

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

1. Define iteration control Statement in C++
2. Enumerate the 3 types iteration statement in C++
3. Write a simple program on how to use the 3 types of iteration statement in C++

Main Content

Iteration statement

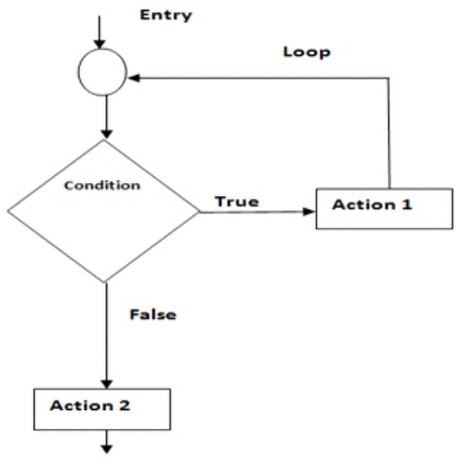
4 min

SAQ 1

Let us start by explaining the term iteration statement means. The statements that cause a set of statements to be executed repeatedly either for a specific number of times or until some condition is satisfied are known as iteration statements. That is, as long as the condition evaluates to True, the set of statement(s) is executed. Do you understand? The various iteration statements used in C++ are for loop, while loop and do while loop.

Look at the diagram below, it shows you the structures of a Loops. Loops have a purpose to repeat a statement a certain number of times or while a condition is fulfilled.

Loop structures



The for Loop

SAQ 2

I want you to bear in mind that the for loop is one of the most widely used loops in C++. The for loop is a deterministic loop in nature, what that means is that the number of times the body of the loop is executed is known in advance.

The syntax of the for loop is

for(initialize; condition; update)


```
{

//body of the for loop

}
```

I want you to note that initialize, condition and update are optional expressions and are always specified in parentheses. All the three expressions I mentioned now are separated by semicolons. The semicolons are mandatory and hence cannot be excluded even if all the three expressions are omitted.

To understand the concept of the for statement, consider this example.

SAQ 3

Example: A program to display a countdown using for loop

```
#include<iostream>

using namespace std;

int main ()

{

int n;

for(n=l; n<=10; n++)

cout<<n<<" "; //body of the loop

cout<<"\n This is an example of for loop!!!";

                    //next statement in sequence

return 0;

}
```

The output of this program is

```
1 2 3 4 5 6 7 8 9 10

This is an example of for loop!! !
```

for loop using comma operator: for loop allows multiple variables to control the loop using comma operator. That is, two or more variables can be used in the initialize and the update parts of the loop. For example, consider this statement.

```
for (i=1,j=50 ;i<10;i++, j- -)
```

This statement initializes two variables, namely i and j and updates them. Note that for loop can have only one condition.

The while Loop

The next loop you should take note of is the while loop. The while loop is used to perform looping operations in situations where the number of iterations is not known in advance. That is, unlike the for loop, the while loop is non deterministic in nature.

The syntax of the while loop is

```
while(condition)
{
/ / body of while loop
}
```

These points should be noted about the while loop.

- Unlike for loops where explicit initialize and update expressions are specified, while loops do not specify any explicit initialize and update expressions. This implies that the control variable must be declared and initialized before the while loop and needs to be updated within the body of the while loop .

- The while loop executes as long as condition evaluates to True. If condition evaluates to False in the first iteration, then the body of while loop never executes.

- While loop can have more than one expression in its condition. However, such multiple expressions must be separated by commas and are executed in the order of their appearance.

To understand the concept of the while loop, consider this example.

Example : A program to determine the sum of first n consecutive positive integers

```
#include<iostream>

using namespace std;

int main ()

{

int n,i,sum; // i is the control variable

cout<<" Enter the number of consecutive positive"

<< "\n integers(starting from 1): ";

Cin>>n;

sum=0;

i=l;     // initialize expression

while (i<=n)

{
```

```
sum+=i;

++i; //update expression

}

Cout<<"\nThe sum is "<<sum;

 return 0;

}
```

The output of the program is

Enter the number of consecutive positive integers(starting from 1): 9

The sum is : 45

The do-while loop

The do-while loop: in a while loop, the condition is evaluated at the beginning of the loop and if the condition evaluates to False, the body of the loop is not executed even once. However, if the body of the loop is to be executed at least once, no matter whether the initial state of the condition is True or False, the do-while loop is used. This loop places the condition to be evaluated at the end of the loop.

The syntax of the do-while loops is given here.Do

```
{

//body of do while loop

}while(condition) ;
```

To understand the concept of do-while loop, consider this example.

Example : A program to calculate the sum of an Arithmetic Progression (AP)


```
#include<iostream>

using namespace std;

int main ()

{

int a,d,n,sum,term=0; /*a is the first term , d is

the common difference, n is the number of terms to be summed */

cout<<"Enter the first term, common difference,"

<<"and the number of terms to be summed"

<<"respectively:\n";

Cin>>a>>d>>n;

sum=0;

int i=l;
```

```
term= a+ (i-1)*d;

sum+=term; //Adding each term to 'sum'

cout<<term<<" “;

++i;

 }
```

```
while (i<=n) ;

cout<<"\nThe sum of A.P. is "<<sum;

return 0;

}
```

The output of the program is

Enter the first term, common difference, and the number of terms to be summed respectively:

3

6

5

The terms are 3 9 15 21 27

The sum of A.P. is 75

Note that, all the three loops (for, while and do-while) can be nested within the body of another loop

Summary

- The statements that cause a set of statements to be executed repeatedly either for a specific number of times or until some condition is satisfied are known as iteration statements.
- The various iteration statements used in C++ are for loop, while loop and do while loop.
- ·The for loop is one of the most widely used loops in C++; The syntax of the for loop is

  for(initialize; condition; update)


```
{
//body of the for loop
}
```

- The while loop is used to perform looping operations in situations where the number of iterations is not known in advance. The syntax for while loop

  while(condition)

  {

  / / body of while loop

  }
- The do-while loop: in a while loop, the condition is evaluated at the beginning of the loop and if the condition evaluates to False, the body of the loop is not executed even once.

  The syntax of the do-while loops is given here.

  Do

  {

  //body of do while loop

  } while(condition) ;

Self-Assessment

- Define Iterative statement in C++
- Enumerate the 3 types of iterative statement in C++
- Write a program to display a countdown using for loop

Tutor Marked Assessment

- What do you mean by iterative statement?
- Write the syntax for while loop in C++
- Write a program to calculate the sum of an Arithmetic Progression (AP)

Further Reading

- https://www.w3schools.in › cplusplus-tutorial
- https://press.rebus.community › chapter ›
- ecomputernotes.com › ... › Control Structures
- https://www.geeksforgeeks.org › loops-in-c-..
- www.cplusplus.com › doc › tutorial › control

References

- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.
- Peter Gottschling (2016): Discovering Modern C++: An Intensive Course for Scientists, Engineers and Programmers. Published Dec 17, 2015 by Addison-Wesley Professional. Part of the C++ In-Depth Series series.
- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde

Unit 2 | Module 3 | CSC 212 Iteration or Loop Statement

UNIT 3

Break/Termination Control Structure

Introduction

In this unit, we shall be discussing on the various types of termination/break control structure, this includes the break statement; continue statement and the goto statement. We will also deal with exit function and the selective structure: switch in the course of this unit.

In C++, there are two statements break; and continue; specifically, to alter the normal flow of a program.

I want you to know that sometimes, it is desirable to skip the execution of a loop for a certain test condition or terminate it immediately without checking the condition.

Learning Outcomes

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

1. Define break statement in C++
2. Enumerate the uses of break statement in C++
3. Highlight the difference between break and continue statement in C++?

Main Content

Jump Statements

4 min

The break statement

SAQ 1

Using break, we can leave a loop even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end. For example, we are going to stop the count down before its natural end (maybe because of an engine check failure?):

```
// break loop example

#include <iostream>

using namespace std;

int main ()

{

int n;                                                  return 0;
                                                       }
for (n=10; n>0; n--)

 {

 cout << n << ", ";

 if (n==3)

 {

 cout << "countdown aborted!";

break;

}

}
```

SAQ 2


```
return 0;

}10, 9, 8, 7, 6, 5, 4, 3, countdown aborted!
```

3.1.2The continue statement

The continue statement causes the program to skip the rest of the loop in the current iteration as if the end of the statement block had been reached, causing it to jump to the start of the following iteration. For example, we are going to skip the number 5 in our countdown:

```
// continue loop example

#include <iostream>

using namespace std;



int main ()

{

 for (int n=10; n>0; n--) {

  if (n==5) continue;

  cout << n << ", ";

 }

 cout << "FIRE!\n";

 return 0;

}10, 9, 8, 7, 6, 4, 3, 2, 1, FIRE!

 return 0;

}
```

The goto statement

goto allows to make an absolute jump to another point in the program. You should use this feature with caution since its execution causes an unconditional jump ignoring any type of nesting limitations.
The destination point is identified by a label, which is then used as an argument for the goto statement. A label is made of a valid identifier followed by a colon (:).

Generally speaking, this instruction has no concrete use in structured or object oriented programming aside from those that low-level programming fans may find for it. For example, here is our countdown loop using goto:

```
// goto loop example

#include <iostream>

using namespace std;

int main ()

{

 int n=10;

 loop:

 cout << n << ", ";

 n--;

 if (n>0) goto loop;

 cout << "FIRE!\n";

 return 0;

}10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIRE!
```


The exit function

exit is a function defined in the cstdlib library.

I want you to observe that the purpose of exit is to terminate the current program with a specific exit code. Its prototype is:
void exit (int exitcode);
The exitcode is used by some operating systems and may be used by calling programs. By convention, an exit code of 0 means that the program finished normally and any other value means that some error or unexpected results happened.

The Selective Structure: Switch

I want you to take note that the syntax of the switch statement is a bit peculiar. Its objective is to check several possible constant values for an expression. Something similar to what we did at the beginning of this section with the concatenation of several if and else if instructions. Its form is the following:
switch (expression)

```
{

 case constant1:

group of statements 1;

 break;

 case constant2:

   group of statements 2;

   break;
```

```
 .

 .

 .

default:

 default group of statements

}
```

works in the following way: switch evaluates expression and checks if it is equivalent to constant1, if it is, it executes group of statements 1 until it finds the break statement. When it finds this break statement the program jumps to the end of the switch selective structure.

If expression was not equal to constant1 it will be checked against constant2. If it is equal to this, it will execute group of statements 2 until a break keyword is found, and then will jump to the end of the switch selective structure.

Finally, if the value of expression did not match any of the previously specified constants (you can include as many case labels as values you want to check), the program will execute the statements included after the default: label, if it exists (since it is optional).

Both of the following code fragments have the same behavior:

switch example	if-else equivalent

switch (x) {
case 1:
  cout << "x is 1";
  break;
case 2:
  cout << "x is 2";
  break;
default:
  cout << "value of x unknown";
}
	if (x == 1) {
	  cout << "x is 1";
	  }
	else if (x == 2) {
	  cout << "x is 2";
	  }
	else {
	  cout << "value of x unknown";
	  }

The switch statement is a bit peculiar within the C++ language because it uses labels instead of blocks. This forces us to put break statements after the group of statements that we want to be executed for a specific condition. Otherwise the remainder statements -including those corresponding to other labels- will also be executed until the end of the switch selective block or a break statement is reached.

Let us take a look at this example, if we did not include a break statement after the first group for case one, the program will not automatically jump to the end of the switch selective block and it would continue executing the rest of statements until it reaches either a break instruction or the end of the switch selective block. This makes it unnecessary to include braces { } surrounding the statements for each of the cases, and it can also be useful to execute the same block of instructions for different possible values for the expression being evaluated. For example:

```
1. switch (x) {

 2. case 1:

 3. case 2:

 4. case 3:

 4. case 3:

5. cout << "x is 1, 2 or 3";

6.break;

7. default:

8. cout << "x is not 1, 2 nor 3";

9. }
```

Notice that switch can only be used to compare an expression against constants. Therefore we cannot put variables as labels (for example case n: where n is a variable) or ranges (case (1..3):) because they are not valid C++ constants.

If you need to check ranges or values that are not constants, use a concatenation of if and else if statements.

Summary

- In C++, there are two statements break; and continue; specifically to alter the normal flow of a program.
- Using break we can leave a loop even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.
- The continue statement causes the program to skip the rest of the loop in the current iteration as if the end of the statement block had been reached, causing it to jump to the start of the following iteration.
- goto allows to make an absolute jump to another point in the program. You should use this feature with caution since its execution causes an unconditional jump ignoring any type of nesting limitations.

Self-Assessment

SAQ

- Define a break statement in C++
- What are the uses of break statement in C++
- Enumerate the difference between the break and continue statement in C++

Tutor Marked Assessment

- What is continue in if statement?
- What is goto statement C++?
- Write the syntax of the switch statement in C++

Further Reading

- https://www.geeksforgeeks.org › break-statement-cc
- https://stackoverflow.com › questions › break-c-which-loop-is-it-actually-bre...
- https://beginnersbook.com › 2014/01 › c-continue-statement
- https://www.programiz.com › break-continue
- https://www.w3adda.com › cplusplus-tutorial

References

- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.
- Peter Gottschling (2016): Discovering Modern C++: An Intensive Course for Scientists, Engineers and Programmers. Published Dec 17, 2015 by Addison-Wesley Professional. Part of the C++ In-Depth Series series.
- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde

Picture:
binary code

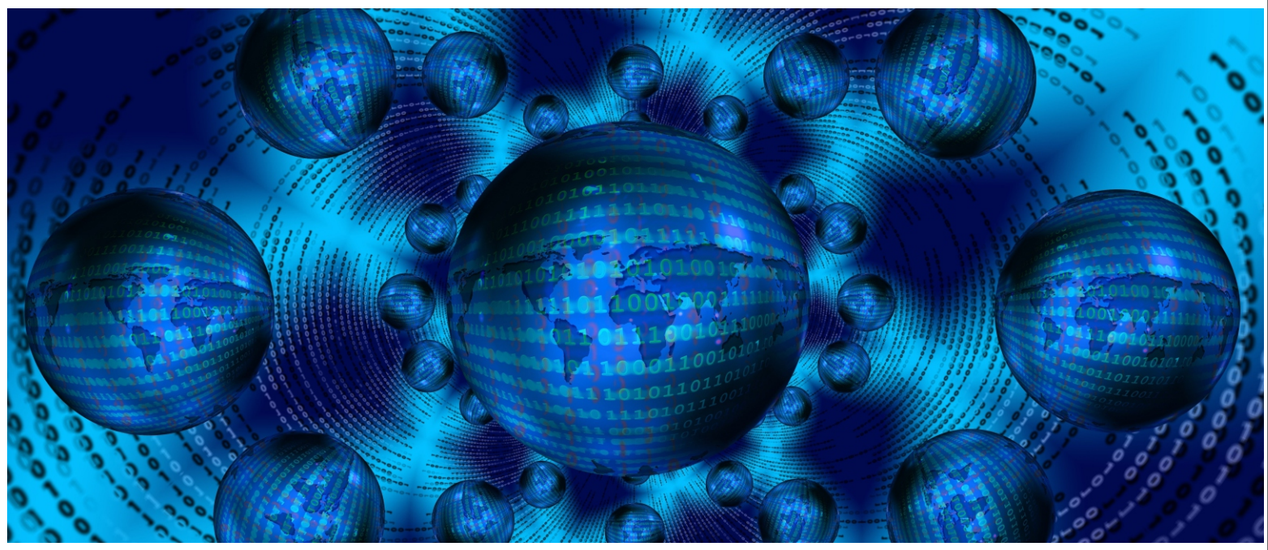
Photo: pixabay

Module 4

Function

Units

Unit 1 - Function
Unit 2 - Function Call and Function prototype (declaration)
Unit 3 - Types of Function



UNIT 1

FUNCTION

Introduction

In the unit, you will learn how function is used to provide modularity to program. In programming, function refers to a segment that group code to perform a specific task.

Creating an application using function makes it easier to understand, edit, check errors. You will also learn library function and user defined function.

Learning Outcomes

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

1. What is function are used in C++
2. Types of functions in C++,
3. How to use the two types of function with examples.

Main Content

Function

2 min

Library Function

The first thing I want you to bear in mind is depending on whether a function is predefined or created by programmer; there are two types of function:

SAQ 2

1.Library Function

2.User-defined Function

Library functions are the built-in function in C++ programming.

Programmer can use library function by invoking function directly; they don't need to write it themselves.

Example 1: Library Function

```
#include <iostream>

#include <cmath>

using namespace std;

int main()

{

double number, squareRoot;

cout << "Enter a number: ";

cin >> number;
```



```
// sqrt() is a library function to calculate square root

squareRoot = sqrt(number);

cout << "Square root of " << number << " = " << squareRoot;

return 0;

}
```

Output

```
Enter a number: 26

Square root of 26 = 5.09902
```

In the example above, sqrt() library function is invoked to calculate the square root of a number.

Notice code #include <cmath> in the above program. Here, cmath is a header file. The function definition of sqrt()(body of that function) is present in the cmath header file.

You can use all functions defined in cmath when you include the content of file cmath in this program using #include <cmath> .

Every valid C++ program has at least one function, that is, main() function.

User-defined Function

C++ allows programmer to define their own function.

A user-defined function groups code to perform a specific task and that group of code is given a name(identifier).

When the function is invoked from any part of program, it all executes the codes defined in the body of function.

How user-defined function works in C Programming?

When a program begins running, the system calls the main() function, that is, the system starts executing codes from main() function.

When control of the program reaches to function_name() inside main(), it moves to void function_name() and all codes inside void function_name() is executed.

Then, control of the program moves back to the main function where the code after the call to the function_name() is executed as shown in figure

Example 2: User Defined Function

C++ program to add two integers. Make a function add() to add integers and display sum in main() function.

```
#include <iostream>

using namespace std;

// Function prototype (declaration)

int add(int, int);

int main()

{

int num1, num2, sum;

cout<<"Enters two numbers to add: ";

 cin >> num1 >> num2;
```



```
// Function call

 sum = add(num1, num2);

 cout << "Sum = " << sum;

return 0;

}

// Function definition

int add(int a, int b)

{

int add;

 add = a + b;

// Return statement

return add;

}
```

Output

```
Enters two integers: 8

-4

Sum = 4
```

Summary

At the end of this unit you have learned that;

- Library functions are the built-in function in C++ programming.
- Programmer can use library function by invoking function directly; they don't need to write it themselves.
- Library Function and User-defined Function are the two types of function in C++.
- A user-defined function groups code to perform a specific task and that group of code is given a name(identifier).
- When the function is invoked from any part of program, it all executes the codes defined in the body of function.

Self-Assessment

SAQ

- Define function as used in C++
- State 2 types of functions in C++
- Write C++ code to describe the two types of function mentioned in (ii).

Tutor Marked Assessment

- Describe a function Prototype in C++ and give at least one example.
- Write a C++ program to add two integers. Make a function add() to add integers and display sum in main() function.

Further Reading

- http://www.cplusplus.com/doc/tutorial/functions/
- https://www.studytonight.com/cpp/operator-overloading-examples.php
- https://www.programiz.com › cpp-programming › function
- https://www.w3schools.com › cpp › cpp_functions

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall

UNIT 2

Function Call and Function prototype

Introduction

In the unit, I will teach you function call and function prototype how function call is used in C++ programming, function definition, passing argument to function and some notes passing argument and return statement.

Learning Outcomes

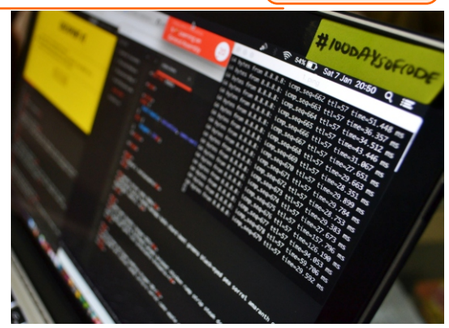
At the end of this unit, you should be able to:

1. What is function call in C++
2. What is function prototype
3. How to use function call is used with examples.
4. How to pass argument to function

Main Content

Function Prototype (Declaration)

3 min



Let us take note that if a user-defined function is defined after main() function, compiler will show error. It is because compiler is unaware of user-defined function, types of argument passed to function and return type.

SAQ 2

In C++, function prototype is a declaration of function without its body to give compiler information about user-defined function. Function prototype in the above example is:

```
int add(int, int);
```

You can see that, there is no body of function in prototype. Also, there are only return type of arguments but no arguments. You can also declare function prototype as below but it's not necessary to write arguments.

```
int add(int a, int b);
```

Note: It is not necessary to define prototype if user-defined function exists before main()function.

Function Call

SAQ 1

To execute the codes of function body, the user-defined function needs to be invoked(called). In the above program, add(num1,num2); inside main() function calls the user-defined function.

The function returns an integer which is stored in variable add.

Function Definition

The function itself is referred as function definition. Function definition in the above program is:

```
// Function definition

int add(int a,int b)

{

int add;

add = a + b;

return add;

}
```

When the function is called, control is transferred to the first statement of the function body.

Then, other statements in function body are executed sequentially.

When all codes inside function definition is executed, control of program moves to the calling program.

Passing Arguments to Function

In programming, argument (parameter) refers to the data which is passed to a function (function definition) while calling it.

In the above example, two variables, num1 and num2 are passed to function during function call. These arguments are known as actual arguments.

The value of num1 and num2 are initialized to variables a and b respectively. These arguments a and b are called formal arguments.

This is demonstrated in figure below:

Notes on passing arguments

- The numbers of actual arguments and formals argument should be the same. (Exception: Function Overloading)
- The type of first actual argument should match the type of first formal argument. Similarly, type of second actual argument should match the type of second formal argument and so on.
- You may call function a without passing any argument. The number(s) of argument passed to a function depends on how programmer want to solve the problem.
- You may assign default values to the argument. These arguments are known as default arguments.
- In the above program, both arguments are of int type. But it's not necessary to have both arguments of same type.

Return Statement

A function can return a single value to the calling program using return statement.

In the above program, the value of add is returned from user-defined function to the calling program using statement below:

```
return add;
```

The figure below demonstrates the working of return statement.

In the above program, the value of add inside user-defined function is returned to the calling function. The value is then stored to a variable sum.

Notice that the variable returned, i.e., add is of type int and sum is also of int type.

Also, notice that the return type of a function is defined in function declarator int add(int a, int b). The int before add(int a, int b) means the function should return a value of type int.

If no value is returned to the calling function then, void should be used.

Summary

At the end of the unit you have learned that;

- If a user-defined function is defined after main() function, compiler will show error.
- In C++, function prototype is a declaration of function without its body to give compiler information about user-defined function.
- To execute the codes of function body, the user-defined function needs to be invoked (called).
- A function can return a single value to the calling program using return statement.
- In programming, argument (parameter) refers to the data which is passed to a function (function definition) while calling it.
- A function can return a single value to the calling program using return statement.

Self-Assessment

SAQ

- Define a function call in C++
- What is function prototype

Tutor Marked Assessment

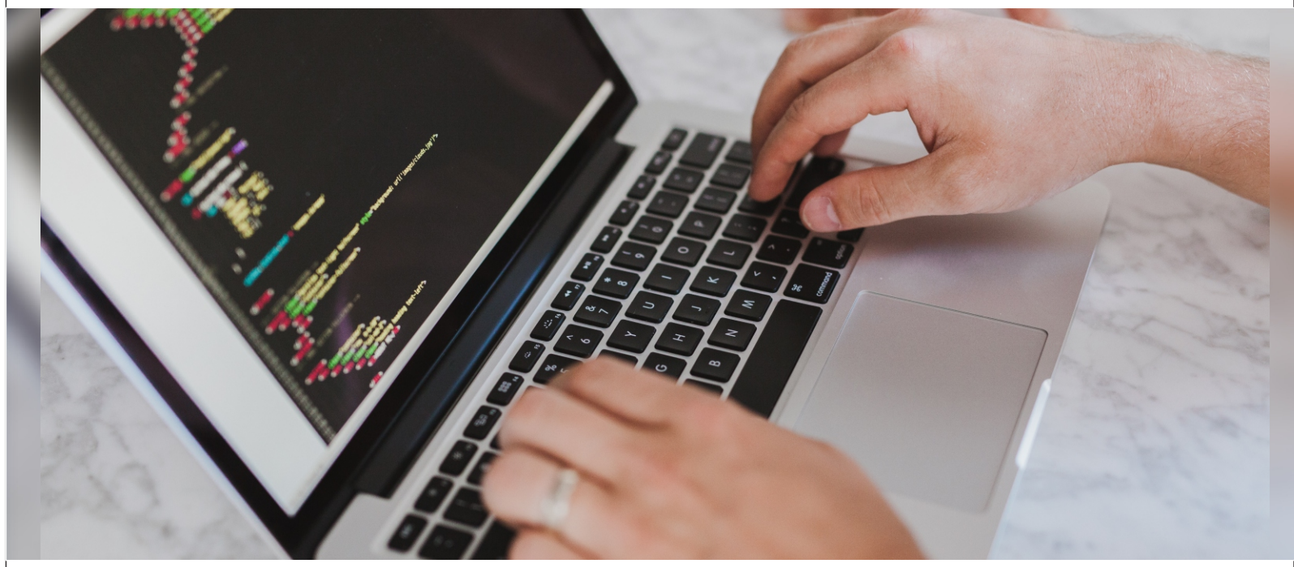
- State how to use function call is used with examples.
- Give example on how to pass argument to function

Further Reading

- https://codescracker.com › cpp › cpp-function-calling\
- https://www.includehelp.com › cpp-tutorial › cascaded-function-call
- https://stackoverflow.com › questions › how-to-call-a-c-class-method-whic...
- https://stackoverflow.com › questions › limits-for-function-call-in-c

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.



UNIT 3

Types of Function

Introduction

In the unit, I want you to learn arguments and return functions in C++ programming, the category of user-defined function. Finally, we will enumerate C++ function overloading.

Learning Outcomes

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

1. What is argument
2. How to define return function
3. State the category of user-defined functions
4. Give example of C++ function overloading

Main Content

Types of Under-defined Function in C++

5 min

For better understanding of arguments and return in functions, user-defined functions can be categorized as:

- Function with no argument and no return value
- Function with no argument but return value
- Function with argument but no return value
- Function with argument and return value

Consider a situation in which you have to check prime number. This problem is solved below by making user-defined function in 4 different ways as mentioned above.

Example 1: No arguments passed and no return value

```
# include <iostream>
using namespace std;
void prime();
int main()
{
   // No argument is passed to prime()
   prime();
   return 0;
}
// Return type of function is void because value is not returned.
void prime()
{
int num, i, flag = 0;
```



```
cout << "Enter a positive integer enter to check: ";
cin >> num;
for(i = 2; i <= num/2; ++i)
 {
if(num % i == 0)
{
flag = 1;
break;
}
}
if (flag == 1)
{
cout << num << " is not a prime number.";
}
 else
 {
 cout << num << " is a prime number."; }
}
```

In the above program, prime() is called from the main() with no arguments.

prime() takes the positive number from the user and checks whether the number is a prime number or not.

Since, return type of prime() is void, no value is returned from the function.

Example 2: No arguments passed but a return value

```
#include <iostream>
using namespace std;
int prime();
int main()
{
```

```
  int num, i, flag = 0;
// No argument is passed to prime()
num = prime();
for (i = 2; i <= num/2; ++i)
{
 if (num%i == 0)
 {
flag = 1;
break;
 }
}
 if (flag == 1)
 {
 cout<<num<<" is not a prime number.";
}
   else
{
cout<<num<<" is a prime number.";
}
return 0;
}
// Return type of function is int
int prime()
{
 int n;
printf("Enter a positive integer to check: ");
 cin >> n;
return n;
}
```


In the above program, prime() function is called from the main() with no arguments.

prime() takes a positive integer from the user. Since, return type of the function is an int, it returns the inputted number from the user back to the calling main() function.

Then, whether the number is prime or not is checked in the main() itself and printed onto the screen.

Example 3: Arguments passed but no return value

```
#include <iostream>
using namespace std;
void prime(int n);
int main()
{
 int num;
cout << "Enter a positive integer to check: ";
cin >> num;

// Argument num is passed to the function prime()
prime(num);
return 0;
}
// There is no return value to calling function. Hence, return type of function is void. */
void prime(int n)
{
int i, flag = 0;
for (i = 2; i <= n/2; ++i)
 {
 if (n%i == 0)
```

Which method is better?

All four programs above gives the same output and all are technically correct program.

There is no hard and fast rule on which method should be chosen.

The particular method is chosen depending upon the situation and how you want to solve a problem.

C++ Function Overloading

These functions having different number or type (or both) of parameters are known as overloaded functions. For example:

```
int test() { }

int test(int a) { }

float test(double a) { }

int test(int a, double b) { }
```

Here, all 4 functions are overloaded functions because argument(s) passed to these functions are different.

Notice that, the return type of all these 4 functions are not same. Overloaded functions may or may not have different return type but it should have different argument(s).

```
// Error code

int test(int a) { }

double test(int b){ }
```


The number and type of arguments passed to these two functions are same even though the return type is different. Hence, the compiler will throw error.

Example 1: Function Overloading

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

void display(int);
void display(float);
void display(int, float);

int main() {

  int a = 5;
  float b = 5.5;

  display(a);
  display(b);
  display(a, b);

  return 0;
}

void display(int var) {
  cout << "Integer number: " << var << endl;
}

void display(float var) {
  cout << "Float number: " << var << endl;
}
```

```
{
flag = 1;
 break;
}
 }
if (flag == 1)
{
cout << n << " is not a prime number.";
}
else {
 cout << n << " is a prime number.";
}
}
```

In the above program, positive number is first asked from the user which is stored in the variable num.

Then, num is passed to the prime() function where, whether the number is prime or not is checked and printed.

Since, the return type of prime() is a void, no value is returned from the function.

Example 4: Arguments passed and a return value.

```
#include <iostream>
using namespace std;
int prime(int n);
int main()
{
 int num, flag = 0;
 cout << "Enter positive integer to check: ";
cin >> num;
```



```
// Argument num is passed to check() function
 flag = prime(num);
if(flag == 1)
 cout << num << " is not a prime number.";
else
cout<< num << " is a prime number.";
return 0;
}
/* This function returns integer value.  */
int prime(int n)
{
int i;
for(i = 2; i <= n/2; ++i)
 {
 if(n % i == 0)
 return 1;
}
return 0;
}
```

In the above program, a positive integer is asked from the user and stored in the variable num.

Then, num is passed to the function prime() where, whether the number is prime or not is checked.

Since, the return type of prime() is an int, 1 or 0 is returned to the main() calling function. If the number is a prime number, 1 is returned. If not, 0 is returned.

Back in the main() function, the returned 1 or 0 is stored in the variable flag, and the corresponding text is printed onto the screen.

```
void display(int var1, float var2) {
   cout << "Integer number: " << var1;
   cout << " and float number:" << var2;
   }
```

Output

```
Absolute value of -5 = 5
Absolute value of 5.5 = 5.5
```

In the above example, two functions absolute() are overloaded.

Both functions take single argument. However, one function takes integer as an argument and other takes float as an argument.

When absolute() function is called with integer as an argument, this function is called:

```
int absolute(int var) {
   if (var < 0)
      var = -var;
   return var;
}
```

When absolute() function is called with float as an argument, this function is called:

```
float absolute(float var){
   if (var < 0.0)
      var = -var;
   return var;
}
```


Summary

At the end of this unit you have learned that:

- For better understanding of arguments and return in functions, user-defined functions can be categorized as:
  - Function with no argument and no return value
  - Function with no argument but return value
  - Function with argument but no return value
  - Function with argument and return value
- These functions having different number or type (or both) of parameters are known as overloaded functions.

Self-Assessment

SAQ

- Define argument with example
- Define a return function with example

Tutor Marked Assessment

1. State 2 the category of user-defined functions
2. Give 2 example of C++ function overloading

Further Reading

- https://www.studytonight.com/cpp/operator-overloading-examples.php
- https://www.programiz.com › cpp-programming › user-defined-function-t.
- https//www.ecomputernotes.com › C++ Programming › Functions
- https://www.dummies.com › programming › cpp › passing-arguments-to-f...

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.

Picture:
analysis

Photo: pixabay

Module 5

Composite Data Types

Units

Unit 1 - Array
Unit 2 - String
Unit 3 - Structure
Unit 4 - Union

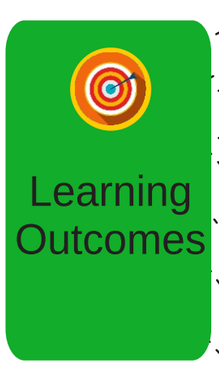
UNIT 1

Array

Introduction

In this unit we will discuss an array, how to declare an array, element of an array and how to access them, initialization of an array and how to insert and print array element. We will also discuss multidimensional arrays initialization of two, three dimensional array with their examples. Finally we will discuss passing array to a function with examples. Note that we have used variables to store values in memory for later reuse. We now explore a means to store multiple values together as one unit, the array.

An array is a fixed number of elements of the same type stored sequentially in memory. Therefore, an integer array holds some number of integers, a character array holds some
number of characters, and so on.



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

1. Define an array
2. Know how to work with arrays.
3. Declare, initialize and, access array elements in C++ programming.
4. Define multi-dimensional arrays in C++ and how todeclare them, access them and use them efficiently in your program.
5. Pass an array to a function in C++.
6. Pass both one-dimensional and multi-dimensional arrays.

Main Content

Array

8 min

In programming, one of the frequently arising problem is to handle numerous data of same type.

SAQ 1

Consider this situation, you are taking a survey of 100 people and you have to store their age. To solve this problem in C++, you can create an integer array having 100 elements.

An array is a collection of data that holds fixed number of values of same type. For example:

```
int age[100];
```

Here, the age array can hold maximum of 100 elements of integer typeThe size and type of arrays cannot be changed after its declaration.

How to declare an array in C++?

```
dataType arrayName[arraySize];
```

For example,

```
float mark[5];
```

Here, we declared an array, mark, of floating-point type and size 5. Meaning, it can hold 5 floating-point values.

Elements of an Array and How to access them?

You can access elements of an array by using indices.

Suppose you declared an array mark as above. The first element is mark[0], second element is mark[1] and so on.

Few key notes:

- Arrays have 0 as the first index not 1. In this example, mark[0] is the first element.
- If the size of an array is n, to access the last element, (n-1) index is used. In this example, mark[4] is the last element.
- Suppose the starting address of mark[0] is 2120d. Then, the next address, a[1], will be 2124d, address of a[2] will be 2128d and so on. It's because the size of float is 4 bytes.

How to initialize an array in C++ programming?

It's possible to initialize an array during declaration. For example,

```
int mark[5] = {19, 10, 8, 17, 9};
```

Another method to initialize array during declaration:

```
int mark[] = {19, 10, 8, 17, 9};
```

Here,

mark[0] is equal to 19

mark[1] is equal to 10

mark[2] is equal to 8

mark[3] is equal to 17

mark[4] is equal to 9

How to insert and print array elements?

```
int mark[5] = {19, 10, 8, 17, 9}

// change 4th element to 9
mark[3] = 9;

// take input from the user and insert in third element
cin >> mark[2];

// take input from the user and insert in (i+1)th element
cin >> mark[i];

// print first element of the array
cout << mark[0];
// print ith element of the array
cout >> mark[i-1];
```


Example: C++ Array

SAQ 3

C++ program to store and calculate the sum of 5 numbers entered by the user using arrays.

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

int main()
{
   int numbers[5], sum = 0;
   cout << "Enter 5 numbers: ";

   // Storing 5 number entered by user in an array
   // Finding the sum of numbers entered
   for (int i = 0; i < 5; ++i)
   {
      cin >> numbers[i];
      sum += numbers[i];
   }

   cout << "Sum = " << sum << endl;

   return 0;
}
```

Output

```
Enter 5 numbers: 3
4
5
4
2
Sum = 18
```

Things to remember when working with arrays in C++

Suppose you declared an array of 10 elements. Let's say,

```
int testArray[10];
```

You can use the array members from testArray[0] to testArray[9].

If you try to access array elements outside of its bound, let's say testArray[14], the compiler may not show any error. However, this may cause unexpected output (undefined behavior).

C++ Multidimensional Arrays

In C++, you can create an array of an array known as multi-dimensional array. For example:

```
int x[3][4];
```

Here, x is a two dimensional array. It can hold a maximum of 12 elements.

You can think this array as table with 3 rows and each row has 4 columns as shown below.

Three dimensional array also works in a similar way. For example:

```
float x[2][4][3];
```


This array x can hold a maximum of 24 elements. You can think this example as: Each of the 2 elements can hold 4 elements, which makes 8 elements and each of those 8 elements can hold 3 elements. Hence, total number of elements this array can hold is 24.

Multidimensional Array Initialisation

You can initialise a multidimensional array in more than one way.

Initialisation of two dimensional array

```
int test[2][3] = {2, 4, -5, 9, 0, 9};
```

Better way to initialise this array with same array elements as above.

```
int  test[2][3] = { {2, 4, 5}, {9, 0 0}};
```

Initialisation of three dimensional array

```
int test[2][3][4] = {3, 4, 2, 3, 0, -3, 9, 11, 23, 12, 23,
                     2, 13, 4, 56, 3, 5, 9, 3, 5, 5, 1, 4, 9};
```

Better way to initialise this array with same elements as above.

```
int test[2][3][4] = {
                { {3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2} },
                { {13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9} }
            };
```

Example 1: Two Dimensional Array

C++ Program to display all elements of an initialised two dimensional array.

SAQ 3

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

```
int main()
{
   int test[3][2] =
   {
      {2, -5},
      {4, 0},
      {9, 1}
   };

   // Accessing two dimensional array using
   // nested for loops
   for(int i = 0; i < 3; ++i)
   {
      for(int j = 0; j < 2; ++j)
      {
         cout<< "test[" << i << "][" << j << "] = " << test[i][j] << endl;
      }
   }

   return 0;
}
```

output

```
test[0][0] = 2
test[0][1] = -5
test[1][0] = 4
test[1][1] = 0
test[2][0] = 9
```


Example 2: Two Dimensional Array

C++ Program to store temperature of two different cities for a week and display it.

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

const int CITY = 2;
const int WEEK = 7;

int main()
{
   int temperature[CITY][WEEK];

   cout << "Enter all temperature for a week of first city and then second city. \n";

   // Inserting the values into the temperature array
   for (int i = 0; i < CITY; ++i)
   {
      for(int j = 0; j < WEEK; ++j)
      {
         cout << "City " << i + 1 << ", Day " << j + 1 << " : ";
         cin >> temperature[i][j];
      }
   }

   cout << "\n\nDisplaying Values:\n";

   // Accessing the values from the temperature array
```

```
    for (int i = 0; i < CITY; ++i)
    {
        for(int j = 0; j < WEEK; ++j)
        {
            cout << "City " << i + 1 << ", Day " << j + 1 << " = " << temperature[i][j]
<< endl;
        }
    }

    return 0;
}
```

output

```
Enter all temperature for a week of first city and then second city.
City 1, Day 1 : 32
City 1, Day 2 : 33
City 1, Day 3 : 32
City 1, Day 4 : 34
City 1, Day 5 : 35
City 1, Day 6 : 36
City 1, Day 7 : 38
City 2, Day 1 : 23
City 2, Day 2 : 24
City 2, Day 3 : 26
City 2, Day 4 : 22
City 2, Day 5 : 29
City 2, Day 6 : 27
City 2, Day 7 : 23
```



```
Displaying Values:
City 1, Day 1 = 32
City 1, Day 2 = 33
City 1, Day 3 = 32
City 1, Day 4 = 34
City 1, Day 5 = 35
City 1, Day 6 = 36
City 1, Day 7 = 38
City 2, Day 1 = 23
City 2, Day 2 = 24
City 2, Day 3 = 26
City 2, Day 4 = 22
City 2, Day 5 = 29
City 2, Day 6 = 27
City 2, Day 7 = 23
```

Example 3: Three Dimensional Array

C++ Program to Store value entered by user in three dimensional array and display it.

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

int main()
{
    // This array can store upto 12 elements (2x3x2)
    int test[2][3][2];

    cout << "Enter 12 values: \n";
```

```
// Inserting the values into the test array
// using 3 nested for loops.
for(int i = 0; i < 2; ++i)
{
   for (int j = 0; j < 3; ++j)
   {
      for(int k = 0; k < 2; ++k )
      {
         cin >> test[i][j][k];
      }
   }
}

cout<<"\nDisplaying Value stored:"<<endl;

// Displaying the values with proper index.
for(int i = 0; i < 2; ++i)
{
   for (int j = 0; j < 3; ++j)
   {
      for(int k = 0; k < 2; ++k)
      {
         cout << "test[" << i << "][" << j << "][" << k << "] = " << test[i][j][k]
<< endl;
      }
   }
}

   return 0;
}
```


output

```
Enter 12 values:
1
2
3
4
5
6
7
8
9
10
11
12

Displaying Value stored:
test[0][0][0] = 1
test[0][0][1] = 2
test[0][1][0] = 3
test[0][1][1] = 4
test[0][2][0] = 5
test[0][2][1] = 6
test[1][0][0] = 7
test[1][0][1] = 8
test[1][1][0] = 9
test[1][1][1] = 10
test[1][2][0] = 11
test[1][2][1] = 12
```

As the number of dimension increases, the complexity also increases

Passing Array to a Function in C++ Programming

Arrays can be passed to a function as an argument. Consider this example to pass one-dimensional array to a function:

Example 1: Passing One-dimensional Array to a Function

C++ Program to display marks of 5 students by passing one-dimensional array to a function.

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

void display(int marks[5]);

int main()
{
   int marks[5] = {88, 76, 90, 61, 69};
   display(marks);
   return 0;
}

void display(int m[5])
{
   cout << "Displaying marks: "<< endl;

   for (int i = 0; i < 5; ++i)
   {
      cout << "Student "<< i + 1 <<": "<< m[i] << endl;
   }
}
```


output

```
Displaying marks:
Student 1: 88
Student 2: 76
Student 3: 90
Student 4: 61
Student 5: 69
```

When an array is passed as an argument to a function, only the name of an array is used as argument.

```
display(marks);
```

Also notice the difference while passing array as an argument rather than a variable.

```
void display(int m[5]);
```

The argument marks in the above code represents the memory address of first element of array marks[5].

And the formal argument int m[5] in function declaration converts to int* m;. This pointer points to the same address pointed by the array marks.

That's the reason, although the function is manipulated in the user-defined function with different array name m[5], the original array marks is manipulated.

C++ handles passing an array to a function in this way to save memory and time.

Passing Multidimensional Array to a Function

Multidimensional array can be passed in similar way as one-dimensional array. Consider this example to pass two dimensional array to a function:

Example 2: Passing Multidimensional Array to a Function

C++ Program to display the elements of two dimensional array by passing it to a function.

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

void display(int n[3][2]);

int main()
{
   int num[3][2] = {
      {3, 4},
      {9, 5},
      {7, 1}

   };
   display(num);

   return 0;
}

void display(int n[3][2])
{

   cout << "Displaying Values: " << endl;
   for(int i = 0;  i < 3; ++i)
   {
      for(int j = 0; j < 2; ++j)
      {
         cout << n[i][j] << " ";
```



```
      }
   }
}
```

output

```
Displaying Values:
3 4 9 5 7 1
```

In the above program, the multi-dimensional array num is passed to the function display().

Inside, display() function, the array n (num) is traversed using a nested for loop.

The program uses 2 for loops to iterate over the elements inside a 2-dimensional array. If it were a 3-dimensional array, you should use 3 for loops.

Finally, all elements are printed onto the screen.

Note: Multidimensional array with dimension more than 2 can be passed in similar way as two dimensional array.

Summary

At end of this unit you have learned that:

- An array is a collection of data that holds fixed number of values of same type.
- You can access elements of an array by using indices.
- Suppose you declared an array mark as above. The first element is mark[0], second element is mark[1] and so on.
- Arrays have 0 as the first index not 1. In this example, mark[0] is the first element.
- If the size of an array is n, to access the last element, (n-1) index is used. In this example, mark[4] is the last element.
- Suppose the starting address of mark[0] is 2120d. Then, the next address, a[1], will be 2124d, address of a[2] will be 2128d and so on. It's because the size of float is 4 bytes.

Self Assessment

- What is an array in C++ Programming ?
- Give an example of one dimensional array.
- How do you declare an array in C++ programming ?

Tutor Marked Assessment

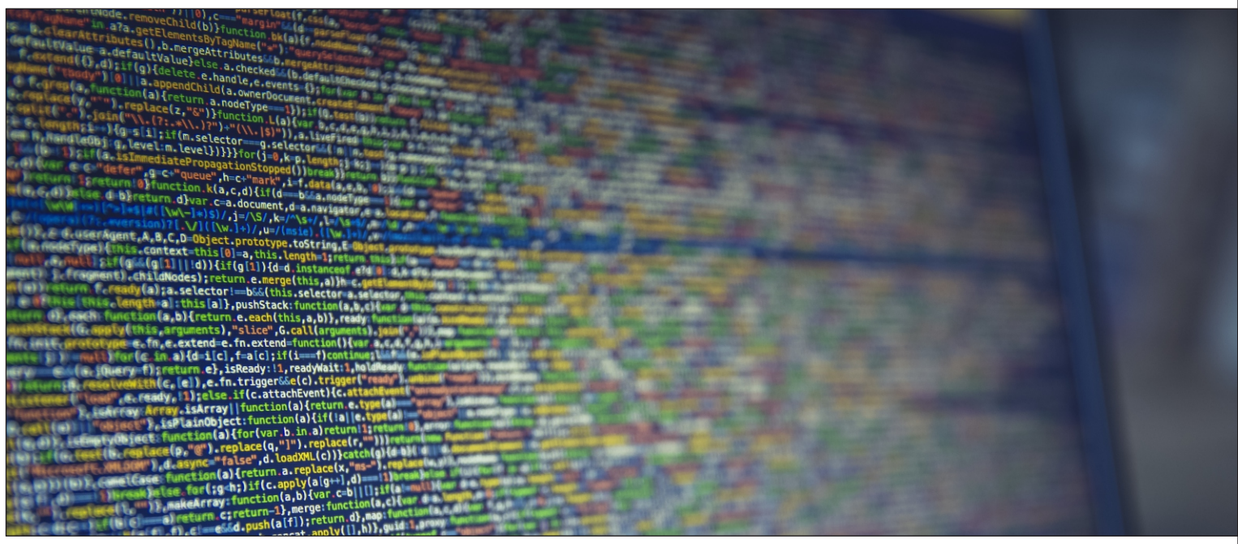
- Write C++ Program to display how the elements of two dimensional array can be pass to a function.
- Write a C++ program to find and print all common elements in three sorted arrays of integers.

Further Reading

- https://www.w3resource.com/cpp-exercises/array/cpp-array-exercise-25.php
- https://www.geeksforgeeks.org › arrays-in-c...
- https://www.tutorialspoint.com › cpp_arrays
- https://beginnersbook.com › cpp-arrays
- https://www.w3resource.com › cpp-exercises

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.



UNIT 2

String

Introduction

In this we going to discuss how to define a C-string with examples, we will also discuss string object and how to pass a string to function with some examples.

Sting literals such as “Hello, world!” are actually represented by C++ as a sequence of characters in memory. In other words, a string is simply a character array and can be manipulated as such.

Learning Outcomes

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

1. Handle strings in C.
2. Declare string, initialize and use them for various input/output operations.
3. State frequency occurrence of a character is checked for both (String object and C-style string)
4. Pass a string to a function with some examples

Main Content

C++ Strings

5 min

String is a collection of characters. There are two types of strings commonly used in C++ programming language:

SAQ 1

- Strings that are objects of string class (The Standard C++ Library string class)
- C-strings (C-style Strings)

C-strings

In C programming, the collection of characters is stored in the form of arrays, this is also supported in C++ programming. Hence it's called C-strings.

C-strings are arrays of type char terminated with null character, that is, \0 (ASCII value of null character is 0).

How to define a C-string?

```
char str[] = "C++";
```

In the above code, str is a string and it holds 4 characters.

Although, "C++" has 3 character, the null character \0 is added to the end of the string automatically.

Alternative ways of defining a string

```
char str[4] = "C++";

char str[] = {'C','+','+','\0'};

char str[4] = {'C','+','+','\0'};
```


Like arrays, it is not necessary to use all the space allocated for the string. For example:

```
char str[100] = "C++";
```

Example 1: C++ String to read a word

C++ program to display a string entered by user.

```
#include <iostream>
using namespace std;
int main()
{
  char str[100];

cout << "Enter a string: ";
  cin >> str;
  cout << "You entered: " << str << endl;

  cout << "\nEnter another string: ";
  cin >> str;
  cout << "You entered: "<<str<<endl;

  return 0;
}
```

output

```
Enter a string: C++
You entered: C++

Enter another string: Programming is fun.
You entered: Programming
```

Notice that, in the second example only "Programming" is displayed instead of "Programming is fun".

This is because the extraction operator >> works as scanf() in C and considers a space " " has a terminating character.

Example 2: C++ String to read a line of text

C++ program to read and display an entire line entered by user.

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

int main()
{
   char str[100];
   cout << "Enter a string: ";
   cin.get(str, 100);

   cout << "You entered: " << str << endl;
   return 0;
}
```

output

```
Enter a string: Programming is fun.
You entered: Programming is fun.
```

To read the text containing blank space, cin.get function can be used. This function takes two arguments.

First argument is the name of the string (address of first element of string) and second argument is the maximum size of the array.

In the above program, str is the name of the string and 100 is the maximum size of the array.

String Object

In C++, you can also create a string object for holding strings. Unlike using char arrays, string objects has no fixed length, and can be extended as per your requirement.

Example 3: C++ string using string data type

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

int main()
{
   // Declaring a string object
   string str;
   cout << "Enter a string: ";
   getline(cin, str);

   cout << "You entered: " << str << endl;
   return 0;
}
```

output

```
Enter a string: Programming is fun.
You entered: Programming is fun.
```

In this program, a string str is declared. Then the string is asked from the user.

Instead of using cin>> or cin.get() function, you can get the entered line of text using getline().

getline() function takes the input stream as the first parameter which is cin and str as the location of the line to be stored.

Passing String to a Function

Strings are passed to a function in a similar way arrays are passed to a function.

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

void display(char *);
void display(string);

int main()
{
  string str1;
  char str[100];

  cout << "Enter a string: ";
  getline(cin, str1);

  cout << "Enter another string: ";
  cin.get(str, 100, '\n');

  display(str1);
  display(str);
  return 0;
}

void display(char s[])
{
  cout << "Entered char array is: " << s << endl;
}

void display(string s)
{
  cout << "Entered string is: " << s << endl;
}
```

output

```
Enter a string:  Programming is fun.
Enter another string:  Really?
Entered string is: Programming is fun.
Entered char array is: Really?
```


In the above program, two strings are asked to enter. These are stored in str and str1respectively, where str is a char array and str1 is a string object.

Then, we have two functions display() that outputs the string onto the string.

The only difference between the two functions is the parameter. The first display() function takes char array as a parameter, while the second takes string as a parameter.

This process is known as function overloading.

C++ Program to Find the Frequency of Characters in a String

In this example, frequency of characters in a string object is computed.

To do this, size() function is used to find the length of a string object. Then, the for loop is iterated until the end of the string.

In each iteration, occurrence of character is checked and if found, the value of count is incremented by 1.

Example 1: Find Frequency of Characters of a String Object

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

int main()
{
  string str = "C++ Programming is awesome";
  char checkCharacter = 'a';
  int count = 0;

  for (int i = 0; i < str.size(); i++)
  {
    if (str[i] ==  checkCharacter)
    {
      ++ count;
    }
  }
```

output

```
Number of a = 2
```

In the example below, loop is iterated until the null character '\0' is encountered. Null character indicates the end of the string.

In each iteration, the occurrence of the character is checked.

Example 2: Find Frequency of Characters in a C-style String

```
#include <iostream>

using namespace std;
int main()
{
  char c[] = "C++ programming is not easy.", check = 'm';
  int count = 0;

  for(int i = 0; c[i] != '\0'; ++i)
  {
     if(check == c[i])
        ++count;
  }
  cout << "Frequency of " << check <<  " = " << count;
  return 0;
}
```

output

```
Number of m = 2
```

C++ Program to Find the Number of Vowels, Consonants, Digits and White Spaces in a String

Example 1: From a C-style string

This program takes a C-style string from the user and calculates the number of vowels, consonants, digits and white-spaces.


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

int main()
{
   char line[150];
   int vowels, consonants, digits, spaces;

   vowels =  consonants = digits = spaces = 0;

   cout << "Enter a line of string: ";
   cin.getline(line, 150);
   for(int i = 0; line[i]!='\0'; ++i)
   {
      if(line[i]=='a' || line[i]=='e' || line[i]=='i' ||
        line[i]=='o' || line[i]=='u' || line[i]=='A' ||
        line[i]=='E' || line[i]=='I' || line[i]=='O' ||
        line[i]=='U')
      {
         ++vowels;
      }
      else if((line[i]>='a'&& line[i]<='z') || (line[i]>='A'&& line[i]<='Z'))
      {
         ++consonants;
      }
      else if(line[i]>='0' && line[i]<='9')
      {
         ++digits;
      }
      else if (line[i]==' ')
      {
         ++spaces;
      }
   }

   cout << "Vowels: " << vowels << endl;
   cout << "Consonants: " << consonants << endl;
   cout << "Digits: " << digits << endl;
   cout << "White spaces: " << spaces << endl;
```

output

```
Enter a line of string: This is 1 hell of a book.
Vowels: 7
Consonants: 10
Digits: 1
White spaces: 6
```

Example 2: From a String Object

This program takes a string object from the user and calculates the number of

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

int main()
{
   string line;
   int vowels, consonants, digits, spaces;

   vowels =  consonants = digits = spaces = 0;

   cout << "Enter a line of string: ";
   getline(cin, line);

   for(int i = 0; i < line.length(); ++i)
   {
      if(line[i]=='a' || line[i]=='e' || line[i]=='i' ||
        line[i]=='o' || line[i]=='u' || line[i]=='A' ||
        line[i]=='E' || line[i]=='I' || line[i]=='O' ||
        line[i]=='U')
      {
         ++vowels;
      }
      else if((line[i]>='a'&& line[i]<='z') || (line[i]>='A'&& line[i]<='Z'))
      {
         ++consonants;
      }
      else if(line[i]>='0' && line[i]<='9')
      {
```



```
         ++digits;
      }
      else if (line[i]==' ')
      {
         ++spaces;
      }
   }

   cout << "Vowels: " << vowels << endl;
   cout << "Consonants: " << consonants << endl;
   cout << "Digits: " << digits << endl;
   cout << "White spaces: " << spaces << endl;

   return 0;
}
```

output

```
Enter a line of string: I have 2 C++ programming books.
Vowels: 8
Consonants: 14
Digits: 1
White spaces: 5
```

Summary

At the end of this unit you have learned that:

- String is a collection of characters.
- There are two types of strings commonly used in C++ programming language:
  - Strings that are objects of string class (The Standard C++ Library string class)
  - C-strings (C-style Strings)
- In C programming, the collection of characters is stored in the form of arrays, this is also supported in C++ programming.
- In C++, you can also create a string object for holding strings.

- In each iteration, occurrence of character is checked and if found, the value of count is incremented by 1.

Self Assessment

- What is a string?
- How do you declare a string in C++?
- Why do we use string in C++?

Tutor Marked Assessment

- State the different between a string and character array
- Write a program for a frequency of characters in a string object

Further Reading

- https://www.tutorialspoint.com › cpp_strings
- https://www.geeksforgeeks.org › c-string-cl..
- https://www.geeksforgeeks.org › c-string-class-and-its-applications
- https://www.w3schools.com › cpp_strings
- https://web.stanford.edu › archive › handouts

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.

UNIT 3

Structure

Introduction

In this unit, we want to look at structure in C++. How to define and declare it, how to access member of a structure with some example. We will also discuss structure and function, how to pass structure to function and returning structure from function. Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collection of data of different data types.

For example: You want to store some information about a person: his/her name, citizenship number and salary. You can easily create different variables name, citNo, salary to store these information separately.

However, in the future, you would want to store information about multiple persons. Now, you'd need to create different variables for each information per person: name1, citNo1, salary1, name2, citNo2, salary2

You can easily visualize how big and messy the code would look. Also, since no relation between the variables (information) would exist, it's going to be a daunting task.

A better approach will be to have a collection of all related information under a single name Person, and use it for every person. Now, the code looks much cleaner, readable and efficient as well.

This collection of all related information under a single name Person is a structure.

Learning Outcomes

In this Unit, you will learn:

1. How to define structures in C++ programming;
2. How to define it and use it in your program.
3. How to pass structures as an argument to a function, and use them in your program.

Main Content

Structure

5 min

SAQ 3

Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collection of data of different data types.

How to declare a structure in C++ programming?

SAQ 1

The struct keyword defines a structure type followed by an identifier (name of the structure).Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:

```
struct Person
{
  char name[50];
  int age;
  float salary;
};
```

Here a structure *person* is defined which has three members: *name*, *age* and *salary*.

When a structure is created, no memory is allocated.

The structure definition is only the blueprint for the creating of variables. You can imagine it as a datatype. When you define an integer as below:

```
int foo;
```


The *int* specifies that, variable *foo* can hold integer element only. Similarly, structure definition only specifies that, what property a structure variable holds when it is defined.

Note: Remember to end the declaration with a semicolon (;)

How to define a structure variable?

Once you declare a structure person as above. You can define a structure

```
Person bill;
```

Here, a structure variable *bill* is defined which is of type structure Person.

When structure variable is defined, only then the required memory is allocated by the compiler.

Considering you have either 32-bit or 64-bit system, the memory of *float* is 4 bytes, memory of *int* is 4 bytes and memory of char is 1 byte.

Hence, 58 bytes of memory is allocated for structure variable *bill*.

How to access members of a structure?

The members of structure variable is accessed using a **dot** (.) operator.

Suppose, you want to access *age* of structure variable *bill* and assign it 50 to it. You can perform this task by using following code below:

```
bill.age = 50;
```

Example: C++ Structure

C++ Program to assign data to members of a structure variable and display it

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

struct Person
{
  char name[50];
  int age;
  float salary;
};

int main()
```

```
{
   Person p1;

   cout << "Enter Full name: ";
   cin.get(p1.name, 50);
   cout << "Enter age: ";
   cin >> p1.age;
   cout << "Enter salary: ";
   cin >> p1.salary;

   cout << "\nDisplaying Information." << endl;
   cout << "Name: " << p1.name << endl;
   cout <<"Age: " << p1.age << endl;
   cout << "Salary: " << p1.salary;

   return 0;
}
```

output

```
Enter Full name: Magdalena Dankova
Enter age: 27
Enter salary: 1024.4

Displaying Information.
Name: Magdalena Dankova
Age: 27
Salary: 1024.4
```

Here a structure Person is declared which has three members *name*, *age* and *salary*.

Inside *main()* function, a structure variable *p1* is defined. Then, the user is asked to enter information and data entered by user is displayed.

C++ Structure and Function

Structure variables can be passed to a function and returned in a similar way as normal arguments.

Passing structure to function in C++

A structure variable can be passed to a function in similar way as normal argument. Consider this example:

Example 1: C++ Structure and Function

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

struct Person
{
   char name[50];
   int age;
   float salary;
};

void displayData(Person);   // Function declaration

int main()
{
   Person p;

   cout << "Enter Full name: ";
   cin.get(p.name, 50);
   cout << "Enter age: ";
   cin >> p.age;
   cout << "Enter salary: ";
   cin >> p.salary;

   // Function call with structure variable as an argument
   displayData(p);

   return 0;
}

void displayData(Person p)
{
   cout << "\nDisplaying Information." << endl;
```

```
    cout << "Name: " << p.name << endl;
    cout <<"Age: " << p.age << endl;
    cout << "Salary: " << p.salary;
}
```

output

```
Enter Full name: Bill Jobs
Enter age: 55
Enter salary: 34233.4

Displaying Information.
Name: Bill Jobs
Age: 55
Salary: 34233.4
```

In this program, user is asked to enter the *name*, *age* and *salary* of a Person inside main()function.

Then, the structure variable *p* is to passed to a function using.

```
displayData(p);
```

The return type of *displayData()* is void and a single argument of type structure *Person* is passed.

Then the members of structure p is displayed from this function.

Example 2: Returning structure from function in C++

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

struct Person {
    char name[50];
    int age;
    float salary;
};

Person getData(Person);
void displayData(Person);
```



```
int main()
{

    Person p;

    p = getData(p);
    displayData(p);

    return 0;
}

Person getData(Person p) {

    cout << "Enter Full name: ";
    cin.get(p.name, 50);

    cout << "Enter age: ";
    cin >> p.age;

    cout << "Enter salary: ";
    cin >> p.salary;

    return p;
}

void displayData(Person p)
{
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p.name << endl;
    cout <<"Age: " << p.age << endl;
    cout << "Salary: " << p.salary;
}
```

The output of this program is same as program above.

In this program, the structure variable *p* of type structure *Person* is defined under main()function.

The structure variable *p* is passed to *getData()* function which takes input from user which is then returned to main function.

```
p = getData(p);
```

Note: The value of all members of a structure variable can be assigned to another structure using assignment operator = if both structure variables are of same type. You don't need to manually assign each members.

Then the structure variable *p* is passed to *displayData()* function, which displays the information.

C++ Program to Add Complex Numbers by Passing Structure to a Function

This program takes two complex numbers as structures and adds them with the use of functions.

Example: Source Code to Add Two Complex Numbers

```
// Complex numbers are entered by the user

#include <iostream>
using namespace std;

typedef struct complex
{
  float real;
  float imag;
} complexNumber;

complexNumber addComplexNumbers(complex, complex);

int main()
{
  complexNumber n1, n2, temporaryNumber;
  char signOfImag;

  cout << "For 1st complex number," << endl;
  cout << "Enter real and imaginary parts respectively:" << endl;
  cin >> n1.real >> n1.imag;

  cout << endl << "For 2nd complex number," << endl;
  cout << "Enter real and imaginary parts respectively:" << endl;
```



```
  cin >> n2.real >> n2.imag;

  signOfImag = (temporaryNumber.imag > 0) ? '+' : '-';
      temporaryNumber.imag = (temporaryNumber.imag > 0) ?
temporaryNumber.imag : -temporaryNumber.imag;

  temporaryNumber = addComplexNumbers(n1, n2);
   cout << "Sum = "  << temporaryNumber.real << temporaryNumber.imag <<
"i";
  return 0;
}

complexNumber addComplexNumbers(complex n1,complex n2)
{
    complex temp;
    temp.real = n1.real+n2.real;
    temp.imag = n1.imag+n2.imag;
    return(temp);
}
```

output

```
Enter real and imaginary parts respectively:
3.4
5.5

For 2nd complex number,
Enter real and imaginary parts respectively:
-4.5
-9.5
Sum = -1.1-4i
```

In the problem, two complex numbers entered by the user is stored in structures *n1* and *n2*.

These two structures are passed to a*ddComplexNumbers()* function which calculates the sum and returns the result to the *main()* function.

Finally, the sum is displayed from the *main()* function.

Summary

At the end of this unit you have learned that:

- Structure is a collection of variables of different data types under a single name.
- The struct keyword defines a structure type followed by an identifier (name of the structure).
- A structure variable can be passed to a function in similar way as normal argument.

Self Assessment

- What is structure in C++ Programming Language
- What is the basic structure of C++ program?
- How do you define structure?
- What is the basic structure of C++ program?

Tutor Marked Assessment

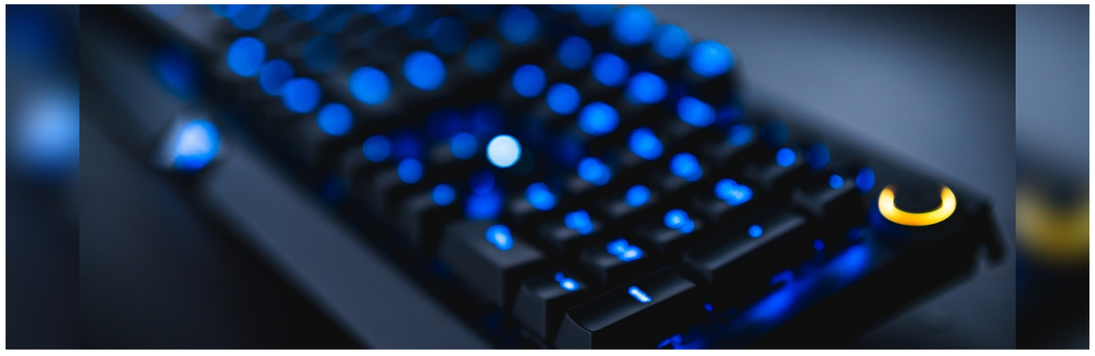
- What is the difference between C and C++ structures
- Write C++ program to assign data to members of a structure variable and display it.

Further Reading

- https://en.m.wikipedia.org › wiki › Struct_(C_programming_language)
- www.cplusplus.com › doc › tutorial › struct..
- https://study.com › academy › lesson › how..
- https://www.tutorialspoint.com › cplusplus
- https://simplesnippets.tech › structures-in-cpp

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.



UNIT 4

Union

Introduction

I welcome you to another unit, we will discuss Union and its declaration, the difference between structure and Union, syntax for declaring union and syntax for creating object of union with some examples. Union is also like a Structure means Unions is also used for Storing data of different data types But the Difference between Structure and Union is that Structure Consume the Memory of addition of all elements of Different Data Types but a Union Consume the Memory that is Highest among all variables.

It Consume Memory of highest variables and it will share the data among all the other Variables Suppose that if a union Contains variables Like Int ,Float ,Char then the Memory Will be consumed of Float variable because float is highest among all variables of data types etc. We can declare a Union in a Structure and Vice-versa.

A union is defined in the same way as a structure by specifying the keyword union in place of keyword struct. Both structure and union are collection of different datatype. They are used to group number of variables of different type in a single unit.

Learning Outcomes

On completion of this unit, you should be able to:

1. Declare union
2. State difference between union and structure
3. Write syntax for declaring union and
4. Create object of union with some examples.
5. Enumerate how to use union in your program.

Main Content

Union

3 min

Difference between structure and union

SAQ 1

1. Declaration and Initialization of structure starts with struct keyword. Declaration and Initialization of union starts with union keyword.
2. Structure allocates different memory locations for all its members while union allocates common memory location for all its members. The memory occupied by a union will be large enough to hold the largest member of the union.

Union Declaration Union declaration

Declaration of union must start with the keyword union followed by the union name and union's member variables are declared within braces.

Example of Syntax for declaring union

```
union union-name
{
    datatype var1;
    datatype var2;
    ----------
    ----------
    datatype varN;
};
```


Accessing the union members

We have to create an object of union to access its members. Object is a variable of type union. Union members are accessed using the dot operator(.) between union's object and union's member name.

Example of Syntax for creating object of union

```
union union-name obj;
```

Example for creating object & accessing union members

```
#include<iostream.h>

union Employee
{
    int Id;
    char Name[25];
    int Age;
    long Salary;
};

void main()
{

    Employee E;

        cout << "\nEnter Employee Id : ";
        cin >> E.Id;

        cout << "\nEnter Employee Name : ";
        cin >> E.Name;

        cout << "\nEnter Employee Age : ";
        cin >> E.Age;

        cout << "\nEnter Employee Salary : ";
        cin >> E.Salary;
```

```
        cout << "\n\nEmployee Id : " << E.Id;
        cout << "\nEmployee Name : " << E.Name;
        cout << "\nEmployee Age : " << E.Age;
        cout << "\nEmployee Salary : " << E.Salary;
}
```

output

```
Enter Employee Id : 1
        Enter Employee Name : Kumar
        Enter Employee Age : 29
        Enter Employee Salary : 45000

        Employee Id : -20536
        Employee Name : ?$?$ ?
        Employee Age : -20536
        Employee Salary : 45000
```

In the above example, we can see that values of Id, Name and Age members of union got corrupted because final value assigned to the variable has occupied the memory location and this is the reason that the value of salary member is getting printed very well. Now let's look into the same example once again where we will use one variable at a time which is the main purpose of having union:

```
#include<iostream.h>

union Employee
{
    int Id;
    char Name[25];
    int Age;
    long Salary;
};

void main()
{

    Employee E;
```



```
    cout << "\nEnter Employee Id : ";
    cin >> E.Id;
    cout << "Employee Id : " << E.Id;

    cout << "\n\nEnter Employee Name : ";
    cin >> E.Name;
    cout << "Employee Name : " << E.Name;

    cout << "\n\nEnter Employee Age : ";
    cin >> E.Age;
    cout << "Employee Age : " << E.Age;

    cout << "\n\nEnter Employee Salary : ";
    cin >> E.Salary;
    cout << "Employee Salary : " << E.Salary;

}
```

output

```
Enter Employee Id : 1
Employee Id : 1

Enter Employee Name : Kumar
Employee Name : Kumar

Enter Employee Age : 29
Employee Age : 29

Enter Employee Salary : 45000
Employee Salary : 45000
```

Here, all the members are getting printed very well because one member is being used at a time.

```
#include<iostream.h>

 struct Employee1
 {
int Id;
```

```
 char Name[25];
 long Salary;
  };

  union Employee2
  {
 int Id;
 char Name[25];
 long Salary;
  };

   void main()
   {

 cout << "\nSize of Employee1 is : " << sizeof(Employee1);
 cout << "\nSize of Employee2 is : " << sizeof(Employee2);

   }
```

output

```
Size of Employee1 is : 31
 Size of Employee2 is : 25
```

Summary

At the end of this unit you have learned that:

- Union is also like a Structure means unions is also used for Storing data of different data types.
- Difference between Structure and Union is that Structure Consume the Memory of addition of all elements of Different Data Types but a Union Consume the Memory that is Highest among all variables.
- Declaration of union must start with the keyword union followed by the union name and union's member variables are declared within braces.

Syntax for declaring union

```
union union-name
{
    datatype var1;
    datatype var2;
    ----------
    ----------
    datatype varN;
};
```

Self Assessment

- What is union PPL?
- Explain the difference between union and structure

Tutor Marked Assessment

- What is the difference between union and anonymous union?
- Why do we use union?

Further Reading

- https://docs.microsoft.com › cpp › cpp › un.
- https://stackoverflow.com › questions › anonymous-union-and-a-normal-uni...
- https://study.com › academy › unions-in-c-programming-definition-examplehttps://www.geeksforgeeks.org › difference-...
- ecomputernotes.com › what-is-union-in-c

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.

Picture:
Data science programming

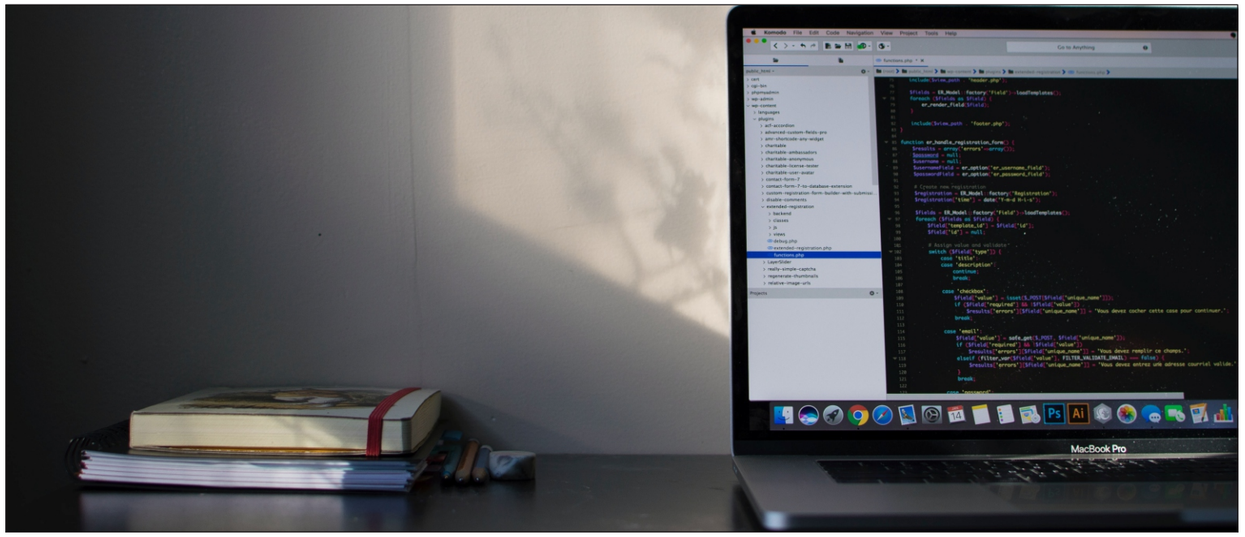
Photo: freepik.com

Module 6

Object Oriented Programming Concept in C++

Units

Unit 1 - OOP Concepts
Unit 2 - Classes and Object
Unit 3 - C++ Constructors



UNIT 1

OOP Concepts

Introduction

In this unit the concept of Object-Oriented Programming (OOP) will be discussed. OOP is a relatively new programming technology involving concepts that are radically different from those in conventional procedural programming.

It is a new way to think about programs and their structure and a new way to write programs. It is a programming technology with a specific definition and a specific set of features. It finds ready application in the development of Graphic User interface (GUI) and event-driven applications. We discuss the origin of OOP, it features, objects, some commonly use terms in OOP. Finally, features of Procedural Programming Language its characteristic and differences with OOP were also enumerated.

Learning Outcomes

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

1. Define OOP
2. Enumerate the features of OOP
3. State the difference between OOP and procedural programming language
4. Write short notes on some commonly used term in OOP

Main Content

Origin of OOP

1 min

Object-oriented programming originated in late 1960s with Simula-67. The concepts in Simula-67 influenced work at XEROX Palo Alto Center (P ARC) in the early 1970s leading to Smalltalk-72, Small-76, several versions of smalltalk-80, and the independently developed smalltalk/V7 and MIT which is a parallel OOP development tool.

SAQ 1

The concept of OOP has evolved across these 25 year or more years mainly out of sight of the mainstream of computing systems development. The first real exposure of the new ideas to the computer world happened in the now classic August 1981 Edition of Byte magazine which is dedicated to smalltalk-80. This was followed by a book "The Language and its implementation "published in 1983. Subsequent publications and development of various Object-oriented programming environments have since followed such that today there exist many applications based on object-oriented programming technology. e.g. object-oriented databases, and object oriented programming languages like Smalltalk, Actor, C++, object Pascal, C#, Java etc.

The basic idea of object-oriented programming is the that Data and the routines to process the data are combined into a single entity called a class. If you want to access the data in the class, you use the routines from the class.

Example

Class student_ record

NAME AGE SEX ADDRESS MATRICNO. COURSES	DATA
STORE( ) CLACULATE( ) PRINT()	ROUTINES OR FUNCTIONS

With the older style, procedural programming. The data and routines are separate and thought of separately. With OOP we consider the different entities in the program as objects and we examine the relationship between

them. The basic idea behind OOP is to break up the problem into a group of objects. Objects contains data and the routines that process the data.

Features or Characteristics of OOP.

SAQ 2

For a programming language to qualify as object-oriented it must has four basic characteristics namely:

Data Hiding (Encapsulation)

Encapsulation means that data and the code that manipulates it are defined together, that the data cannot be separated from or accessed separately from the associated code. The data is encapsulated within the code such that those portions of a program that are not part of a given definition cannot access any of the data, within the definition.

The data and the function that process the data are formed into a single entity called a class. The data are called data members of the class and the functions are called member functions of the class. The challenge of OOP is to design objects that accurately model the real world.

Inheritance

Most object-oriented languages define a new object by writing a class. A class consist of the code and the definition of the data for a single kind of object. Class is usually related to some real world things or concepts. New objects are defined by first a general concept and then refining the concept into a hierarchy of definitions. Definitions lower in hierarchy are said to inherit from definitions above, this characteristic is called inheritance. This brings about the creating of new objects by expanding existing ones.

When you create a new class from an existing class, the new class is called a derived class. The previously existing class is the base class. (sometimes derived classes are called children and base classes are called parents. The terms subclass (child) and superclass (parent) are used. The act of creating a derived class is called subclassing.

Polymorphism

This is means that more than one routine has the same name. The routine must belong to different kinds of objects. This makes it possible for different objects to react differently to the same message. Function names are considered unique and local to their class. e.g. "Print()" can take different meaning in different objects.

Single Type

Variables in fully object-oriented languages have a single type, an object. Any variable can hold an object. Hence it is possible to have a sort routine that sorts any kind of data.

Code can be written that depends only on the messages sent to an object and not on the kind of object.

Object

A collection of data and its associated subroutine is called an object. In OOP a subroutine is referred to as a method and a subroutine call referred to as message Object can be things, concepts, relationships, actions or whatever you can conceive of, write code for, and identify data for e.g. a car engine, sales call record, medical record, student record etc. Simply, an object is anything in the computer which has a data (characteristics) and methods (operation or action).

Example

Object: student record

Method: print record

Data: name, sex, age, marital status, level etc.

Parts of an Object

Objects have two parts these are information and operation. Information is sometimes called data. It tells something about the object but not how the object acts. Operations sometimes called instruction or programs or procedures or actions or methods tells how objects acts; what it will do when something happens.

Object Membership and Containership

The variables in an object are also called the member of that object. An object contains other objects. Therefore a parent child relationship exists in such a situation. When a parent object is removed from memory its child objects are also removed from memory.

Key Terms in OOP

Method

A subroutine which belongs to an object and which tells how to perform some action.

Class

It is a specification or template, for creating multiple copies of a particular object. It is a recipe for building objects. It defines the variables that are in the object and holds the methods for each of the objects.

Instance

This is an object that holds data and a reference (pointer) to the class. It must have this reference to the class since the class has the methods.

Message

This is similar to a subroutine call. Asking an object to do something is called "sending a message".

Properties

These are the memory variables contained in an object. When codes are attached to some properties it becomes a method.

OOP versus Procedural Programming

Procedural programming is the term used to describe programming done in conventional programming languages such as C, Pascal, Fortran or Cobol, etc.

Features of Procedural Languages

(i) Code and data are considered as separate ;

(ii) Data types are built into the language;

(iii) Programs are designed around code structure; and

(iv) Related function is distributed through modules.

Differences between Procedural Languages and Object- Oriented Programming

(i) Code and data are defined separately unlike in Object-oriented languages where they are together.

(ii) The names of procedures and global must be unique, and must not conflict with each other hence does not support polymorphism as in object-oriented languages.

(iii) Data types in procedural languages must be one of the built in predefined types, or arrays or structures of these types. There is more than one data type unlike in Object-Oriented languages where a single type (object exist).

(iv) The structures of procedural languages are code-based and not data-based as in object-oriented languages. The structure of the program depends on the use envisioned when the user who first wrote it.

(v) There is Distributed functionality in procedural languages i.e. routines knows so much about other routines such that changes to one force changes in other routines.

(vi) Codes written in procedural languages are not very reusable i.e. very little code reuse. For example a code sort integer values cannot be used to sort real values with any modification. Object-oriented languages allows for code reusability.

Summary

At the end of this unit you have learnt that:

- OOP is a relatively new programming technology involving concepts that are radically different from those in conventional procedural programming.
- Object-oriented programming originated in late 1960s with Simula-67. The concepts in Simula-67 influenced work at XEROX Palo Alto Center (P ARC) in the early 1970s leading to Smalltalk-72, Small-76, several versions of smalltalk-80, and the independently developed smalltalk/V7 and MIT which is a parallel OOP development tool.
- The basic idea of object-oriented programming is the Data and the routines to process the data are combined into a single entity called a class.

Self-Assessment

SAQ

- Define OOP
- Enumerate five (5) features of OOP

Tutor Marked Assessment

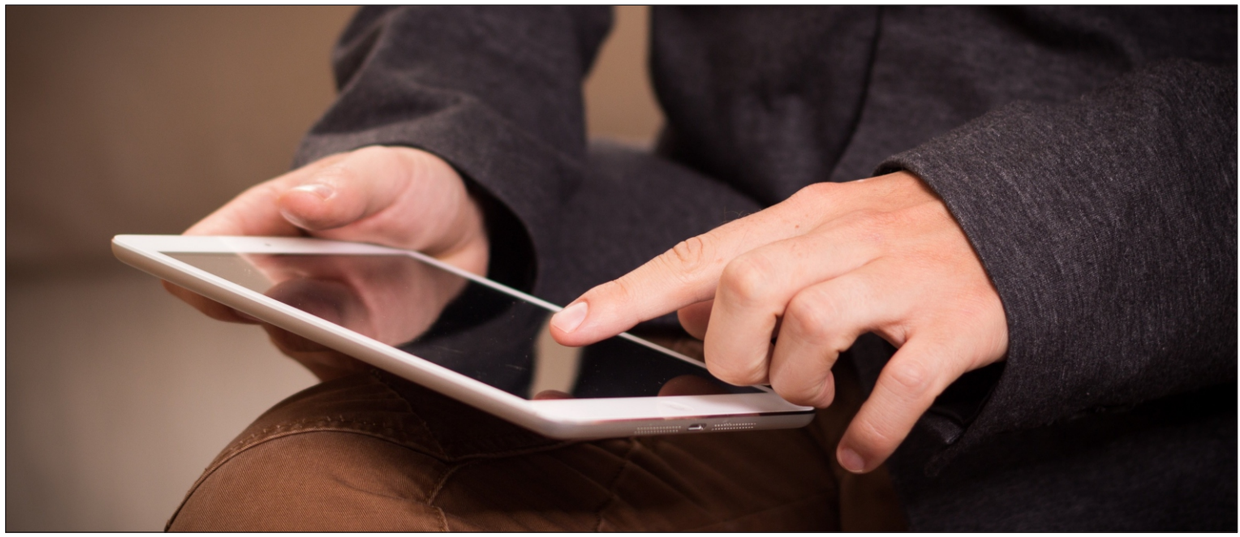
- State five (5) main difference between OOP and procedural programming language
- Write shorts note on five (5) of the following OOP terms
  (a)Inheritance
  (b)Polymorphism
  (c)Encapsulation
  (d)Single type
  (e)Instance
  (f)Object
  (g)Methods

Further Reading

- https://www.edureka.co › blog › object-oriented-programming
- https://blog.usejournal.com › object-oriented-programming-concepts-in-si...
- https://www.studytonight.com › cpp › cpp-and-oops-concepts
- https://www.geeksforgeeks.org › object-oriented-programming-oops-conc...
- https://en.wikipedia.org › wiki › Object-oriented_programming

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.



UNIT 2

Classes and Object

Introduction

In this unit, we will discuss how to work with object and classes, how to define a class with examples and some keywords. Further discussion on syntax to define object and how to assess data member and member function in C++ with example will be fully discussed. C++ is a multi-paradigm programming language. Meaning, it supports different programming styles.

One of the popular ways to solve a programming problem is by creating objects, known as object-oriented style of programming.

C++ supports object-oriented (OO) style of programming which allows you to divide complex problems into smaller sets by creating objects.

Object is simply a collection of data and functions that act on those data.

Learning Outcomes

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

1. Work with object and classes.
2. Define a class with example
3. Access data member and member function in C++
4. State the syntax to define object.

Main Content

CLASS

3 min

Before you create an object in C++, you need to define a class.

A class is a blueprint for the object.

We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.

As, many houses can be made from the same description, we can create many objects from a class.

How to define a class in C++?

A class is defined in C++ using keyword class followed by the name of class.

The body of class is defined inside the curly brackets and terminated by a semicolon at the end.

SAQ 1

```
class className
  {
  // some data
  // some functions
  };
```

Example: Class in C++

```
class Test
{
   private:
      int data1;
      float data2;

   public:
      void function1()
      {  data1 = 2; }
      float function2()
      {
         data2 = 3.5;
         return data2;
      }
};
```


Here, we defined a class named Test.

This class has two data members: data1 and data2 and two member functions: function1()and function2().

Keywords: private and public

You may have noticed two keywords: private and public in the above example.

The private keyword makes data and functions private. Private data and functions can be accessed only from inside the same class.

The public keyword makes data and functions public. Public data and functions can be accessed out of the class.

Here, data1 and data2 are private members where as function1() and function2() are public members.

If you try to access private data from outside of the class, compiler throws error. This feature in OOP is known as data hiding.

Object

When class is defined, only the specification for the object is defined; no memory or storage is allocated.

To use the data and access functions defined in the class, you need to create

Syntax to Define Object in C++

SAQ 2

```
className objectVariableName;
```

You can create objects of Test class (defined in above example) as follows:

```
class Test
{
   private:
      int data1;
```

```
    float data2;

  public:
    void function1()
    {  data1 = 2; }

    float function2()
    {
      data2 = 3.5;
      return data2;
    }
 };

int main()
{
  Test o1, o2;
}
```

Here, two objects o1 and o2 of Test class are created.

In the above class Test, data1 and data2 are data members and function1() and function2() are member functions.

How to access data member and member function in C++?

You can access the data members and member functions by using a . (dot) operator. For example,

```
o2.function1();
```

This will call the function1() function inside the Test class for objects o2.

Similarly, the data member can be accessed as:

```
o1.data2 = 5.5;
```

It is important to note that, the private members can be accessed only from inside the class.

So, you can use o2.function1(); from any function or class in the above example. However, the code o1.data2 = 5.5; should always be inside the class Test.

Example: Object and Class in C++ Programming

```
// Program to illustrate the working of objects and class in C++ Programming
#include <iostream>
using namespace std;

class Test
{
  private:
    int data1;
    float data2;

  public:

    void insertIntegerData(int d)
    {
      data1 = d;
      cout << "Number: " << data1;
    }

    float insertFloatData()
    {
      cout << "\nEnter data: ";
      cin >> data2;
      return data2;
    }
};

int main()
{
   Test o1, o2;
   float secondDataOfObject2;

   o1.insertIntegerData(12);
   secondDataOfObject2 = o2.insertFloatData();

   cout << "You entered " << secondDataOfObject2;
   return 0;
}
```

output

```
Number: 12
Enter data: 23.3
You entered 23.3
```

In this program, two data members data1 and data2 and two member functions insertIntegerData() and insertFloatData() are defined under Test class.

Two objects o1 and o2 of the same class are declared.

The insertIntegerData() function is called for the o1 object using:

```
o1.insertIntegerData(12);
```

This sets the value of data1 for object o1 to 12.

Then, the insertFloatData() function for object o2 is called and the return value from the function is stored in variable secondDataOfObject2 using:

```
secondDataOfObject2 = o2.insertFloatData();
```

In this program, data2 of o1 and data1 of o2 are not used and contains garbage value.

You should also check these topics to learn more on objects and classes:

Summary

In this unit you have learned that:

- C++ is a multi-paradigm programming language. Meaning, it supports different programming styles.
- One of the popular ways to solve a programming problem is by creating objects, known as object-oriented style of programming.
- C++ supports object-oriented (OO) style of programming which allows you to divide complex problems into smaller sets by creating objects.
- A class is defined in C++ using keyword class followed by the name of class.
- The body of class is defined inside the curly brackets and terminated by a semicolon at the end.


```
class className
{
// some data
// some functions
};
```

Self-Assessment

- What is Class in C++ Programming Language
- Write out the syntax to define an object

Tutor Marked Assessment

- State the major different between a classes and object in C++
- Why do we use class in C++?

Further Reading

- https://www.w3schools.com › cpp › cpp_classes
- https://www.w3schools.in › cplusplus-tutorial › objects-classes
- https://www.geeksforgeeks.org › c-classes-and-objects
- https://www.studytonight.com › cpp › class-and-objects
- https://www.programiz.com › cpp-programming › object-class

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.

UNIT 3

C++ Constructors

Introduction

In this unit we are going to discuss how a constructor works, uses of constructor with specific examples. Constructor overloading with example will be examined. Default copy constructor with prefix, postfix, operator overloading of decrement operator with some examples will enumerated. A constructor is a special type of member function that initialises an object automatically when it is created.

Compiler identifies a given member function is a constructor by its name and the return type.

Learning Outcomes

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

1. Know how a constructor worked in C++.
2. Define a constructor,
3. State how to create a constructor
4. Enumerate the types of constructors in C++.

Main Content

Constructors

7 min

Constructor has the same name as that of the class and it does not have any return type. Also, the constructor is always public.

SAQ 1

```
……
class temporary
{
private:
int x;
float y;
public:
// Constructor
temporary(): x(5), y(5.5)
{
// Body of constructor
}
…… …
};

int main()
{
Temporary t1;
……
}
```

Above program shows a constructor is defined without a return type and the same name as the class.

How constructor works?

In the above pseudo code, temporary() is a constructor.

SAQ 2

When an object of class temporary is created, the constructor is called automatically, and xis initialized to 5 and y is initialized to 5.5.

You can also initialise the data members inside the constructor's body as below. However, this method is not preferred.


```
temporary()
{
  x = 5;
  y = 5.5;
}
// This method is not preferred.
```

Use of Constructor in C++

Suppose you are working on 100's of Person objects and the default value of a data member age is 0. Initialising all objects manually will be a very tedious task.

Instead, you can define a constructor that initialises age to 0. Then, all you have to do is create a Person object and the constructor will automatically initialise the age.

These situations arise frequently while handling array of objects.

Also, if you want to execute some code immediately after an object is created, you can place the code inside the body of the constructor.

Example 1: Constructor in C++

Calculate the area of a rectangle and display it.

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

class Area
{
  private:
    int length;
    int breadth;

  public:
    // Constructor
    Area(): length(5), breadth(2){}

    void GetLength()
```

```
    {
      cout << "Enter length and breadth respectively: ";
      cin >> length >> breadth;
    }

    int AreaCalculation() { return (length * breadth); }

    void DisplayArea(int temp)
    {
      cout << "Area: " << temp;
    }
};

int main()
{
  Area A1, A2;
  int temp;

  A1.GetLength();
  temp = A1.AreaCalculation();
  A1.DisplayArea(temp);

  cout << endl << "Default Area when value is not taken from user" << endl;

  temp = A2.AreaCalculation();
  A2.DisplayArea(temp);

  return 0;
}
```

In this program, class Area is created to handle area related functionalities. It has two data members length and breadth.

A constructor is defined which initialises length to 5 and breadth to 2.

We also have three additional member functions GetLength(), AreaCalculation() and DisplayArea() to get length from the user, calculate the area and display the area respectively.

When, objects A1 and A2 are created, the length and breadth of both objects are initialized to 5 and 2 respectively, because of the constructor.

Then, the member function GetLength() is invoked which takes the value of length and breadth from the user for object A1. This changes the length and breadth of the object A1.

Then, the area for the object A1 is calculated and stored in variable temp by calling AreaCalculation() function and finally, it is displayed.

For object A2, no data is asked from the user. So, the length and breadth remains 5 and 2 respectively.

Then, the area for A2 is calculated and displayed which is 10.

Output

```
Enter length and breadth respectively: 6
7
Area: 42
Default Area when value is not taken from user
Area: 10
```

Constructor Overloading

Constructor can be overloaded in a similar way as function overloading.

Overloaded constructors have the same name (name of the class) but different number of arguments.

Depending upon the number and type of arguments passed, specific constructor is called.

Since, there are multiple constructors present, argument to the constructor should also be passed while creating an object.

Example 2: Constructor overloading

```
// Source Code to demonstrate the working of overloaded constructors
#include <iostream>
using namespace std;
```

```
class Area
{
  private:
    int length;
    int breadth;

  public:
    // Constructor with no arguments
    Area(): length(5), breadth(2) {}

    // Constructor with two arguments
    Area(int l, int b): length(l), breadth(b){}

    void GetLength()
    {
       cout << "Enter length and breadth respectively: ";
       cin >> length >> breadth;
    }

    int AreaCalculation() { return length * breadth; }

    void DisplayArea(int temp)
    {
       cout << "Area: " << temp << endl;
    }
};

int main()
{
  Area A1, A2(2, 1);
  int temp;

  cout << "Default Area when no argument is passed." << endl;
  temp = A1.AreaCalculation();
  A1.DisplayArea(temp);

  cout << "Area when (2,1) is passed as argument." << endl;
  temp = A2.AreaCalculation();
  A2.DisplayArea(temp);
```



```
  return 0;
}
```

For object A1, no argument is passed while creating the object.

Thus, the constructor with no argument is invoked which initialises length to 5 and breadthto 2. Hence, area of the object A1 will be 10.

For object A2, 2 and 1 are passed as arguments while creating the object.

Thus, the constructor with two arguments is invoked which initialises length to l (2 in this case) and breadth to b (1 in this case). Hence, area of the object A2 will be 2.

Output

```
Default Area when no argument is passed.
Area: 10
Area when (2,1) is passed as argument.
Area: 2
```

Default Copy Constructor

An object can be initialized with another object of same type. This is same as copying the contents of a class to another class.

In the above program, if you want to initialise an object A3 so that it contains same values as A2, this can be performed as:

```
....
int main()
{
  Area A1, A2(2, 1);

  // Copies the content of A2 to A3
  Area A3(A2);
   OR,
  Area A3 = A2;
}
```

You might think, you need to create a new constructor to perform this task. But, no additional constructor is needed. This is because the copy constructor is

already built into all classes by default.

increment ++ and Decrement -- Operator Overloading in C++ Programming

In this example, you'll learn to overload increment ++ and decrement -- operators in C++.

In this tutorial, increment ++ and decrements -- operator are overloaded in best possible way, i.e., increase the value of a data member by 1 if ++ operator operates on an object and decrease value of data member by 1 if -- operator is used.

Example 1: Prefix ++ Increment Operator Overloading with no return type

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

class Check
{
  private:
    int i;
  public:
    Check(): i(0) { }
    void operator ++()
      { ++i; }
    void Display()
      { cout << "i=" << i << endl; }
};

int main()
{
  Check obj;

  // Displays the value of data member i for object obj
  obj.Display();

  // Invokes operator function void operator ++( )
  ++obj;

  // Displays the value of data member i for object obj
```



```
  obj.Display();

  return 0;
}
```

Output

```
i=0
i=1
```

Initially when the object obj is declared, the value of data member i for object obj is 0 (constructor initializes i to 0).

When ++ operator is operated on obj, operator function void operator++( ) is invoked which increases the value of data member i to 1.

This program is not complete in the sense that, you cannot used code:

```
obj1 = ++obj;
```

It is because the return type of operator function in above program is void.

Here is the little modification of above program so that you can use code `obj1 = ++obj`.

Example 2: Prefix Increment ++ operator overloading with return type

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

class Check
{
 private:
   int i;
 public:
   Check(): i(0) { }

   // Return type is Check
   Check operator ++()
```

```
  {
    Check temp;
    ++i;
    temp.i = i;

    return temp;
  }

  void Display()
  { cout << "i = " << i << endl; }
};

int main()
{
  Check obj, obj1;
  obj.Display();
  obj1.Display();

  obj1 = ++obj;

  obj.Display();
  obj1.Display();

  return 0;
}
```

Output

```
i = 0
i = 0
i = 1
I = 1
```

This program is similar to the one above.

The only difference is that, the return type of operator function is `Check` in this case which allows to use both codes `++obj; obj1 = ++obj;`. It is because, `temp` returned from operator function is stored in object `obj`.

Since, the return type of operator function is Check, you can also assign the

value of obj to another object.

Notice that, = (assignment operator) does not need to be overloaded because this operator is already overloaded in C++ library.

Example 3: Postfix Increment ++ Operator Overloading

Overloading of increment operator up to this point is only true if it is used in prefix form.

This is the modification of above program to make this work both for prefix form and postfix form.

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

class Check
{
 private:
   int i;
 public:
  Check(): i(0) { }
  Check operator ++ ()
  {
     Check temp;
     temp.i = ++i;
     return temp;
  }

  // Notice int inside barcket which indicates postfix increment.
  Check operator ++ (int)
  {
     Check temp;
     temp.i = i++;
     return temp;
  }

  void Display()
  { cout << "i = "<< i <<endl; }
```

```
};

int main()
{
   Check obj, obj1;
   obj.Display();
   obj1.Display();

   // Operator function is called, only then value of obj is assigned to obj1
   obj1 = ++obj;
   obj.Display();
   obj1.Display();

   // Assigns value of obj to obj1, only then operator function is called.
   obj1 = obj++;
   obj.Display();
   obj1.Display();

   return 0;
}
```

Output

```
i = 0
i = 0
i = 1
i = 1
i = 2
I = 1
```

When increment operator is overloaded in prefix form; Check operator ++ () is called but, when increment operator is overloaded in postfix form; Check operator ++ (int) is invoked.

Notice, the int inside bracket. This int gives information to the compiler that it is the postfix version of operator.

Don't confuse this int doesn't indicate integer.\

Example 4: Operator Overloading of Decrement -- Operator

Decrement operator can be overloaded in similar way as increment operator.

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

class Check
{
 private:
   int i;
 public:
   Check(): i(3) { }
   Check operator -- ()
   {
      Check temp;
      temp.i = --i;
      return temp;
   }

   // Notice int inside barcket which indicates postfix decrement.
   Check operator -- (int)
   {
      Check temp;
      temp.i = i--;
      return temp;
   }

   void Display()
   { cout << "i = "<< i <<endl; }
};

int main()
{
   Check obj, obj1;
   obj.Display();
   obj1.Display();

   // Operator function is called, only then value of obj is assigned to obj1
```

Output

```
i = 3
i = 3
i = 2
i = 2
i = 1
I = 2
```

Also, unary operators like: !, ~ etc can be overloaded in similar manner.

Summary

At the end of this unit, you learned that:

- A constructor is a special type of member function that initialises an object automatically when it is created.
- Constructor has the same name as that of the class and it does not have any return type.
- You can also initialise the data members inside the constructor's body as below.

```
temporary()
{
  x = 5;
  y = 5.5;
}
```

- Constructor can be overloaded in a similar way as function overloading.
- Overloaded constructors have the same name (name of the class) but different number of arguments.

Self Assessment

- What is a constructor?
- State how constructors worked in C++.
- State how to create a constructor
- Enumerate the types of constructors in C++.

Tutor Marked Assessment

- What is a constructor?
- Give an example of a constructor
- Write a program to describe a postfix operator

Further Reading

- https://www.studytonight.com › cpp › constructors-and-destructors-in-cpp
- https://www.geeksforgeeks.org › constructors-c
- https://www.w3schools.com › cpp › cpp_constructors
- https://www.programiz.com › cpp-programming › constructors
- https://www.studytonight.com › cpp › constructors-and-destructors-in-cpp

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.

Picture:
Data science programming

Photo: freepik.com

Module 7

Advance Object Oriented Technique

Units

Unit 1 - Inheritance
Unit 2 - Data Hiding
Unit 3 - Polymorphism

UNIT 1

Inheritance

Introduction

In this unit we shall be discussing inheritance which is one of the advanced features of OOP, why should we use inheritance, its implementation and some of the examples. Furthermore C++ overriding will be dealt with and C++ Multiple, multilevel and hierarchical inheritance with its syntax and example will be enumerated. Inheritance is one of the key features of Object-oriented programming in C++. It allows user to create a new class (derived class) from an existing class(base class).

The derived class inherits all the features from the base class and can have additional features of its own.

Learning Outcomes

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

1. Define is inheritance and different ways to implement it with examples?
2. State the reason for using inheritance
3. Enumerate how to access the overridden function in the base class from the derived class?
4. State the different models of inheritance in C++ programming
5. State Multiple, Multilevel and Hierarchical inheritance with examples.

Main Content

Why inheritance should be used?

7 min

Suppose, in your game, you want three characters - a maths teacher, a footballer and a businessman.

Since, all of the characters are persons, they can walk and talk. However, they also have some special skills. A maths teacher can teach maths, a footballer can play football and a businessman can run a business.

SAQ 2

You can individually create three classes who can walk, talk and perform their special skill as shown in the figure below.

In each of the classes, you would be copying the same code for walk and talk for each character.

If you want to add a new feature - eat, you need to implement the same code for each character. This can easily become error prone (when copying) and duplicate codes.

It'd be a lot easier if we had a Person class with basic features like talk, walk, eat, sleep, and add special skills to those features as per our characters. This is done using inheritance.

Using inheritance, now you don't implement the same code for walk and talk for each class. You just need to inherit them.

So, for Maths teacher (derived class), you inherit all features of a Person (base class) and add a new feature TeachMaths. Likewise, for a footballer, you inherit all the features of a Person and add a new feature PlayFootball and so on.

This makes your code cleaner, understandable and extendable.

It is important to remember: When working with inheritance, each derived class should satisfy the condition whether it "is a" base class or not. In the example above, Maths teacher is a Person, Footballer is a Person. You cannot have: Businessman is a Business.

Implementation of Inheritance in C++ Programming

```
class Person
{
```



```
  ……
};

class MathsTeacher : public Person
{
  ……
};

class Footballer : public Person
{
  ……
};
```

In the above example, class Person is a base class and classes MathsTeacher and Footballerare the derived from Person.

The derived class appears with the declaration of a class followed by a colon, the keyword public and the name of base class from which it is derived.

Since, MathsTeacher and Footballer are derived from Person, all data member and member function of Person can be accessible from them.

Example: Inheritance in C++ Programming

Create game characters using the concept of inheritance.

SAQ 1

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

class Person
{
   public:
     string profession;
     int age;

     Person(): profession("unemployed"), age(16) {}
     void display()
     {
        cout << "My profession is: " << profession << endl;
        cout << "My age is: " << age << endl;
        walk();
```

```
        talk();
    }
    void walk() { cout << "I can walk." << endl; }
    void talk() { cout << "I can talk." << endl; }
};

// MathsTeacher class is derived from base class Person.
class MathsTeacher : public Person
{
  public:
    void teachMaths() { cout << "I can teach Maths." << endl; }
};

// Footballer class is derived from base class Person.
class Footballer : public Person
{
  public:
    void playFootball() { cout << "I can play Football." << endl; }
};

int main()
{
    MathsTeacher teacher;
    teacher.profession = "Teacher";
    teacher.age = 23;
    teacher.display();
    teacher.teachMaths();

    Footballer footballer;
    footballer.profession = "Footballer";
    footballer.age = 19;
    footballer.display();
    footballer.playFootball();

    return 0;
}
```


Output

```
My profession is: Teacher
My age is: 23
I can walk.
I can talk.
I can teach Maths.
My profession is: Footballer
My age is: 19
I can walk.
I can talk.
I can play Football.
```

In this program, Person is a base class, while MathsTeacher and Footballer are derived from Person.

Person class has two data members - profession and age. It also has two member functions - walk() and talk().

Both MathsTeacher and Footballer can access all data members and member functions of Person.

However, MathsTeacher and Footballer have their own member functions as well: teachMaths() and playFootball() respectively. These functions are only accessed by their own class.

In the main() function, a new MathsTeacher object teacher is created.

Since, it has access to Person's data members, profession and age of teacher is set. This data is displayed using the display() function defined in the Person class. Also, the teachMaths() function is called, defined in the MathsTeacher class.

Likewise, a new Footballer object footballer is also created. It has access to Person's data members as well, which is displayed by invoking the display() function. The playFootball()function only accessible by the footballer is called then after.

Access specifiers in Inheritance

When creating a derived class from a base class, you can use different access specifiers to inherit the data members of the base class.

These can be public, protected or private.

In the above example, the base class Person has been inherited public-ly by MathsTeacherand Footballer.

Member Function Overriding in Inheritance

Suppose, base class and derived class have member functions with same name and arguments.

If you create an object of the derived class and try to access that member function, the member function in derived class is only invoked.

The member function of derived class overrides the member function of base class.

C++ Function Overriding

How to access the overridden function in the base class from the derived class?

Inheritance allows software developers to derive a new class from the existing class. The derived class inherits features of the base class (existing class).

Suppose, both base class and derived class have a member function with same name and arguments (number and type of arguments).

If you create an object of the derived class and call the member function which exists in both classes (base and derived), the member function of the derived class is invoked and the function of the base class is ignored.

This feature in C++ is known as function overriding.

How to access the overridden function in the base class from the derived class?

To access the overridden function of the base class from the derived class, scope resolution operator :: is used. For example,

If you want to access getData() function of the base class, you can use the following statement in the derived class.

```
Base::getData();
```

C++ Multiple, Multilevel and Hierarchical Inheritance

Inheritance is one of the core feature of an object-oriented programming language. It allows software developers to derive a new class from the existing class. The derived class inherits the features of the base class (existing class).

There are various models of inheritance in C++ programming.

C++ Multilevel Inheritance

In C++ programming, not only you can derive a class from the base class but you can also derive a class from the derived class. This form of inheritance is known as multilevel inheritance.

```
class A
{
……
};
class B: public A
{
……
};
class C: public B
{
……
};
```

Here, class B is derived from the base class A and the class C is derived from the derived class B.

Example 1: C++ Multilevel Inheritance

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

class A
{
  public:
    void display()
    {
      cout<<"Base class content.";
    }
};

class B : public A
{

};

class C : public B
{

};

int main()
{
  C obj;
  obj.display();
  return 0;
}
```

Output

```
Base class content.
```

In this program, class C is derived from class B (which is derived from base class A).

The obj object of class C is defined in the main() function.

When the display() function is called, display() in class A is executed. It's because there is no display() function in class C and class B.

The compiler first looks for the display() function in class C. Since the function doesn't exist there, it looks for the function in class B (as C is derived from B).

The function also doesn't exist in class B, so the compiler looks for it in class A (as B is derived from A).

If display() function exists in C, the compiler overrides display() of class A (because of member function overriding).

C++ Multiple Inheritance

In C++ programming, a class can be derived from more than one parents. For example: A class Bat is derived from base classes Mammal and WingedAnimal. It makes sense because bat is a mammal as well as a winged animal

Example 2: Multiple Inheritance in C++ Programming

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

class Mammal {
 public:
   Mammal()
   {
    cout << "Mammals can give direct birth." << endl;
   }
};

class WingedAnimal {
 public:
   WingedAnimal()
   {
    cout << "Winged animal can flap." << endl;
   }
};
```

```
class Bat: public Mammal, public WingedAnimal {

};

int main()
{
  Bat b1;
  return 0;
}
```

Output

```
Mammals can give direct birth.

Winged animal can flap.
```

Ambiguity in Multiple Inheritance

The most obvious problem with multiple inheritance occurs during function overriding.

Suppose, two base classes have a same function which is not overridden in derived class.

If you try to call the function using the object of the derived class, compiler shows error. It's because compiler doesn't know which function to call. For example,

```
class base1
{
 public:
   void someFunction( )
   { ..... ..... ..... }
};
class base2
{
  void someFunction( )
   { ..... ..... ..... }
};
```



```
class derived : public base1, public base2
{

};

int main()
{
  derived obj;

  obj.someFunction() // Error!
}
```

This problem can be solved using scope resolution function to specify which function to class either base1 or base2

```
int main()
{
  obj.base1::someFunction( );  // Function of base1 class is called
  obj.base2::someFunction();   // Function of base2 class is called.
}
```

C++ Hierarchical Inheritance

If more than one class is inherited from the base class, it's known as hierarchical inheritance. In hierarchical inheritance, all features that are common in child classes are included in the base class.

For example: Physics, Chemistry, Biology are derived from Science class.

Syntax of Hierarchical Inheritance

```
class base_class {
   ... .. ...
}

class first_derived_class: public base_class {
   ... .. ...
}

class second_derived_class: public base_class {
```

```
    ……
}

class third_derived_class: public base_class {
    ……
}
```

Summary

At the end of this unit you have learned that:

- Inheritance is one of the key features of Object-oriented programming in C++.
- Inheritance allows user to create a new class (derived class) from an existing class(base class).
- If you create an object of the derived class and try to access that member function, the member function in derived class is only invoked.
- The member function of derived class overrides the member function of base class.
- If you create an object of the derived class and call the member function which exists in both classes (base and derived), the member function of the derived class is invoked and the function of the base class is ignored.
- If you try to call the function using the object of the derived class, compiler shows error. It's because compiler doesn't know which function to call.

Self Assessment

- What is inheritance in C++
- State the reasons for using inheritance?
- Enumerate five type of inheritance in C++

Tutor Marked Assessment

- Write a program to access the overridden function in the base class from the derived class?
- State the different models of inheritance in C++ programming
- Multiple, Multilevel and Hierarchical inheritance with examples.

Further Reading

- https://beginnersbook.com › 2017/08 › cpp..
- https://www.includehelp.com › cpp-programs
- https://www.w3schools.in › cplusplus-tutorial
- www.trytoprogram.com › single-inheritance
- https://study.com › academy › lesson › inhe.

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.

UNIT 2

Data hiding

Introduction

In this unit we shall be discussing yet another important concepts of OOP which is called data hiding, i.e., a nonmember function cannot access an object's private or protected data. We will also enumerate C++ friend function and friend class, declaration of friend function with working examples.

But, sometimes this restriction may force programmer to write long and complex codes. So, there is mechanism built in C++ programming to access private or protected data from non-member functions. This is done using a friend function or/and a friend class.

Learning Outcomes

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

1. Define data hiding
2. Create friend function
3. Create friend classes in C++,
4. Declare friend function in C++ with example

Main Content

C++ friend Function and friend Classes | 5 min

friend Function in C++

Let us assume a function is defined as a friend function then, the private and protected data of a class can be accessed using the function.

SAQ 2

The complier knows a given function is a friend function by the use of the keyword friend.

For accessing the data, the declaration of a friend function should be made inside the body of the class (can be anywhere inside class either in private or public section) starting with keyword friend.

Declaration of friend function in C++

```
class class_name
{
   ……
   friend return_type function_name(argument/s);
   ……
}
```

Now, you can define the friend function as a normal function to access the data of the class. No friend keyword is used in the definition.

```
class className
{
   ……
   friend return_type functionName(argument/s);
   ……
}

return_type functionName(argument/s)
{
   ……
   // Private and protected data of className can be accessed from
   // this function because it is a friend function of className.
   ……
}
```


Example 1: Working of friend Function

```
/* C++ program to demonstrate the working of friend function.*/
#include <iostream>
using namespace std;

class Distance
{
   private:
      int meter;
   public:
      Distance(): meter(0) {}
      //friend function
      friend int addFive(Distance);
};

// friend function definition
int addFive(Distance d)
{
   //accessing private data from non-member function
   d.meter += 5;
   return d.meter;
}

int main()
{
   Distance D;
   cout<<"Distance: "<< addFive(D);
   return 0;
}
```

Output

```
Distance: 5
```

Here, friend function addFive() is declared inside Distance class. So, the private data metercan be accessed from this function.

Though this example gives you an idea about the concept of a friend function, it

doesn't show any meaningful use.

A more meaningful use would to when you need to operate on objects of two different classes. That's when the friend function can be very helpful.

You can definitely operate on two objects of different classes without using the friend function but the program will be long, complex and hard to understand.

Example 2: Addition of members of two different classes using friend Function

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

// forward declaration
class B;
class A {
  private:
    int numA;
  public:
    A(): numA(12) { }
    // friend function declaration
    friend int add(A, B);
};

class B {
  private:
    int numB;
  public:
    B(): numB(1) { }
    // friend function declaration
    friend int add(A , B);
};

// Function add() is the friend function of classes A and B
// that accesses the member variables numA and numB
int add(A objectA, B objectB)
{
  return (objectA.numA + objectB.numB);
}
```



```
int main()
{
  A objectA;
  B objectB;
  cout<<"Sum: "<< add(objectA, objectB);
  return 0;
}
```

Output

```
Sum: 13
```

In this program, classes A and B have declared add() as a friend function. Thus, this function can access private data of both class.

Here, add() function adds the private data numA and numB of two objects objectA and objectB, and returns it to the main function.

To make this program work properly, a forward declaration of a class class B should be made as shown in the above example.

This is because class B is referenced within the class A using code: friend int add(A , B);.

friend Class in C++ Programming

Similarly, like a friend function, a class can also be made a friend of another class using keyword friend. For example:

```
……
class B;
class A
{
  // class B is a friend class of class A
  friend class B;
  ……
}

class B
```

```
{
  ……
}
```

When a class is made a friend class, all the member functions of that class becomes friend functions.

In this program, all member functions of class B will be friend functions of class A. Thus, any member function of class B can access the private and protected data of class A. But, member functions of class A cannot access the data of class B.

Remember, friend relation in C++ is only granted, not taken.

friend Function in C++

If a function is defined as a friend function then, the private and protected data of a class can be accessed using the function.

The complier knows a given function is a friend function by the use of the keyword friend.

For accessing the data, the declaration of a friend function should be made inside the body of the class (can be anywhere inside class either in private or public section) starting with keyword friend.

Declaration of friend function in C++

```
class class_name
{
  ……
  friend return_type function_name(argument/s);
  ……
}
```

Now, you can define the friend function as a normal function to access the data of the class. No friend keyword is used in the definition.

```
class className
{
  ……
  friend return_type functionName(argument/s);
```



```
  ……
}

return_type functionName(argument/s)
{
  ……
  // Private and protected data of className can be accessed from
  // this function because it is a friend function of className.
  ……
}
```

Example 1: Working of friend Function

```
/* C++ program to demonstrate the working of friend function.*/
#include <iostream>
using namespace std;

class Distance
{
  private:
    int meter;55
  public:
    Distance(): meter(0) {}
    //friend function
    friend int addFive(Distance);
};

// friend function definition
int addFive(Distance d)
{
  //accessing private data from non-member function
  d.meter += 5;
  return d.meter;
}

int main()
{
  Distance D;
  cout<<"Distance: "<< addFive(D);
```

```
Distance: 5
```

Here, friend function addFive() is declared inside Distance class. So, the private data metercan be accessed from this function.

Though this example gives you an idea about the concept of a friend function, it doesn't show any meaningful use.

A more meaningful use would to when you need to operate on objects of two different classes. That's when the friend function can be very helpful.

You can definitely operate on two objects of different classes without using the friend function but the program will be long, complex and hard to understand.

Example 2: Addition of members of two different classes using friend Function

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

// forward declaration
class B;
class A {
   private:
     int numA;
   public:
     A(): numA(12) {}
     // friend function declaration
     friend int add(A, B);
};

class B {
   private:
      int numB;
   public:
      B(): numB(1) {}
      // friend function declaration
      friend int add(A , B);
};
```



```
// Function add() is the friend function of classes A and B
// that accesses the member variables numA and numB
int add(A objectA, B objectB)
{
  return (objectA.numA + objectB.numB);
}

int main()
{
   A objectA;
   B objectB;
   cout<<"Sum: "<< add(objectA, objectB);
   return 0;
}
```

Output

```
Sum: 13
```

In this program, classes A and B have declared add() as a friend function. Thus, this function can access private data of both class.

Here, add() function adds the private data numA and numB of two objects objectA and objectB, and returns it to the main function.

To make this program work properly, a forward declaration of a class class B should be made as shown in the above example.

This is because class B is referenced within the class A using code: friend int add(A , B);.

Summary

At the end of this unit you have learned that:

- If a function is defined as a friend function then, the private and protected data of a class can be accessed using the function.
- The complier knows a given function is a friend function by the use of the keyword friend.
- When a class is made a friend class, all the member functions of that class becomes friend functions.

Self Assessment

- Define data hiding
- State how create friend function
- Explain how create friend classes in C++,
- Enumerate how to declare friend function in C++ with example

Tutor Marked Assessment

- What is friend function in C++ with example ?
- State the characteristics of friend function ?
- What is information hiding in OOP?

Further Reading

- https://www.techopedia.com › definition › data-hiding
- https://www.cpp.thiyagaraaj.com › home › blog-1 › datahidingandencapsu...
- https://www.tutorialspoint.com › cplusplus › cpp_data_encapsulation
- https://www.ntu.edu.sg › home › ehchua › programming › cpp › cp3_OOP
- https://www.quora.com › What-is-data-hiding-in-a-program

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.

UNIT 3

Polymorphism

Introduction

In this unit we are going to discuss yet another important feature of OOP called polymorphism which means having many forms. The following will also be discussed pointer to base class, virtual members and abstract base classes. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Learning Outcomes

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

1. Define polymorphism
2. State the hierarchy of classes related to inheritance.
3. Write a program on pointer to base class
4. Write a program on virtual member

Main Content

Pointers to base class

7 min

One of the key features of derived classes is that a pointer to a derived class is type-compatible with a pointer to its base class. Polymorphism is the art of taking advantage of this simple but powerful and versatile feature, that brings Object Oriented Methodologies to its full potential.

SAQ 1

We are going to start by rewriting our program about the rectangle and the triangle of the previous section taking into consideration this pointer compatibility property:

```
// pointers to base class                    20
#include <iostream>                          10
using namespace std;

class CPolygon {
 protected:
  int width, height;
 public:
  void set_values (int a, int b)
    { width=a; height=b; }
 };

class CRectangle: public CPolygon {
 public:
  int area ()
    { return (width * height); }
 };

class CTriangle: public CPolygon {
 public:
  int area ()
    { return (width * height / 2); }
 };

int main () {
```



```
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << rect.area() << endl;
  cout << trgl.area() << endl;
  return 0;
}
```

In function main, we create two pointers that point to objects of class CPolygon (ppoly1 and ppoly2). Then we assign references to rect and trgl to these pointers, and because both are objects of classes derived from CPolygon, both are valid assignment operations.

The only limitation in using *ppoly1 and *ppoly2 instead of rect and trgl is that both *ppoly1 and *ppoly2 are of type CPolygon* and therefore we can only use these pointers to refer to the members that CRectangle and CTriangle inherit from CPolygon. For that reason when we call the area() members at the end of the program we have had to use directly the objects rect and trgl instead of the pointers *ppoly1 and *ppoly2.

In order to use area() with the pointers to class CPolygon, this member should also have been declared in the class CPolygon, and not only in its derived classes, but the problem is that CRectangle and CTriangle implement different

Virtual Members

A member of a class that can be redefined in its derived classes is known as a virtual member. In order to declare a member of a class as virtual, we must precede its declaration with the keyword virtual:

SAQ 5

```
// virtual members                           20
#include <iostream>                          10
using namespace std;                          0

class CPolygon {
 protected:
```

```
  int width, height;
 public:
  void set_values (int a, int b)
    { width=a; height=b; }
  virtual int area ()
    { return (0); }
};

class CRectangle: public CPolygon {
 public:
  int area ()
    { return (width * height); }
};

class CTriangle: public CPolygon {
 public:
  int area ()
    { return (width * height / 2); }
};

int main () {
 CRectangle rect;
 CTriangle trgl;
 CPolygon poly;
 CPolygon * ppoly1 = &rect;
 CPolygon * ppoly2 = &trgl;
 CPolygon * ppoly3 = &poly;
 ppoly1->set_values (4,5);
 ppoly2->set_values (4,5);
 ppoly3->set_values (4,5);
 cout << ppoly1->area() << endl;
 cout << ppoly2->area() << endl;
 cout << ppoly3->area() << endl;
 return 0;
}
```

Now the three classes (CPolygon, CRectangle and CTriangle) have all the same members: width, height, set_values() and area().

The member function area() has been declared as virtual in the base class because it is later redefined in each derived class. You can verify if you want

that if you remove this virtual keyword from the declaration of area() within CPolygon, and then you run the program the result will be 0 for the three polygons instead of 20, 10 and 0. That is because instead of calling the corresponding area() function for each object (CRectangle::area(), CTriangle::area() and CPolygon::area(), respectively), CPolygon::area() will be called in all cases since the calls are via a pointer whose type is CPolygon*.

Therefore, what the virtual keyword does is to allow a member of a derived class with the same name as one in the base class to be appropriately called from a pointer, and more precisely when the type of the pointer is a pointer to the base class but is pointing to an object of the derived class, as in the above example.

A class that declares or inherits a virtual function is called a polymorphic class.

Note that despite of its virtuality, we have also been able to declare an object of type CPolygon and to call its own area() function, which always returns 0.

Abstract base classes

SAQ 4

Abstract base classes are something very similar to our CPolygon class of our previous example. The only difference is that in our previous example we have defined a valid area() function with a minimal functionality for objects that were of class CPolygon (like the object poly), whereas in an abstract base classes we could leave that area() member function without implementation at all. This is done by appending =0 (equal to zero) to the function declaration.

An abstract base CPolygon class could look like this:// abstract class CPolygon

```
class CPolygon {
 protected:
  int width, height;
 public:
  void set_values (int a, int b)
    { width=a; height=b; }
  virtual int area () =0;
}                                                    ;
```

I want you to take note of how we appended =0 to virtual int area () instead of specifying an implementation for the function. This type of function is called a pure virtual function, and all classes that contain at least one pure virtual function are abstract base classes.

I want you to observe the main difference between an abstract base class and a regular polymorphic class is that because in abstract base classes at least one of its members lacks implementation we cannot create instances (objects) of it.

But a class that cannot instantiate objects is not totally useless. We can create pointers to it and take advantage of all its polymorphic abilities. Therefore a declaration like:

```
Cpolygon poly;
```

would not be valid for the abstract base class we have just declared, because tries to instantiate an object. Nevertheless, the following pointers:CPolygon * ppoly1;

CPolygon * ppoly2;
would be perfectly valid.

This is so for as long as CPolygon includes a pure virtual function and therefore it's an abstract base class. However, pointers to this abstract base class can be used to point to objects of derived classes.

```
// abstract base class                    20
#include <iostream>                       10
using namespace std;

class CPolygon {
 protected:
   int width, height;
 public:
   void set_values (int a, int b)
     { width=a; height=b; }
   virtual int area (void) =0;
 };
```



```
class CRectangle: public CPolygon {
 public:
   int area (void)
     { return (width * height); }
 };

class CTriangle: public CPolygon {
 public:
   int area (void)
     { return (width * height / 2); }
 };

int main () {
 CRectangle rect;
 CTriangle trgl;
 CPolygon * ppoly1 = &rect;
 CPolygon * ppoly2 = &trgl;
 ppoly1->set_values (4,5);
 ppoly2->set_values (4,5);
 cout << ppoly1->area() << endl;
 cout << ppoly2->area() << endl;
 return 0;
}
```

If you review the program you will notice that we refer to objects of different but related classes using a unique type of pointer (CPolygon*). This can be tremendously useful. For example, now we can create a function member of the abstract base class CPolygon that is able to print on screen the result of the area() function even though CPolygon itself has no implementation for this function:

```
// pure virtual members can be called     20
// from the abstract base class           10
#include <iostream>
using namespace std;

class CPolygon {
 protected:
```

```
  int width, height;
 public:
  void set_values (int a, int b)
    { width=a; height=b; }
  virtual int area (void) =0;
  void printarea (void)
    { cout << this->area() << endl; }
 };

class CRectangle: public CPolygon {
 public:
  int area (void)
    { return (width * height); }
 };

class CTriangle: public CPolygon {
 public:
  int area (void)
    { return (width * height / 2); }
 };

int main () {
 CRectangle rect;
 CTriangle trgl;
 CPolygon * ppoly1 = &rect;
 CPolygon * ppoly2 = &trgl;
 ppoly1->set_values (4,5);
 ppoly2->set_values (4,5);
 ppoly1->printarea();
 ppoly2->printarea();
 return 0;
}
```

Virtual members and abstract classes grant C++ the polymorphic characteristics that make object-oriented programming such a useful instrument in big projects. Of course, we have seen very simple uses of these features, but these features can be applied to arrays of objects or dynamically allocated objects.

Let's end with the same example again, but this time with objects that are dynamically allocated:

```
// dynamic allocation and polymorphism                20
#include <iostream>                                   10
using namespace std;

class CPolygon {
 protected:
  int width, height;
 public:
  void set_values (int a, int b)
    { width=a; height=b; }
  virtual int area (void) =0;
  void printarea (void)
    { cout << this->area() << endl; }
 };

class CRectangle: public CPolygon {
 public:
  int area (void)
    { return (width * height); }
 };

class CTriangle: public CPolygon {
 public:
  int area (void)
    { return (width * height / 2); }
 };

int main () {
 CPolygon * ppoly1 = new CRectangle;
 CPolygon * ppoly2 = new CTriangle;
 ppoly1->set_values (4,5);
 ppoly2->set_values (4,5);
 ppoly1->printarea();
 ppoly2->printarea();
 delete ppoly1;
 delete ppoly2;
 return 0;
}
```

Notice that the ppoly pointers:CPolygon * ppoly1 = new CRectangle;

Cpolygon * ppoly2 = new CTriangle;

are declared being of type pointer to CPolygon but the objects dynamically allocated have been declared having the derived class type directly.

Summary

At the end of this unit you learned that:

- One of the key features of derived classes is that a pointer to a derived class is type-compatible with a pointer to its base class.
- Polymorphism is the art of taking advantage of this simple but powerful and versatile feature that brings Object Oriented Methodologies to its full potential.
- Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.
- C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Self Assessment

- What is polymorphism?
- Why do we need polymorphism in C++ ?
- State the hierarchy of classes related to inheritance.
- Write a program on pointer to base class
- Write a program on virtual member

Tutor Marked Assessment

- State the advantage and disadvantages of Polymorphism
- How does polymorphism promote extensibility?

Further Reading

- https://www.geeksforgeeks.org › polymorp...
- https://www.tutorialspoint.com › cplusplus
- https://beginnersbook.com › 2017/08 › cpp...
- https://study.com › academy › lesson › poly...
- https://www.cs.bu.edu › cpp › intro
- https://www.w3schools.in › cplusplus-tutorial

References

- P. B. Shola (2002): Learn C++ Programming Language: Reflect Publishers, Ibadan, Oyo State, Nigeria
- John R Hubbard (2006): Programming with C++, Schaum's outlines second edition: Tata McGraw-Hill Publishing Company Limited Newde
- Paul Deitel & Harvey Deitel (2012): C++ How to program, Eight Edition: Pearson Education, Inc., Publishing as Prentice Hall.