TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

Central Department of Computer Science and Information Technology

Kirtipur, Kathmandu

A Case Study Report

on

C++ Programming Language

Submitted by:

Name: Brihat Ratna Bajracharya

Roll No.: 19/075

Submitted to:

Mrs. Lalita Sthapit

Central Department of Computer Science and Information Technology

Date of submission: *2077 Falgun 28*

TABLE OF CONTENTS

Introduction

C++ is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes". This name is credited to Rick Mascitti (mid-1983) and was first used in December 1983. During C++'s development period, the language was referred to as "new C" and "C with Classes" before acquiring its final name.

The first edition of The C++ Programming Language was released in 1985, which became the definitive reference for the language, as there was not yet an official standard. The first commercial implementation of C++ was released in October of the same year.

In 1998, the ISO working group standardized C++ for the first time as ISO/IEC 14882:1998, which is informally known as C++98. In 2003, it published a new version of the C++ standard called ISO/IEC 14882:2003, which fixed problems identified in C++98.

The next major revision of the standard was informally referred to as "C++0x", but it was not released until 2011. C++11 (14882:2011) included many additions to both the core language and the standard library. In 2014, C++14 (also known as C++1y) was released as a small extension to C++11, featuring mainly bug fixes and small improvements. After C++14, a major revision C++17, informally known as C++1z, was completed by the ISO C++ Committee in mid July 2017 and was approved and published in December 2017.

The latest stable release version is C++20 (ISO/IEC 14882:2020) released on 15 December 2020. The current C++20 standard supersedes these with new features and an enlarged standard library. Present version of the language has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation. It is almost always implemented as a compiled language, and many vendors provide C++ compilers, including the Free Software Foundation, LLVM, Microsoft, Intel, Oracle, and IBM, so it is available on many platforms.

C++ was designed with an orientation toward system programming and embedded, resource-constrained software and large systems, with performance, efficiency, and flexibility of use as its design highlights. C++ has also been found useful in many other applications including desktop applications, video games, servers (e.g. e-commerce, web search, or SQL servers), and performance-critical applications (e.g. telephone switches or space probes).

Since 2012, C++ is on a three-year release schedule, with C++23 being the next planned standard.

C++ Program Structure

A C++ program, is a collection of objects that communicate via invoking each other's methods. The major terms/concepts in C++ are:

Class

A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.

Object

An object is an instance of a class. Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging, barking, eating.

Methods

A method (also called functions) is basically a behavior. A class can contain many methods. It is in methods where the logic are written, data is manipulated and all the actions are executed.

Instance Variables

Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

Sample C++ Program

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

// main() is where program execution begins.
int main() {
   cout << "Hello World"; // prints Hello World
   return 0;
}
```

The C++ language defines several headers, which contain information that is either necessary or useful to the program. For this program, the header <iostream> is needed. The line **using namespace std;** tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++. The next line **'// main() is where program execution begins.'** is a single-line comment available in C++. Single-line comments begin with // and stop at the end of the line. The line **int main()** is the main function where program execution begins. The next line **cout << "Hello**

World"; causes the message "Hello World" to be displayed on the screen. The next line **return 0;** terminates main( )function and causes it to return the value 0 to the calling process. The semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

C++ Identifiers

A C++ identifier is a name used to identify a variable, function, class, module, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9). C++ does not allow punctuation characters such as @, $, and % within identifiers. C++ is a case-sensitive programming language. Thus, Manpower and manpower are two different identifiers in C++.

The following list shows the reserved words in C++. These reserved words may not be used as constant or variable or any other identifier names.

asm	**else**	**new**	**this**	**auto**	**enum**	**break**
operator	**throw**	**bool**	**explicit**	**private**	**true**	**export**
protected	**try**	**case**	**extern**	**public**	**typedef**	**catch**
false	**register**	**typeid**	**char**	**float**	**typename**	**class**
for	**return**	**union**	**const**	**friend**	**short**	**mutable**
unsigned	**const_cast**	**goto**	**signed**	**using**	**if**	**sizeof**
continue	**virtual**	**default**	**inline**	**static**	**void**	**delete**
int	**volatile**	**do**	**long**	**struct**	**wchar_t**	**double**
switch	**while**	**template**	**namespace**	**dynamic_cast**		
static_cast		**reinterpret_cast**				

Data Types

Data types in C++ is mainly divided into three types. Primitive Data Types are built-in or predefined data types and can be used directly by the user to declare variables. Primitive data types available in C++ are: *Integer, Character, Boolean, Floating Point, Double Floating Point, Valueless or Void and Wide Character*. Several of the basic types can be modified using one or more of the type modifiers − signed, unsigned, short and long

Derived Data Types are derived from the primitive or built-in datatypes. These can be of four types namely: *Function, Array, Pointer, Reference*

Abstract or User-Defined Data Types are defined by user itself. Like, defining a class in C++ or a structure. C++ provides the following user-defined datatypes: Class, Structure, Union, Enumeration and Typedef defined datatype.

Primitive data types

Integer

Keyword used for integer data types is int. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.

Character

Character data type is used for storing characters. Keyword used for character data type is char. Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.

Boolean

Boolean data type is used for storing boolean or logical values. A boolean variable can store either true or false. Keyword used for boolean data type is bool. It is stored as 8-bit ASCII code

Floating Point

Floating Point data type is used for storing single precision floating point values or decimal values. Keyword used for floating point data type is float. Float variables typically requires 4 byte of memory space.

Double Floating Point

Double Floating Point data type is used for storing double precision floating point values or decimal values. Keyword used for double floating point data type is double. Double variables typically requires 8 byte of memory space.

Void

Void means without any value. void datatype represents a valueless entity. Void data type is used for those function which does not returns a value.

Wide Character

Wide character data type is also a character data type but this data type has size greater than the normal 8-bit datatype. Represented by wchar_t. It is generally 2 or 4 bytes long.

Derived Data Types

Function

A function is a block of code or program-segment that is defined to perform a specific well-defined task. A function is generally defined to save the user from writing the same lines of code again and again for the same input. All the lines of code are put together inside a single function and this can be called anywhere required. main() is a default function that is defined in every program of C++.

Syntax

```
FunctionType FunctionName(parameters) {}
```

Array

An array is a collection of items stored at continuous memory locations. The idea of array is to represent many instances in one variable.

Syntax

```
DataType ArrayName[size_of_array];
```

Pointers

Pointers are symbolic representation of addresses. They enable programs to simulate call-by-reference as well as to create and manipulate dynamic data structures. It's general declaration in C/ C++ has following format:

Syntax

```
datatype *var_name;
```

Reference

When a variable is declared as reference, it becomes an alternative name for an existing variable. A variable can be declared as reference by putting '**&**' in the declaration.

User-Defined Data Types

Class

The building block of C++ that leads to Object-Oriented programming is a class. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

Syntax

```
class ClassName {
    Access specifier:              // can be private, public or protected
    Data Members;                  // variables to be used
    Member Functions( ) {}         // methods to access data members
};
```

Structure

A structure is a user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type.

Example

```
struct address {
    char name[50];
    char street[100];
    char city[50];
    char state[20];
    int pin;
};
```

Union

Like Structures, union is a user defined data type. In union, all members share the same memory location. For example in the following C program, both x and y share the same location. If we change x, we can see the changes being reflected in y.

Enumeration

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

Example

```
enum State {Working = 1, Failed = 0};
```

Typedef

C++ allows you to define explicitly new data type names by using the keyword typedef. Using typedef does not actually create a new data class, rather it defines a name for an existing type. This can increase the portability(the ability of a program to be used across different types of machines; i.e., mini, mainframe, micro, etc; without much changes into the code) of a program as only the typedef statements would have to be changed. Using typedef one can also aid in self-documenting code by allowing descriptive names for the standard data types.

Syntax

```
typedef type name;
```

where type is any C++ data type and name is the new name for this data type.
This defines another name for the standard type of C++.

Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ is rich in built-in operators and provide the following types of operators − Arithmetic Operators, Relational Operators, Logical Operators, Bitwise Operators, Assignment Operators and Misc Operators

Arithmetic Operators

There are following arithmetic operators supported by C++ language

Suppose A = 10, B = 20

Operator	Description	Example
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator (remainder after int division)	B % A = 0
++	Increment operator, increases value by one	A++ will give 11
--	Decrement operator, decreases value by one	A-- will give 9

Relational Operators

There are following relational operators supported by C++ language. For A = 10 and B = 20,

== Checks if the values of two operands are equal or not, if yes then condition becomes true. *(A == B) is not true.*

!= Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. *(A != B) is true.*

> Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. *(A > B) is not true.*

< Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. *(A < B) is true.*

>= Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. *(A >= B) is not true.*

<= Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. *(A <= B) is true.*

Logical Operators

There are following logical operators supported by C++ language.

Suppose variable A holds 1 and variable B holds 0, then

&& Logical AND operator. If both the operands are non-zero, then condition becomes true. *(A && B) is false.*

|| Logical OR Operator. If any of the two operands is non-zero, then condition becomes true. *(A || B) is true.*

! Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false. *!(A && B) is true.*

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation.

Assume if A = 60; and B = 13; now in binary format they will be as follows −

```
A  =  0011 1100
B  =  0000 1101
--------------
A&B = 0000 1100
A|B = 0011 1101
```

```
A^B = 0011 0001
~A  = 1100 0011
```

Assignment Operators

There are following assignment operators supported by C++ language −

= Simple assignment operator, Assigns values from right side operands to left side operand. *C = A + B will assign value of A + B into C*

+= Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand. *C += A is equivalent to C = C + A*

-= Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand. *C -= A is equivalent to C = C - A*

*= Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand. *C *= A is equivalent to C = C * A*

/= Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand. *C /= A is equivalent to C = C / A*

%= Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand. *C %= A is equivalent to C = C % A*

<<= Left shift AND assignment operator. *C <<= 2 is same as C = C << 2*

>>= Right shift AND assignment operator. *C >>= 2 is same as C = C >> 2*

&= Bitwise AND assignment operator. *C &= 2 is same as C = C & 2*

^= Bitwise exclusive OR and assignment operator. *C ^= 2 is same as C = C ^ 2*

|= Bitwise inclusive OR and assignment operator. *C |= 2 is same as C = C | 2*

Misc Operators

The following table lists some other operators that C++ supports.

sizeof

sizeof operator returns the size of a variable. For example, sizeof(a), where ‘a’ is integer, and will return 4.

Condition ? X : Y

Conditional operator (?). If Condition is true then it returns value of X otherwise returns value of Y.

,

Comma operator causes a sequence of operations to be performed. The value of the entire comma expression is the value of the last expression of the comma-separated list.

. (dot) and -> (arrow)

Member operators are used to reference individual members of classes, structures, and unions.

Cast

Casting operators convert one data type to another. For example, int(2.2000) would return 2.

&

Pointer operator & returns the address of a variable. For example &a; will give actual address of the variable.

*****

Pointer operator * is pointer to a variable. For example *var; will pointer to a variable var.

C++ Loop Types

C++ programming language provides the following type of loops to handle looping requirements.

while loop

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

for loop

Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

do...while loop

Like a 'while' statement, except that it tests the condition at the end of the loop body.

nested loops

You can use one or more loop inside any another ‘while’, ‘for’ or ‘do..while’ loop.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. C++ supports the following control statements.

break statement

Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.

continue statement

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

goto statement

Transfers control to the labeled statement. Though it is not advised to use goto statement in your program.

Statement Level Control Structures in C++

Selection Statements

Two-Way Selection Statements (If-clause)

Syntax

```
if (control_expression) {
    clause
}
else {
    clause
}
```

Also support nested if clause

Multiple Selection Statements

switch.. case statement in C++ is the example of multiple selection statements whose syntax is given as below:

Syntax

```
switch (expression) {
    case constant_expr_1:
        statement_1;
        break;
    . . .
    case constant_expr_n:
        statement_n;
        break;
    [default:
        statementn + 1]
}
```

Multiple Selection can also be implemented using if-clause as follows:

Syntax:

```
if (control_expression_1) {
    clause_1;
}
else if (control_expression_2) {
    clause_2;
}
…
else if (control_expression_n) {
    clause_n;
}
else {
    clause;
}
```

Iterative Statements

For-loop, while-loop and do-while loop are the iterative statements in C++

for-loop

Syntax:

```
for (expression_1; expression_2; expression_3) {
    loop body statements;
}
```

while loop

Syntax:

```
while (control_expression) {
    loop body statements;
}
```

do-while loop

Syntax:

```
do {
    loop body statements;
} while (control_expression);
```

Sequence Subprogram control (Functions in C++)

A function is a group of statements that together perform a task. Every C++ program has at least one function, which is main(), and all the most trivial programs can define additional functions. The code can be divided into separate functions. The division is usually done logically such that each function performs a specific task. A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function. The C++ standard library also provides numerous built-in functions. For example, function strcat() to concatenate two strings, function memcpy() to copy one memory location to another location and many more functions. The general form of a C++ function definition is as follows:

```
return_type function_name( parameter list ) {
   body of the function
}
```

Return Type − A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In those case, the return_type is the keyword void.

Function Name − This is the actual name of the function. The function name and the parameter list together constitute the function signature.

Parameters − A parameter is like a placeholder. When a function is invoked, values are passed as parameters. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

Function Body − The function body contains a collection of statements that define what the function does.

Parameter Passing Methods in C++

Pass by Value

This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument. By default, C++ uses pass by value to pass arguments.

Pass by Pointer

This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

Pass by Reference

This method copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

Type Checking Parameters in C++

In C++, all functions must have their formal parameters in prototype form. However, type checking can be avoided for some of the parameters by replacing the last part of the parameter list with an ellipsis, as in

```
int printf(const char* format_string, . . .);
```

A call to printf must include at least one parameter, a pointer to a literal character string.

Generic Functions in C++

Generic functions in C++ have the descriptive name of template functions. The definition of a template function has the general form

```
template <template parameters>
—a function definition that may include the template parameters
```

A template parameter (there must be at least one) has one of the forms

```
class identifier
typename identifier
```

The class form is used for type names. The typename form is used for passing a value to the template function.

Example:

```
template <class Type>
Type max(Type first, Type second) {
    return first > second ? first : second;
}
```

Instantiation:

```
int max(int first, int second) {
    return first > second ? first : second;
}
```

Object Oriented Features of C++

Objects

Objects are the basic unit of OOP (Object Oriented Programming). They are instances of class, which have data members and uses various member functions to perform tasks.

Class

It is similar to structures in C language. Class can also be defined as user defined data type but it also contains functions in it. So, class is basically a blueprint for object. It declare & defines what data variables the object will have and what operations can be performed on the class's object.

Abstraction

Abstraction refers to showing only the essential features of the application and hiding the details. In C++, classes can provide methods to the outside world to access & use the data variables, keeping the variables hidden from direct access, or classes can even declare everything accessible to everyone, or maybe just to the classes inheriting it. This can be done using access specifiers.

Abstract Data Type in C++

The first important additions in C++ were those to support object-oriented programming. Because one of the primary components of object-oriented programming is abstract data types, C++ obviously is required to support them. C++ provides two constructs that are very similar to each other, the class and the struct, which more directly support abstract data types. C++ classes are types. A C++ program unit that declares an instance of a class can also access any of the public entities in that class, but only through an instance of the class.

Encapsulation

It can also be said data binding. Encapsulation is all about binding the data variables and functions together in class.

Inheritance

Inheritance is a way to reuse once written code again and again. The class which is inherited is called the Base class & the class which inherits is called the Derived class. They are also called parent and child class.

Abstraction means making visible what you want the external world to see and keeping the other details behind the wraps. It hides the complexity in which we decide what information to hide and what to expose. **Information hiding** is making inaccessible certain details which would not affect other parts of the system. **Encapsulation** is like enclosing in a capsule i.e. enclosing the related operations and data related to an object into that object. Encapsulation is not “the same thing as

information hiding". For example, even though information may be encapsulated within record structures and arrays, this information is usually not hidden and is available for use. Encapsulation means just getting a set of operations and data together which belong together and putting them in a capsule.

Abstraction, information hiding, and encapsulation are different but related. Abstraction is a technique that helps us identify which specific information should be visible, and which information should be hidden. Encapsulation can be thought of as the implementation of the abstract class.

Polymorphism

It is a feature, which lets us create functions with same name but different arguments, which will perform different actions. That means, functions with same name, but functioning in different ways. Or, it also allows us to redefine a function to provide it with a completely new definition. You will learn how to do this in details soon in coming lessons.

Exception Handling in C++

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero. Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: **try, catch**, and **throw**.

throw − A program throws an exception when a problem shows up. This is done using a throw keyword.

catch − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

try − A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Assuming a block will raise an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch as follows −

```
try {
   // protected code
} catch( ExceptionName e1 ) {
   // catch block
} catch( ExceptionName e2 ) {
   // catch block
} catch( ExceptionName eN ) {
   // catch block
}
```

C++ Standard Exceptions

C++ provides a list of standard exceptions defined in <exception> which we can use in our programs. Following are such exceptions

std::exception

An exception and parent class of all the standard C++ exceptions.

std::bad_alloc

This can be thrown by **new**.

std::bad_cast

This can be thrown by **dynamic_cast**.

std::bad_exception

This is useful device to handle unexpected exceptions in a C++ program.

std::bad_typeid

This can be thrown by **typeid**.

std::logic_error

An exception that theoretically can be detected by reading the code.

std::domain_error

This is an exception thrown when a mathematically invalid domain is used.

std::invalid_argument

This is thrown due to invalid arguments.

std::length_error

This is thrown when a too big **std::string** is created.

std::out_of_range

This can be thrown by the 'at' method, for example a **std::vector** and **std::bitset<>::operator[]().**

std::runtime_error

An exception that theoretically cannot be detected by reading the code.

std::overflow_error

This is thrown if a mathematical overflow occurs.

std::range_error

This is occurred when you try to store a value which is out of range.

std::underflow_error

This is thrown if a mathematical underflow occurs.

Bibliography

[1] R. W. Sebesta, “Generic Functions in C++” in *Concepts of Programming Languages*, 11th ed., Essex: Pearson Education Limited, 2016, pp. 424-425.

[2] R. W. Sebesta, “Iterative Statements” in *Concepts of Programming Languages*, 11th ed., Essex: Pearson Education Limited, 2016, pp. 367-377.

[3] R. W. Sebesta, “Selection Statements” in *Concepts of Programming Languages*, 11th ed., Essex: Pearson Education Limited, 2016, pp. 356-367.

[4] R. W. Sebesta, “Type Checking Parameters” in *Concepts of Programming Languages*, 11th ed., Essex: Pearson Education Limited, 2016, pp. 410-411.

[5] “C++ Basic Syntax,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/cplusplus/cpp_basic_syntax.htm. [Accessed: 7-Mar-2021].

[6] “C++ Data Types,” *GeeksforGeeks*, 14-Oct-2020. [Online]. Available: https://www.geeksforgeeks.org/c-data-types/. [Accessed: 7-Mar-2021].

[7] “C++ Exception Handling,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm. [Accessed: 11-Mar-2021].

[8] “C++ Functions,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/cplusplus/cpp_functions.htm. [Accessed: 9-Mar-2021].

[9] “C++ Loop Types,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/cplusplus/cpp_loop_types.htm. [Accessed: 8-Mar-2021].

[10] “C++,” *Wikipedia*, 28-Feb-2021. [Online]. Available: https://en.wikipedia.org/wiki/C%2B%2B. [Accessed: 7-Mar-2021].

[11] “Derived Data Types in C++,” *GeeksforGeeks*, 26-Feb-2020. [Online]. Available: https://www.geeksforgeeks.org/derived-data-types-in-c. [Accessed: 7-Mar-2021].

[12] “Object Oriented Programming in C++,” *studytonight.com*. [Online]. Available: https://www.studytonight.com/cpp/cpp-and-oops-concepts.php. [Accessed: 10-Mar-2021].

[13] “Operators in C++,” *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/cplusplus/cpp_operators.htm. [Accessed: 8-Mar-2021].

[14] “User defined Data Types in C++,” *GeeksforGeeks*, 08-Mar-2021. [Online]. Available: https://www.geeksforgeeks.org/user-defined-derived-data-types-in-c/. [Accessed: 7-Mar-2021].