SCHOOL OF COMPUTER SCIENCES

SESSION 2019/2020 SEMESTER II

CPT 211 : PROGRAMMING LANGUAGE CONCEPTS AND PARADIGMS

ASSIGNMENT 2: MULTI PARADIGM LANGUAGE

SUBMISSION DATE: 6TH APRIL 2020

GROUP 13

Name	Matric number
Ainil Hawa binti Abdul Rozak	141928
Nur Aisyah binti Ahmad Zaki	141870
Farah Mursyidah binti Fuahaidi	144395
Nurul Adilah binti Mohd Asri	142083
Nurul Murshida binti Omar Bakri	144184

1.0 Multi paradigm language and Introduction to C++

Multiparadigm language is described as a programming language that supports more than one programming paradigm. A programming paradigm is known as a way to classify programming languages based on their features as it characterizes the styles, concepts and methods of the language for describing, processing and solving problems. Paradigms such as object-oriented, imperative, functional and logical are the four most common paradigms that help in maximizing freestyling work tasks, freely intermixing constructs from different paradigms. However, a combination of multiple paradigms in a language is often recognized as the best implementation as a particular paradigm might only serve best in one particular area. C++ is one example of multi-paradigm languages that offer a lot of advantages and is becoming one of the most popular programming languages in the world for its capability to solve problems and provide solutions with maximum functionalities.

C++ is a programming language developed by Bjarne Stroustrup as an extension to the C language. C++ language is actually a family and successor of C. C was created in 1979 until Bjarne Stroustrup decided to elevate and enhance it with new features. Bjarne chose to implement simula-like features in C as it was very helpful especially with large software development. As C language is general-purpose, fast, portable and widely-used, it assimilates really well with Simula and other programming languages including ALGOL68, CLU and ML and Ada which are as well providing huge contributions to the creation of C++.

C++ as it is called means an increment to the C language. At first, C++ was only added on features like classes, derived classes, strong typing, inlining and default arguments. However, the language has expanded significantly over time where now it encompasses new features including virtual functions, function name, operator overloading, references, constants, type-safe free memory allocation and many more. In 1982, the first edition of The C++ Programming Language was released and it became the definitive reference for the language. It was well perceived by all, hence triggered the release of an updated version (C++ 2.0) in 1989 where it includes inheritance, abstract classes, static member functions and many more. The latest version of C++ is C++17 and for the record, it holds the fourth most popular programming language in 2019. Further changes and improvements are already planned and soon will be implemented this year (2020).

As aforementioned, C++ is one of the most incredible multi-paradigm languages that provide programmers with maximum functionalities. C++ holds many programming paradigms under its wing, but four of the paradigms that will be highlighted and further elaborated in this paper are object-oriented, functional, imperative and logical style.

2.0 Advantages and disadvantages of C++

Established in 1982, C++ language has gone through a long journey and without a doubt is one of the most powerful languages that still dominate the world of programming. However, despite the advantages and benefits it offers, there are also some trade-offs that we have to take well note of. Below are the elaborations of benefits and flaws of C++ language in terms of implementation and functionalities.

2.1 Advantages of C++:

- **Multiparadigm language**
  As discussed before, C++ is one of the programming languages that implement multi-paradigm. It encompasses logic, procedures and structures of the program. The power of C++ is that it can implement any paradigm in its language. One famous paradigm under C++ is object-oriented style as well as there is functional programming support in the modern standard library version. Having multiple paradigms in a language like C++ allow users to produce cleaner and maintainable code compared to single paradigm implementation.

- **Portability**
  Language's portability depends on the object code created by compilers. C++ code is known as machine-dependent and tied to a particular operating system. Even so, C++ offers features of platform independence that enables users to run the same code at different operating systems easily. For instance, the code is written in Linux and you somehow switch your operating system to Windows. The code will still run without any errors.

- **Large community support**
  C++ is ranked as the top 5 most powerful languages in the programming world. Because of the usability and maximum functionalities, many people try to harvest their skills to program in this language. This contributes to a lot of community support to enhance and improve the knowledge of C++. There are a lot of courses online, providing support in C++ knowledge. StackOverflow, Github and GeeksforGeeks are among online teaching resources for C++.

- **Compatibility with C language**
  C++ is pretty much compatible with C. Virtually, every error-free C program is a valid C++ program. Depending on the compiler used, every program of C++ can run on a file with .cpp extension (Soffar, 2017).

- **Scalability**
  In terms of computational power, C++ has higher scalability compared to Java. It enables users to directly communicate with GPU via CUDA/OpenCL APIs in some parallel tasks. Apart from that, it also offers huge scalability based on its flexible memory management. It enables users to write their own model of memory management which gives the users a huge advantage especially in developing applications with scalability purposes.

2.2 Disadvantages of C++

- **Security issues**
  There are some complications with C++ due to its memory unsafety. There are various types of memory unsafety vulnerabilities with C/C++ such as:
    i) Type confusion: it mixes up the type of value that exists at a place in memory
    ii) Use after free: it uses a piece of memory even after you are done with it
    iii) Use of uninitialized memory: it uses a piece of memory even before you've stored anything on it (Priyadarshini, 2018).

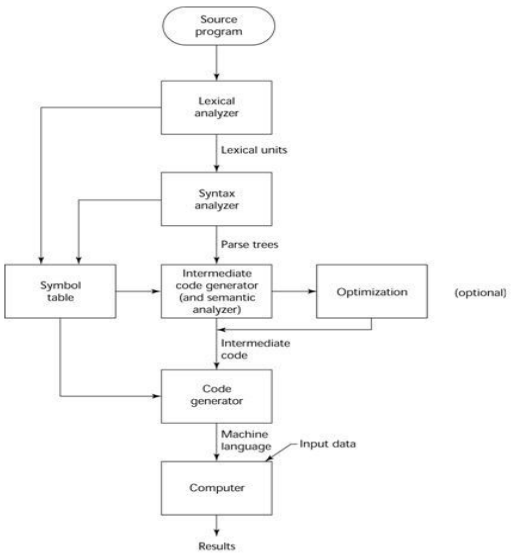
Security threats also happen due to the implementation of global variables, friend functions and pointers.

- **Impractical for complex web applications**
  Web development requires a highly interactive process. However, C++ has quite a slower compilation time. Before compiling completes, C++ will go through several processes that require more time hence makes it slower than any other languages. The process are roughly as follows:
    i) Configuration
    ii) Build tool startup
    iii) Dependency checking
    iv) Compilation
    v) Linking

  Apart from that, web development also needs a set of libraries like String escaping or URL Parameter Parsing which is not included in C++.

3.0 Programming language implementation

Programming language implementation describes how a system executes or translates codes to a language that the machine can understand. All computers have a set of machine language instructions that are it's macroinstructions; however, coding with these instructions is difficult and laborious, necessitating high-level languages. Nevertheless, language implementations must eventually communicate with the computer's processor and memory through machine language instructions. A language implementation system also requires that a computer have a set of software on it called an operating system. The operating system performs common actions such as memory management, disk and file access, input and output Functions (*Basics of Implementation (VCSU-MEP)*, n.d.-a). There are three methods on how programming languages can be implemented which are Compilation, Pure Interpretation and Hybrid. C++ language uses the first method which is a compilation.



The compilation is a process of translating a program from source language to target language. It consumes a longer time to prepare the program because the instructions of the high-level languages have to be converted into machine language macroinstructions for faster execution. The compilation process will start by a lexical analyzer breaking down the source program to lexical units (e.g: operators, identifiers). Syntax analyzer will then gather the syntactic structure and parse tree will be built. The process will then continue with an intermediate code generator where it converts the high-level source to low-level source. Along with this step, semantic errors will be checked by the semantic analyzer in case the glitches were undetectable by syntax analyzer. Optimization will occur if the program must be run quickly and efficiently or if there are any requests for final product optimization. Lastly, the machine language code is generated from the optimized code. One last process is

the symbol table. This "database" holds information such as types and attributes for all user-defined names in the program (such as variables, procedures, and functions)(*Basics of Implementation (VCSU-MEP)*, n.d.-b).

4.0 Programming paradigms

4.1 Imperative style

The imperative programming paradigm is one of the oldest programming paradigms and was developed using machine-language. it's also the idea on which all hardware is implemented, based on Von Neumann architecture. It uses statements that change a program's state. Imperative programming elaborates on describing how a program operates. It consists of statements which are instructions at the native machine-level, and that contain states and variables after the execution of all the results stored to the memory. The benefits of the imperative paradigm are they are very efficient to implement, fast execution time, more efficient as they contain loops, variables and more. Whilst, on the opposite hand, its weaknesses are order sensitive, Complex problems cannot be solved and also the limitation of abstraction as well as parallel programming is not possible.

Example :

```
10 #include <iostream>
11 using namespace std;
12 int main()
13 {
14     int marks[4] = { 75, 82, 65, 93};
15     int sum = 0;
16     float average = 0.0;
17     for (int i = 0; i < 4; i++)
18     {
19         sum = sum + marks[i];
20     }
21     ave = sum / 4;
22     cout<<"Average of 4 test is : "<<ave<<endl;
23     return 0;
24 }
25
```

4.2 Functional style

The functional programming paradigms have been originated from a mathematical discipline and it is language independent. The main feature of this paradigm is the execution of a series of mathematical functions. Build based on the style of building the structure and elements of computer programs, which means that it treats computation as the evaluation of mathematical functions and avoids changing state and mutable data. The central model for the abstraction is the function which is meant for some specific computation and not the data structure. Data are loosely coupled to functions. The function hides their implementation. So this functional programming, the model problem rather than solution Functional paradigm focuses on what to be computed as it accepts input and produces output, it does not have the internal state and it can be replaced with their values without changing the meaning of the program and it. The advantages of functional programming are it is essentially less complex, offers better readability and compared to imperative it has a higher level of abstraction, is not tied to dependency. Some of the weakness includes less efficiency,

troubleshooting variables or it's sequential activities are better handled in both Object-Oriented or imperatively. Some of the languages like perl, javascript mostly use this paradigm.

Example :

```
#include <iostream>
using namespace std;
int factorial(int);
int main()
{
    int a;
    a=factorial(4);
    cout<<"Factorial of 4 is : "<<a<<endl;
 }
int factorial(int x)
{
    if (x == 0) return 1;
    return x*factorial(x-1);
}
```

4.3 Logical style

Logical paradigm is different from other paradigms because it primarily focuses on predicate logic and relation. It contains program statements that express facts and rules about problems within a system of logic. Rules are written as logical clauses with a head and a body. For example, "A is true if C1, C2, and C3 are true." Facts have expressed a kind of like rules but without a body. For example, "A is true." They also play a vital part in the logic circuit of a computer. It is often termed as an abstract model of computation. it'd solve logical problems like puzzles, series and more. In logic programming we have a cognitive content which we all know before, we call it as facts and rules in prolog. So, the results are going to be produced when the question and cognitive content is given to the machine. In normal programming languages, such concepts of cognitive content are not available but while using the concept of AI, machine learning we have some models like Perception model which is using the identical mechanism. The most emphasis is on cognitive content and also the problem. The execution of the program is incredibly very like proof of the statement. The highlight of this paradigm is "problems are solved by the system and providing the validity of a given program is simple" (the University of Central Florida,n,d).

Example:

```
Brother(X,Y) :-                  /* X is the brother of Y */
                               /* if there are two people F and M for which*/
 father(F,X),                    /* F is the father of X */
 father(F,Y),                    /* and F is the father of Y*/
 mother(M,X),                    /* and M is the mother of X */
 mother(M,Y),                    /* and M is the mother of Y */
 male(X).                        /* and X is male*/

"That is X is the brother of Y if they have the same father and mother and X is male."
```

4.4 Object oriented style

Object-oriented programming is a programming paradigm based upon objects that include both data and methods. This paradigm aims to include the benefits of modularity and reusability. The program concept is a set of classes and objects which are meant for communication. The smallest and basic entity is the object and each of the computation is performed on the objects only. The concept of "objects", which might contain data, within the form of fields often known as attributes or properties, and code, within the sort of procedures often known as methods. We write programs in object-oriented programming using classes and objects utilising features of it such as abstraction, encapsulation, inheritance and polymorphism. It is more emphasis is on data rather procedure.

Object-oriented Programming languages are diverse, but the foremost popular ones are class-based, meaning that objects are instances of classes, which also determine their types. It can handle most quite real problems which are today in the scenario. There are several important features of object-oriented programming which are bottom-up approach in program design, programs organized around objects, grouped in classes, focus on data with methods to control upon object's data, the interaction between objects through functions and reusability of design through the creation of latest classes by adding features to existing classes. The advantages of object-oriented programming are data security, inheritance, code reusability and flexibility and abstraction.

Many of the foremost widely-used programming languages like C++, Java, Python are multi-paradigm and they support object-oriented programming to a greater or lesser degree, typically together with imperative, procedural programming. Some samples of object-oriented programming languages are C++, Java, Smalltalk, Delphi, C#, Perl, Python, Ruby, and PHP.

Example: The Circle class

```
// average of 4 test in C++
//Calculate factorial of 4 by calling function factorial code that
//will return average without storage value in memory
//by using C++ programming
#include <iostream>    // using IO functions
#include <string>      // using string
using namespace std;
class Circle {
private:
   double radius;      // Data member (Variable)
   string colour;       // Data member (Variable)
public:
   // Constructor with default values for data members
   Circle(double r = 2.0, string c = "yellow") {
      radius = r;
      colour = c;
   }
   double getRadius() {  // Member function (Getter)
      return radius;
   }
   string getColour() {   // Member function (Getter)
      return colour;
   }
   double getArea() {    // Member function
      return radius*radius*3.1416;
   }
};
```

```
int main() {
   // Construct a Circle instance
   Circle c1(2.2, "pink");
   cout << "Radius=" << c1.getRadius() << " Area=" << c1.getArea()
        << " Colour=" << c1.getColour() << endl;

   // Construct another Circle instance
   Circle c2(2.5); // default colour
   cout << "Radius=" << c2.getRadius() << " Area=" << c2.getArea()
        << " Colour=" << c2.getColour() << endl;

   // Construct a Circle instance using default no-argument constructor
   Circle c3;      // default radius and colour
   cout << "Radius=" << c3.getRadius() << " Area=" << c3.getArea()
        << " Colour=" << c3.getColour() << endl;
   return 0;
}
```

5.0 C++ Evaluation Criteria

5.1 Readability

C++ is a broad language that complicates its readability. There are a number of readability evaluation criteria which include overall simplicity, orthogonality, data type and syntax design. These all will be discussed later on. The overall simplicity of a programming language has a serious impact on its readability. Readability issues arise when the author of the program has learned a different subset from the subset reader is familiar with(Sebesta, 2016a). Problems of readability together with multiplicity features and operator overloading lead to the overall simplicity. In C++, when it comes to feature multiplicity – that is providing more than one way to perform a given operation, C++ has more than one way of executing an operation, like adding +1 to the variable value. For example, the following operations can do the increment, and everything is true:

(1) Increment = Increment + 1;

(2) Increment + = 1;

(3) Increment + +;

(4) + + Increment;

Although in some contexts the last two statements have slightly different meanings to each other and to the others, they all have the same meaning when used as stand-alone expressions (Sebesta, 2016b). This shows that C++ has the same complicated features on multiplicity. Operator overloading is also a factor when evaluating a language's overall simplicity and C++ has more overloading operations compared to Java. Operator overloading is when there are more than one definitions to a single operator symbol which could lead to confusion. Even if overloading can aid readability, it can also be detrimental to readability. For example, in C++ ampersand (&) has completely different meanings in unary (&x) and binary (x & y) operators. Another example is in C++ you can allocate memory pointers and do pointer arithmetic with them so that more complex operations can be done with pointers such as using the incrementing operator to move a pointer's

memory location and it works for all types of the variable pointer (*Writing Assignment - Grade: A - CS 408 - Cal Poly Pomona - StuDocu*, n.d.). Orthogonality a programming language means that a relatively small set of primitive constructs can be combined in a relatively small number of ways to build the control and data structures of the language (Sebesta, 2016b). In simple terms, orthogonality can be described as when one element does not affect others when it changes. For instance, in C++ the expression $x + y$, the type x can give effect to y. The y value can be influenced if y is an integer while x is a pointer. For example, if the float value of four bytes was pointed by x, then the value of y must be scaled which in this scenario should be multiplied by 4 before adding it to x *(Sebesta, 2016b)*. Thus, the treatment of the value of y depends on the type of x and its meaning is influenced by the context of y. Another significant support to readability is the existence of adequate facilities for defining data types and data structures in a language (Sebesta, 2016b). C++ is a language that includes Boolean types and it supports numeric types as Boolean expressions which are flexible but it affects its readability. For example, expression flag = 1 is hard to read and ambiguous but if we replace it with Boolean type flag = true then the expression will be much easier to understand and read *(Xulong Peng,2014)*.

5.2 Writability

Writability is a measure of how simple it is to use a language to create programs for a chosen problem domain (Sebesta, 2016b) and because of it, we will need to consider the language's simplicity, orthogonality, data types, syntax design, support for abstraction and expressivity. If the language increases good readability it also has good writability which shows that writability is highly related to readability. C++ has a lot of data types and supports the abstraction that enhances its writability. Data abstraction refers to covering the background details or implementation from the outside world and only supplying only essential information about the data. In abstraction, we can use access specifiers which are the main pillar of implementing abstraction in C++ (*Abstraction in C++ - GeeksforGeeks*, 2017a). We can enforce restrictions on class members by using access specifiers. For example:

- Members in public class can be accessed from anywhere in the program
- Members of the private class can be accessed only within the class and it is not allowed to be accessed from any part of code outside the class (*Abstraction in C++ - GeeksforGeeks*, 2017b).

```
#include <iostream>
using namespace std;
class implementAbstraction
{
        private:
                double x, y;

        public:
                // method to set values of
                // private members
                void setValue(double a, double b)
                {
                        x = a;
                        y = b;
                }
                void displayValue()
                {
                        cout<<"x is " <<x << endl;
                        cout<<"y is " << y << endl;
                }
};
int main()
{
        implementAbstraction obj1;
        obj1.setValue(110, 230);
        obj1.displayValue();
        return 0;
}
```

A rich collection of operators that enables the programmer to solve different or difficult problems with a very small program making it very writable and providing an efficient expressiveness in C++. For example, in C++ the incrementing and decrementing operator, instead of writing x + 1 to a variable, we can shorten it by writing it as x + +. The powerful '++' operator makes it more convenient in writing variable increments. Efficient expression helps writability. The elements that control the flow of program execution in the source code is called control statements. Flexible and efficient control statements help

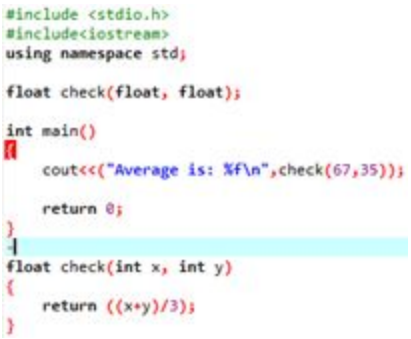
in increasing writability of a language in solving complicated problems. For example, in C++ there are two types of control statements which are conditional statements and unconditional statements. C++ uses many conditional statements such as if-else, while, do-while and for. Below is one of the examples of the if-else conditional statement:

```
if (condition){

        statement(s);

}
```

Unconditional statements in C++ include goto, break and continue. The distinguishing between the upper and lower case identifiers also hinders and helps writability (*Website*, n.d.). Expressivity, however, is greatly enhanced as it supports abstraction. The identifiers in all caps in C++ is usually a macro or global variable. Usually, structs and classes begin with capital letters (*Website*, n.d.). There are no rules that should be followed but these are the things that are common among programmers or we could say that it has been a common law to the programmers. These rules make it easier to write. The variables enjoy abstractions because variable tags such as in 'global_Myglobal' or 'tag_structs_Mystruct' do not need to be included *(Website, n.d.)*.The programmer can make the programs look more expressive by using case sensitive syntax. For example Mykelas mykelas; ". The mykelas is an instance of the class Mykelas (*Website*, n.d.). Without case-sensitive syntax, this kind of expressivity would not be allowed in C++. C++ supports generic types, such as template, which has its advantage for improving writability *(Xulong Peng, 2018).*

5.3 Reliability

Another criteria that will be evaluated in a language is the language's reliability which it can be said that a program is reliable if a program operates under all conditions according to its specifications. The major factors that can affect a program's reliability are readability, writability, type checking, exception handling and restricted aliasing. C++ is reliable if the programmer used it carefully. Type checking means simply checking by the compiler or during program execution for type errors in a given program (Sebesta, 2016b). C++ is statically typed language which means during compile time all types must be known by the compiler. For example, the C++ statement is *int x = 3;* the compiler needs to know the data types of variable '*x*'. As mentioned above, type checking is used to test for type errors in a given program. For example, if a function is expecting a value in float but receives in integer instead. In C++if the type class may have already existed, you might also run into "does not name a type" class error because the compiler thinks this class type has not been defined. It could be a situation where an operation is performed on an integer with the intention that it is a float, or something like adding together a string and an integer *(Thecodeboss.dev 2015)*: *x= 1+"2"*. C++ is a language with coercion which is less reliable than language that has no coercion. Coercion can make the program not detect errors. For example: "int a; float b; cout<<a+b; ". The compiler will detect any error because the value of a will be coerced to float type. Dangling pointer and memory leaking can happen when using



the pointer type even though the pointer type is very flexible. Other than that, exception handling helps to improve the reliability of C++ where this language provides a comprehensive exception handling system that uses the try and catch block and it can catch runtime errors, fix the problem and continue the program *(Xulong Peng, 2018).* The general form of try and catch is : *(Sebesta, 2016b)*

```
try {
//** Code that might raise an exception
} catch (formal parameter) {
//** A handler body
}
. . .
catch (formal parameter) {
//** A handler body
}
```

Lastly, aliasing could reduce language's reliability and could lead to error. For example, "int x=5; int *y=&x" which shows that both variables refer to the same memory cell *(Xulong Peng, 2018).* The language's reliability increases when it has better writability and readability.

5.4 Cost

C++ is a large language. Training programmers, writing programs, compiling programs, executing programs, language implementing systems, poor reliability and maintaining programs are the factors that will be evaluated in the cost of C++. Firstly, the cost of training programmers is a function of the simplicity and orthogonality of the language and the experience of the programmers (Sebesta, 2016b). It is hard to learn because understanding the major features will take a few months, especially for a new programmer to be able to write a middle-level program. Secondly, the cost of writing programs depends on the applications. Generally speaking, writing in C++ language is simple since its syntax is very straightforward. For example:

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

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

(*tutorialspoint.com, C++ Basic Syntax)*

C++ is backwards compatible with C since C++ is directly developed from the C language. It gives low cost for C programmers because they already learn C and it will give advantages to them in learning C++. Other than that, C++ has inexpensive compilers which decrease the cost of compilation and execution. C++'s flexibility and efficiency help in speed up execution which does not require run-time type checking. C++ may increase the cost on the reliability and maintenance but it may cost more to ensure greater reliability and trade-offs mean it decreases other costs with the cost of reliability maintenance *(Xulong Peng, 2018).* For example, C++ does not check index ranges of arrays as the trade-off, it increases execution speed and it is necessary for many C++ applications because system execution speed has higher priorities. C++'s syntax and constructs style also improve its maintainability *(Xulong Peng, 2018).*

6.0 Lexical, Syntax, Semantics Analysis

When a programmer writes code, there might be errors here and there. Hence, during the execution process, there will be a lot of bugs that will disturb and drift a program from executing and giving the right output. This is where the compiler comes to interplay. Apart from converting high-level language to computer language, the compiler also works as a filter mechanism where it will analyse any glitches and errors found in the code. As mentioned before in Section 1.0 (Programming language implementation), the compilation includes many stages, the first step is the lexical analysis, followed by syntax analysis and semantic analysis.

6.1 Lexical Analysis

```
    for(i = 0; i < 8; ++i){
    if(ch == operators[i])
            cout<<" "<<ch<<" is operator\n";
    }

    for(k = 0; k < 6; ++k){
    if(ch == symbols[k])
            cout<<" "<<ch<<" is symbol\n";
    }

    if(isalnum(ch)){
        buffer[j++] = ch;
    }
    else if((ch == ' ' || ch == '\n') && (j != 0)){
            buffer[j] = '\0';
            j = 0;

            if(isKeyword(buffer) == 1)
                cout<<" "<<buffer<<" is keyword\n";
            else
                cout<<" "<<buffer<<" is indentifier\n";
    }

}

fin.close();

return 0;
```

Inputfile:

```
int a , b ;
float d ;

if ( a < b )
  d = a + b ;
```

The compiler is responsible for converting high-level languages to computer languages. It includes many stages, the first step is the lexical analysis. Lexical analyser reads and translates the character stream from source code into a token stream. Once the input file is open and read, the output would be int, float, if as keyword; a, b, d as an identifier; “,”, “;”, “(“, “)” as a symbol and “+”, “<”, “=” as an operator.

Errors that are found during lexical analysis are called lexical errors. Lexical errors mean any sequence of characters that does not match the pattern of any token. To make it simple, lexical errors can be spelling errors, exceeding the length of an identifier or numeric constants, the appearance of illegal characters in the code and many more. When the code enters the lexical analysis phase, it will be read by analyzer character by character. This phase will then convert lexemes found in the code with tokens. Consider the following.

Line 5 of this code contains lexemes which are:

```
 9  #include <iostream>
10  int main()
11  {
12     int a, b=10, c=345;
13     a = b + c;
14     cout<<"Value of a is: "<<a;
15  }
```

a		=		b		+		c	;

The lexical analyzer will replace the lexemes with tokens. A lexeme is a particular instance of a token or a stream of characters that are matched by a pattern or a token. For every lexeme, there are predefined rules called patterns (regular expression) where it identifies whether a token is valid or not. These rules are defined by grammar rules, by means of a pattern. A pattern explains what can be a token, and these patterns are defined by means of regular expressions. Tokens, on the other hand, are like identifiers, operators, literals, keywords, special symbols and so on. For instance, tokens in line 5 will be generated as follows. Notice that the whitespace is removed. Basically, that is how a lexical analyzer approved a statement.

a identifier	= operator	b identifier	+ operator	c identifier	; Special symbol

After being converted, the tokens are then compared with the patterns/regular expression to verify whether it is valid or not. Based on the above example, the statement executes successfully as every token generated is in line with the valid tokens in the regular expression. However, in line 6, notice that there is a grammatical error in 'Cout' whereas it is supposed to be 'cout'. How does lexical analyzer detect this error? It applies the same concept. 'Cout' is a valid identifier, so it must be converted to a token. When the token is compared to the patterns, it will find that the token cannot be recognized thus errors will be generated. This bug needs error recovery solution thus it triggers panic mode recovery. Four recovery options are :

1. deleting the successive character,
2. inserting missing character,
3. replacing an incorrect character with a correct character or
4. transposing two adjacent characters.

In this case, the third option is applied.

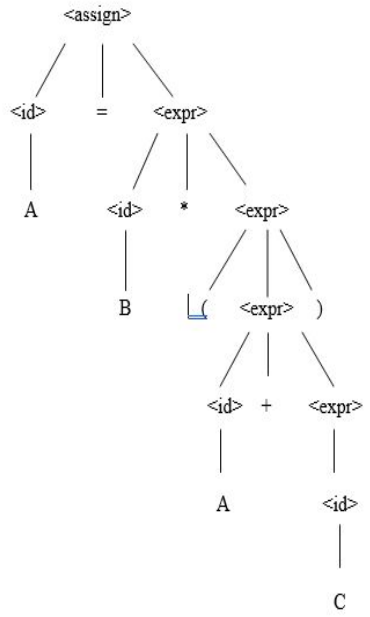
6.2 Syntax Analysis

Syntax analysis is a second phase of the process of compiler design which comes after lexical analysis. The syntactic structure of the given input is analyzed. It checks whether the given input is in the correct syntax of the programming language in which the written input is stored.

C++ statement: A=B*(A+C)

<assign> → <id>=<expr>
<id>→A|B|C
<expr>→<id>+<expr>
|<id>*<expr>
|(<expr>)
|<id>

Parse tree:



We now know that lexical analyzer enables token identifying processes to happen with the assistance of regular expressions or patterns. However, regular expressions have their own limitations where it cannot check balancing tokens, for instance, parenthesis. The symbol **'{'** is a valid token, but if it does not come with its pair, errors will be generated. So how does the bug be detected? Basically, it applies the use of context-free grammar. Context-free grammar has ways of defining syntax which are:

1. **A set of terminal symbols** (basic symbols from which strings are formed.)
2. **A set of non-terminal symbols** (syntactic variables that denote sets of strings.)
3. **A set of production rules of the form** (The productions of a grammar specify the manner in which the terminals and non-terminal can be combined to form strings)
4. One of the non-terminals is designated as the start symbol (S); from where the production begins(*Compiler Design - Syntax Analysis - Tutorialspoint*, n.d.)

Syntax analyzers have two aims: to detect syntax errors in a given program and to produce a parse tree for a given program. Syntax analyzers are either top-down, meaning they construct leftmost derivations and a parse tree in top-down order, or bottom-up, in which case they construct the reverse of a rightmost derivation and a parse tree in bottom-up order(Sebesta, n.d.). Here, we are going to discuss the use of parse trees.

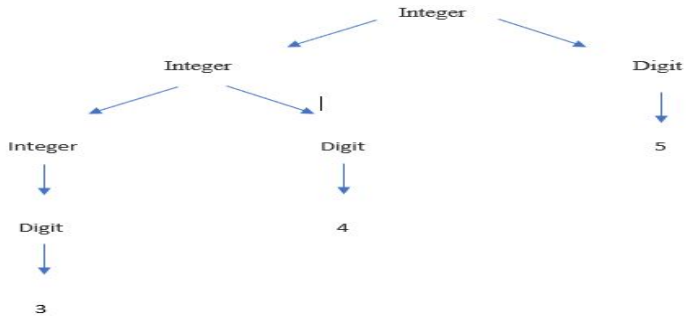
```
#include <iostream>
int main()
{
   int a, b=10, c=345;
   a = b + c;
   cout<<"Value of a is: "<<a;
}
```

Consider the following grammar/production rules:-

Integer: Digit | Integer digit

Digit: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10

Based on line 12 in the code, how do we justify whether the value 345 is an integer? Parse trees can solve this. Below is the generated parse tree that approves 345 as an integer as it complies with the grammar given. If the input integer is 'abc', a syntax error will be generated because the given input does not comply with the production rules above. According to the production rules given, an integer consists of digit or integer digit where the digit is in the range of number 0 until 10. Hence, any input that does not fulfil the standard fixed, will not be accepted. Below is the parse tree generated for determining the input entered:



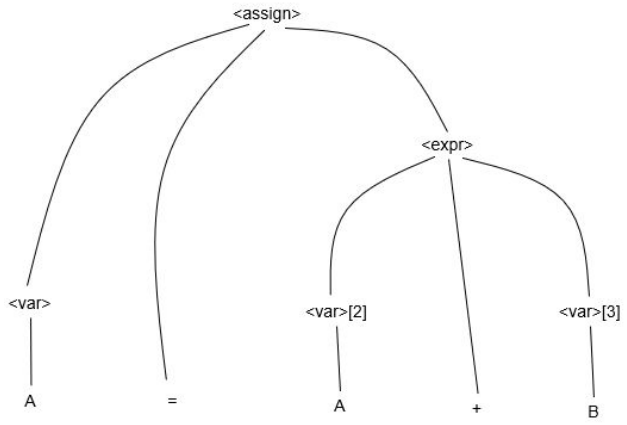
3.0 Semantic Analysis

Semantic Analysis ensures that declarations and statements of a program are correct semantically. To validate a statement whether it is semantically correct or not, attribute grammar is used. An attribute grammar is an add-on to context-free grammar. It allows some characteristics of the structure of programming languages that are either difficult or impossible to be described using BNF. An attribute grammar has 4 components which are grammar, a set of attributes, a set of attribute computation functions, and a set of predicates that describe static semantics rules. The static semantics defines whether a syntactically valid sentence has a meaning. Attribute grammar links syntax with semantics, hence every production rule has semantic rules with action to modify values of attributes of non-terminal. Attributes have values that hold semantic information about the non-terminal. For example, 'A.a' is an attribute of a non-terminal <A>. Besides, semantic rules will be used to produce syntax trees and enforce static semantics. Below is an example of attribute grammar for simple assignment statements:

1. Syntax rule: <assign> → <var> = <expr>
   Semantic rule: <expr>.expected_type ← <var>.actual_type
2. Syntax rule: <expr> → <var>[2] + <var>[3]
   Semantic rule: <expr>.actual_type ←
   if (<var>[2].actual_type = int) and
   (<var>[3].actual_type = int)
   then int
   else real
   end if
   Predicate: <expr>.actual_type == <expr>.expected_type
3. Syntax rule: <expr> → <var>
   Semantic rule: <expr>.actual_type ← <var>.actual_type
   Predicate: <expr>.actual_type == <expr>.expected_type
4. Syntax rule: <var> → A | B | C
   Semantic rule: <var>.actual_type ← look-up(<var>.string)

The look-up function looks up a given variable name in the symbol table and returns the variable's type.

And below is the parse tree:



If all attributes were inherited, the tree could be decorated in top-down order. And, if all attributes were synthesized, the tree could be decorated in bottom-up order.

When rules are enforced by the compiler by generating code to perform the checks, Dynamic semantic will occur. Dynamic semantic denotes meaning, of the expressions, statements, and program units of a programming language. Dynamic semantics are as below:

Operational Semantics

Operational semantics describe the meaning of the statement step by step of a program.

C++ statement: D=(B+A)*C

Meaning: Variable B and A is added together as long as it's not zero. Once it's summed up together, it will multiply with variable C for as long it's not zero value.

Denotational Semantics

Denotational semantics is describing the meaning of programs and based on recursive function theory.

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

int fib(int n)
{
    if (n==0) return 0;
    if (n==1) return 1;
    return fib(n-1) + fib(n-2);
}

int main ()
{
    int k=3;
    cout << fib(k);
}
```

M_{fib} (‘0’)= 0

M_{fib} (‘1’)= 1

M_{fib} (<decimal_num> - ‘1’)= decimal_num - 1

M_{fib} (<decimal_num> - ‘2’)= decimal_num - 2

M_{fib} (<decimal_num>)= M_{fib} (<decimal_num>-’1’) + M_{fib} (<decimal_num>-’2’)

Axiomatic Semantics

Axiomatic semantics describe the significance of a program command by defining its effect on assumptions about the state of the system. The assertions are logical statements which predicate with variables, where the variables describe the program state.

C++ statement: D=2*(x+1) {D>1}

Post-condition: {D>1},
Weakest precondition: {x>0}
Valid precondition: {x>10}, {x>100}, {x>1000}

7.0 Applications of C++

There are hundreds of programming languages out there in the market, but when it comes to performance nobody challenges the big daddy which is the C++ programming language. No doubt C++ is a tough language to learn when comes to its Advanced concepts, but still, it is the language of choice for most of the game developers. Big companies like Google, Facebook and others deeply rely on the power of this language. Most of the applications in the real-world nowadays had been implemented using C++ programming language. That's why it widely used and popular.

C++ is used in many real-world applications, among them is game and game engine development.

The performance is important for a game, therefore in this area the choice of a programming language is limited. C++ turns out to be one of the best options for 3D, multiplayer or other games since it is the fastest programming language until now. For instance, Counterstrike, StarCraft: Brood War, Diablo I, World of Warcraft are all written in C++. Not to mention Xbox and PlayStation consoles which are based on C++ programming. The core of the Unity game engine also used C++ development. And Unity is the most popular engine for building video games, targeting multiple OS at once. Even the most intensive game graphics can be handled with C++ software development. It helps you customize and adjust how exactly the data structures and memory resources of the games should be used. Consequently, you have full control over the development of the game.

Then, C++ is close to the hardware, can easily manipulate resources, provide procedural programming over CPU intensive functions and is fast. It can also overcome the complexities of 3D games and provides multilayer networking. All these advantages of C++ make it as a primary choice to develop the gaming systems as well as game development suites.

In addition, C++ overrides the complexities of 3D games, optimizes resource management and facilitates multiplayer with networking. The language is very fast and it has been widely used in the development of gaming engines since it allows procedural programming for CPU intensive functions and provides greater control over hardware. For example, the science fiction game Doom 3 is a game that used C++ well and the Unreal Engine, a suite of game development tools, is written in C++.

Next, C++ is also used in writing database management software. The two most popular databases MySQL and Postgres are written in C++.

MySQL, one of the most popular database software that is used widely in many real-world application is written in C++. This is the world's most popular open-source database written in C++ and is used by most of the organizations. It has downloaded or distributed more than 100 million copies of its applications in its entire history. MySQL has been used by many of the world's largest and fastest-growing organizations since it can save time and money, powering their high-volume websites, critical business systems, and packaged software. MySQL open-source C++ code can be found at GitHub. With its proven performance, reliability and ease-of-use, MySQL has become the leading database choice for web-based applications such as Google and Wikipedia and the software forms the backbone of a variety of database-based enterprises, used by high profile web properties including Facebook, Twitter, YouTube, Yahoo! and many more.

The SQL parser is written in yacc, but a home-brewed lexical analyzer is used. MySQL works on several system platforms, like AIX, BSDi, FreeBSD, HP-UX, eComStation, i5/OS, IRIX, Linux, macOS, Microsoft Windows, NetBSD, Novell NetWare, OpenBSD, OpenSolaris, OS/2 Warp, QNX, Oracle Solaris, Symbian, SunOS, SCO OpenServer, SCO UnixWare, Sanos and Tru64. There is also a port of MySQL to OpenVMS.

The MySQL server software itself and the client libraries use dual-licensing distribution. They are offered under GPL version 2, or a proprietary license. Support can be obtained from the official manual. Additional free support is available in various IRC and forums. Oracle provides paid support through its software MySQL Enterprise products which they differ in the scope of services and in price. Additionally, a number of third party organizations exist to provide support and services, including MariaDB and Percona.

MySQL has received positive feedbacks, and reviewers have found that "in the average case it works extremely well" and that "developer interfaces are there, and the documentation is very, very good". It has also been tested to be a "fast, stable and true multi-user, multi-threaded SQL database server". As for Oracle, they are driving progress in MySQL, delivering new capabilities to power next-generation web, cloud, mobile and embedded applications. C++ had played the role helps Oracle and MySQL in managing data by the use of its features like file handling, high speed, and reliability, classes and objects, and functions.

Finally, web browsers which are Mozilla Firefox and Thunderbird used C++ programming language.

The development of specialized languages like PHP and Java makes C++ is only adopted for scripting of websites and web applications. However, when it comes to speed and reliability, C++ is still preferred above others. For instance, Google implements C++ as a part of its backend. Not only that, the rendering engine of a few open-source projects like web browser Mozilla Firefox and email client Mozilla Thunderbird also apply C++.

Because both of them are open source projects, their source codes can be found on Mercurial Repositories. For programming web browsers, C++ would most likely be a choice for its high speed. Not only that, but it also offers and completes with data security implemented through the concept of object-oriented programming.

Because the rendering engines demand faster execution, C++ would be the top choice for various web browsers as it offers high speed. The rendering engines have to make sure that users don't have to wait for the content to come up on the screen. Because of that, most rendering software will use C++.

As for information, C++ has the following 2 features that make it a preferred choice in most of the applications:

- **Speed:** C++ is faster than most other programming languages and it provides excellent concurrency support. Those areas where performance is quite critical, it would be really useful. In fact, the latency required is very low. These kinds of requirements occur all the time in high-load servers such as web servers, application servers, database servers, etc..
- **Closer to hardware:** C++ is closer to hardware than most other programming languages like Python, etc. This makes it useful in those areas where the software is closely coupled with hardware and low-level support is required at the software level.

Some of the other interesting features of C++ are:

- **Object-oriented:** C++ is an object-oriented programming language. It mainly focuses on "objects" and manipulations around these objects. Information about how these manipulations work is abstracted out from the consumer of the object.(Goel et al.)
- **Rich library support:** Through C++ Standard Template Library (STL) many functions are available that help in quickly writing code. For instance, there are standard libraries for various containers like sets, maps, hash tables, etc.(Goel et al.)

- **Speed:** C++ is the preferred choice when the latency is a critical metric. The compilation, as well as the execution time of a C++ program, is much faster than most other general-purpose programming languages(Goel et al.).
- **Compiled:** A C++ code must be first compiled into low-level code and then executed, unlike interpreted programming languages where no compilation is needed.(Goel et al.)
- **Pointer Support:** C++ also supports pointers which are widely used in programming and are often not available in several programming languages.(Goel et al.)

It is one of the most important programming languages because almost all the programs/systems that you use have some or the other part of the codebase that is written in C/C++. Be it Windows, be it the photo editing software, be it your favourite game, be it your web browser, C++ plays an integral role in almost all applications that we use.

References

Abstraction in C++ - GeeksforGeeks. (2017b, May 27). GeeksforGeeks. https://www.geeksforgeeks.org/abstraction-in-c/

Basics of Implementation (VCSU-MEP). (n.d.-b). Retrieved March 10, 2020, from http://euler.vcsu.edu:7000/2102/

Compiler Design - Syntax Analysis - Tutorialspoint. (n.d.). Retrieved March 28, 2020, from https://www.tutorialspoint.com/compiler_design/compiler_design_syntax_analysis.htm

Priyadarshini, M. (2018, November 16). *The Huge Security Problem With C/C++ And Why You Shouldn't Use It*. Fossbytes. https://fossbytes.com/security-problem-with-c-c-and-why-you-shouldnt-use-it/

Sebesta, R. W. (n.d.). *Concepts of Programming languages Eleventh Edition* (11th ed., p. 218).

Sebesta, R. W. (2016a). *Concepts of Programming Languages*. Pearson.

Sebesta, R. W. (2016b). *Concepts of Programming Languages, Global Edition*. Pearson Higher Ed.

Soffar, H. (2017, January 18). *C++ programming language advantages and disadvantages | Science online*. Science Online. https://www.online-sciences.com/programming/c-programming-language-advantages-and-disadvantages/

Website. (n.d.). Retrieved March 30, 2020, from http://b.web.umkc.edu/buckleyb/cs_441.html

Writing Assignment - Grade: A - CS 408 - Cal Poly Pomona - StuDocu. (n.d.). StuDocu. Retrieved March 30, 2020, from https://www.studocu.com/en-us/document/california-state-polytechnic-university-pomona/programming-languages/essays/writing-assignment-grade-a/1834036/view

Xulong Peng (2014), "Programming Language C++".

tutorialspoint.com, "C++ Basic Syntax". Retrieved from
https://www.tutorialspoint.com/cplusplus/cpp_basic_syntax.html

Neeraj Mishra (2017), "Lexical Analyzer in C and C++". Retrieved from
https://www.thecrazyprogrammer.com/2017/02/lexical-analyzer-in-c.html

Lộc Nguyễn (2014), "Syntax analysis: an example". Retrieved from
https://www.codeproject.com/Articles/841749/Syntax-analysis-an-example

Sonal Tuteja (n.d), "Recursion". Retrieved from https://www.geeksforgeeks.org/recursion/

Bhumika_Rani,"Introduction of Programming Paradigms"-GeeksforGeeks.Retrieved from
https://www.geeksforgeeks.org/introduction-of-programming-paradigms/

The four different programming paradigms - Groope Multimedia.Retrieved from
https://groope.io/the-four-different-programming-paradigms/

Wikipedia contributors. (2020, March 15). Functional programming. In Wikipedia, The Free Encyclopedia. Retrieved 15:07, March 19, 2020, from
https://en.wikipedia.org/w/index.php?title=Functional_programming&oldid=945752000

KumariPoojaMandal,"Functional Programming Paradigm" - GeeksforGeeks
https://www.geeksforgeeks.org/functional-programming-paradigm/

Functional Programming in C++-Codeproject.com
https://www.codeproject.com/Articles/1267996/Functional-Programming-in-Cplusplus

L Definitions(2019),"What is Logic Programming?"
https://www.computerhope.com/jargon/l/logic-programming.htm

OOAD - Object Oriented Paradigm - Tutorialspoint
https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_paradigm.htm

Introduction of Programming Paradigms - GeeksforGeeks
https://www.geeksforgeeks.org/introduction-of-programming-paradigms/

Wikipedia contributors. (2020, March 17). Object-oriented programming. In Wikipedia, The Free Encyclopedia. Retrieved 14:57, March 19, 2020, from
https://en.wikipedia.org/w/index.php?title=Object-oriented_programming&oldid=945966882

Chaitanya Singh."OOPs Concepts in C++"- beginnersbook.com
https://beginnersbook.com/2017/08/cpp-oops-concepts/

Chua Hock Chuan (2013), "C++ Programming Language Object-Oriented Programming (OOP) in C++". Retrieved from https://www3.ntu.edu.sg/home/ehchua/programming/cpp/cp3_OOP.html

Anonymous. (2020, January 18). "Why and where should you still use C/C++ languages? - Hackernoon". Retrieved from https://hackernoon.com/why-and-where-should-you-still-use-cc-languages-6l1r838gh

Aman Goel. (2020, February 29). "C++ Language: Features, Uses, Applications & Advantages". Retrieved from https://hackr.io/blog/features-uses-applications-of-c-plus-plus-language

Anonymous. (2020, April 1). "Top 10 best applications written in C/C++ - mycplus.com". Retrieved from https://www.mycplus.com/featured-articles/top-10-applications-written-in-c-cplusplus/

Anonymous. (2020, February 3). "7 Mind-Blowing C++ Application that you Must Know - Data Flair". Retrieved from https://data-flair.training/blogs/cpp-application/

Arvind Rongala. (2015, March 16). "Applications of C / C++ in the Real World". Retrieved from https://www.invensis.net/blog/it/applications-of-c-c-plus-plus-in-the-real-world/

Anonymous. (2019, November 10). "What Is C++ Used For? Top 12 Real-World Applications And Uses Of C++ - Software Testing Help". Retrieved from https://www.softwaretestinghelp.com/cpp-applications/

Anonymous. (2020, April 3). "MySQL - Wikipedia The Free Encyclopedia". Retrieved from https://en.wikipedia.org/wiki/MySQL