# White paper for C+

Rockerz Team

October 28, 2020

This document is generated by LaTeX

## Contents

# 1 Introduction

The C+ programming language is a high level and procedural type of programming language which supports Object Oriented programming, Lexical scoping, Function Overloading and Nested Comments. When considered either C or C++, some features are exclusive to only of the language. So in order to cover these exclusive features and also to support additional functionalities we came up with our own language C+ which is a union of C and C++. This C+ can be considered as a prototype as it is not much helpful to work with hard core programs. The main purpose of designing this language is to cover up the drawbacks of C and C++ by covering the features of both these languages in single language and there by producing a new language C+.

# 2 Operators

C+ language supports a wide variety of operations. Most of the operators are similar to C. The different types of operators that are allowed by C+ are given here.

## 2.1 Arithmetic Operators

All the arithmetic operators that are supported by C such as addition(+), subtraction(-), multiplication(*), division(/), remainder(%) are also supported by C+.

## 2.2 Logical Operators

C+ supports the logical operators such as logical OR(||), logical AND(&&) and logical NOT(!). In addition to these, keyword 'and' can also be used for logical AND, similarly 'or' for logical OR and 'not' for logical NOT.

## 2.3 Bitwise Operators

All the Bitwise Operators - Bitwise OR (|), Bitwise AND(&), Bitwise XOR(^), Bitwise NOT or one's complement(∼), Bitwise Left Shift(≪), Bitwise Right Shift(≫) are supported by C+.

## 2.4 Relational Operators

All the Relational Operators - Less than(<), Greater than(>), Less than or equal to(<=), greater than or equal to(>=), equality(==), Not equality(! =) are supported by C+.

## 2.5 Assignment Operators

A wide variety of assignment operators like $=, + =, - =, * =, / =, \% =, |=, \& =, \hat{} =, \ll =, \gg =$ are supported by C+.

## 2.6 Predefined Constants

As some of the universal constants are used many times, we included them as global constants and can be used anytime. There are two predefined constants we introduced in this language

1. _pi = 3.14
2. _e = 2.78

# 3 Data Types

The Data types included in C+ are int, char, float, bool.

## 3.1 Int

A Variable of type 'int' occupies 4 bytes of memory which in turn requires 32bits. The highest bit($32^{nd}$ bit) is used to store the sign of the integer. This bit is 1 if the number is negative, and 0 if the number is positive. So the range that can be stored under int-data type ranges from $-2^{31}\ to\ 2^{31} - 1$.

## 3.2 Char

A variable of type 'char' is assigned according to ASCII rules and since ASCII tabulates a total of 128 characters, C+ also supports 128 characters which can be stored using a variable of data type 'char'. A variable of 'char' data type is assigned 1byte of memory.

## 3.3 Float

A 'float' datatype can be used to store any decimal valued constants. It occupies 4 byte of memory and follows IEEE standard representation. It ranges from $-1.2e^{38} to + 3.4e^{38}$.

## 3.4 Bool

C+ supports variable of data type 'bool'. There are 2 boolean constants - 'true' or 'false'.

# 4 Comments

C+ supports two types of comments.

## 4.1 Single Line Comments

A single line comment begin with two back slashes(//) and ends with new line character and comments are ignored by the compiler.

## 4.2 Multi line Comments or Block Comments

A comment extending over multiple lines can be inserted in between /* and */. Unlike C, C+ also support Nested comments.
Example:

```
/*
 * This is a comment
 * Containing Multiple lines
 * /* This is a Nested Comment*/
 */-->
int main(){
    int a = 5;  // This is a single Line comment
}
```

# 5 Decision Control Statements

## 5.1 *if* Statement

For a single statement, it can be simply written as

```
if(expression)
    statement;
```

When the expression inside the if condition evaluates to Boolean true value the statement present below the if condition will be evaluated. For multiple statements, the block of code to be included under the if condition is wrapped with a pair of braces.

## 5.2 *if-else* statement

An else statement can be added to the if statement and will be evaluated when the expression inside the if condition evaluates to Boolean false value. We can also use else if clauses and Nested if-else(an entire if-else construct within either the body of the if statement or the body of an else statement) for multiple evaluation as required.

Example:

```
if (condition1)              //  Simple if
    statement1;
if (condition2){             //  if-else
    if (condition3)          //  nested if
        statement2;
    else if(condition4)      // else if
        statement3;
    else
        statement4;
}
else{
    statement5;
}
```

# 6 Jump statement

These statements are used to jump from one line of execution to other line of execution.

## 6.1 *break* statement

'break' keyword is used to exit from the execution of the code below it which is wrapped under a loop and the line of execution becomes the immediate line after the loop constituting this keyword in the code.

## 6.2 *continue* statement

'continue' keyword can be used to skip the code execution below it which is wrapped under a loop and line execution becomes the first line in the loop constituting this keyword.

## 6.3 *return* statement

'return' keyword can be used to exit from a function and while doing so can also return a constant(can be int or char or float or bool depending on the return type of the function).

# 7 Loop Statements

Loops are used to executing a same sequence of statements for desired no.of times.

## 7.1 *for* Loop

A structure which is used to execute a sequence of statements multiple times. We provide two types of formats:

1. *for ( initialisation ; condition ; increment ){}*
   This is much similar to C 'for' loop but here we provide an optional initialisation, conditional, increment statements.

```
for(initialise counter ; test counter ; increment counter){
    statement1;
    statement2;
    statement3;
}
```

2. *for( int i : arr ){}*
   This structure is known as ranged for loop which is used when we are iterating over a range or over an array elements and it can be used to avoid array out of bounds error. This is similar to as that of ranged for loop in python.

```
for(int i=0; i < sizeof(arr)/sizeof(arr[0]); i++){}
```

   Example:

```
int main(){
     int arr[10] = {0};
     for(int i : arr){
         arr[i] = i+1;
     }
}
```

## 7.2 *while* Loop

Repeats a sequence of statements until the decision making condition becomes false.

```
while (condition) {
    do this;
    increment if required;
}
```

# 8 Array

Array is a data structure which contains group of elements of same data type (int or char or float or bool) to be represented as a single variable. The indexing of the array is begin with 0. The data is allocated in contiguous memory locations.

## 8.1 Initialisation

Initialisation of an array can be done in two ways:-

1. To initialize all the elements of the array to a single value. All the array elements can be initialised to a single value using the following initialisation code.

```
int arr[10] = {10};
```

   The above code will initialise all array elements to 10.

2. To initialize the different elements of the array with different values:-. Each element in the array can be initialised with a specific value by following the structure as described below.

```
int arr[10] = {1,2,3,4,5,6,7,8,9,10};
```

   The above code initialises arr[0] = 1, arr[1] = 2, arr[2] = 3 .... so on.

## 8.2 String

C+ doesn't support any exclusive datatype to store string constants. But enables strings to be stored as an array of characters but the number of characters in the string should be predefined. A string can be initialsed by the following ways:-

```
char arr1[11] = {'H','e','l','l','o',' ','w','o','r','l','d'};
char arr2[11] = "Hello World";
```

## 8.3 Multidimensional Array

C+ supports multidimensional array. This feature enables the users to store multiple strings. A multidimensional array can be initialised by the following ways:-

```
int arr1[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
int arr2[3][3] = {{0}};
```

# 9 Functions

The declaration of function is similar to C and the arguments of the function are separated by comma and the return type should be declared before the function name.

## 9.1 Lexical Scoping

Unlike C and C++, C+ support Lexical Scoping or Nested Functions. Example:

```
int func1(int a, int b){
    int func2(int a,int b){
        return a*b;
    }
    return func2(a,b);
}
```

Here the func2() should be defined before it is called by the func1().

## 9.2 Function overloading

Unlike C, C+ supports Function Overloading. Example:

```
void func(char s[20]){
    printf("Function with string %s",s);
}
void func(int a){
    printf("Function with integer %d",a);
}
void func(int a, char b){
    printf("Function with integer  character %d, %c",a,b);
}
int main(){
    func("cs18btech");
    func(25);
    func(16,'c');
    return 0;
}
```

# 10 OOP

C language doesn't support Object oriented programming but C++ does,So we would like to introduce it in this language.The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function. In C+ we like to support Classes & Objects and facilitate Encapsulation, Polymorphism and also Inheritance.

## 10.1 Classes and Objects

Class is a user-defined Data type, which holds its own data members  member functions. They can be accessed by creating an instance of that class.
An Object is an instance of class. When a class is defined no memory is allocated, but when it is instantiated memory is allocated.

```
class student{                    //student is class
    char name[50];                //Data member
    char rollno[15];
    int marks;
public:
    void get_details(){
    printf("Name is %s", name);
    printf("Roll No is %s", rollno);
    printf("Marks in the Exam: %d", marks);
    }                             //Member function
};
int main(){
    student A;                    //A is an object
    student B;                    //B is an object
}
```

## 10.2 Encapsulation

Generally encapsulation is defined as wrapping up of data information under a single unit, here it is defined as binding together the data and the functions that manipulates them. By this we can hide some data from certain parts of code and make it available for some parts of code.

```
class example{
private:                          //Private variable only accessible for class
members
    char pvt[20];
public:                           //Public functions, variables accessible for
all members of code.
    int public;
    void write(char s[20]){
        pvt = s;
    }
    void read(){
        printf("Private variable is %s", pvt);
        printf("Public variable is %d", public);
    }

};
int main(){
    example ex1;                  //ex1 is an Object
    ex1.public = 25;
  //ex1.pvt = "string";           //This is inaccessible from main()
    ex1.write("string");          //write can access the private variable since
                                  //it is member of class
    ex1.read();                   //Prints the public & private variable
    return 0;
}
```

## 10.3 Polymorphism

Generally polymorphism means having many forms.C+ supports Compile time polymorphism and Which can be achieved by function overloading or operator overloading.
Function Overloading: If there are multiple function with different parameters, and same name.

```
class poly {
    public:
        void func (char s[20]){              //Function with string as argument
            printf("Function with string %s", s);
        }
        void func (int a){                   //Function with integer as argument
            printf("Function with integer %d", a);
        }
        void func (int a, char b){//Function with integer & character as
argument
            printf ("Function with integer & character %d, %c", a, b);
        }
};
int main(){
    poly obj;
    obj.func("cs18btech");                   //1st func is called
    obj.func(25);                            //2nd func is called
    obj.func(16, 'c');                       //3rd func is called
    return 0;
}
```

## 10.4 Inheritance

The capability of a class to inherit properties and characteristics from another class is called Inheritance.
Sub Class: the class that inherits properties from another class is called Sub class or Derived class.
Super Class: The class whose properties are inherited by sub class is called base class or Super Class.
Private members of any class cannot be inherited by any other class, but public, protected members can be inherited. C+ supports only single inheritance i.e a class is allowed to inherit from only one class.

```
class A {
public:
    int x;
    void A(){
        printf("This is class A");
    }
protected:
    int y;
private:
    int z;
};
class B inherits public A {
    // x is public
    // y is protected
    // z is not accessible from B
};
class C inherits protected A{
    // x is protected
    // y is protected
    // z is not accessible from C
};
class D inherits private A {
    // x is private
    // y is private
    // z is not accessible from D
};
int main(){
    B obj;
    return 0;
}
```

# 11 ECOOP Compiler Design

ECOOP(Execptional C with object oriented programming) compiler has been designed for the C+ language and the design is explained here.

## 11.1 Lexer & Parser

Lexer and parser are written combined in ANTLR grammar.

## 11.2 Error Handling

### 11.2.1 Array out of bound error

Trying to access the elements of an array which are beyond the size of the array would result in array out of bound error.

```
#include <stdio.h>
void main()
{
 int array[5]={0,1,2,3,4};
 array[5]=array[5]+array[2];
}
```

The above program would result in Array out of bounds error because the index of array start from 0 that is in the above the indices of the 'array' runs from 0-4 (since size of the array is 5), So therefore accessing array[5] is not acceptable.

### 11.2.2 Floating point error

An operation that leads to unterminating floating point will be reported as Floating point error by the compiler.
Eg:- division with 0.

```
#include <stdio.h>
void main()
{
  int x;
  int y=314;
  x=y/0;
}
```

The above would result in Floating point error because division of an integer with 0 is not legit in this language as it would result in unterminated floating point number.

```
#include <stdio.h>
void main()
{
  int x=22/7;
}
```

The above program wouldn't result in any error even though 22/7 is unterminating because the compiler would just initialize x with the integral part of 22/7 as the x is declared under data type 'int'

### 11.2.3 Variable out of scope error

Every variable in this language has some scope that is the area of the program only through which this variable can be accessed. Accessing the variable from a function

or piece of code in the program which is not in the scope the variable then the compiler would report this is as a Variable out of scope error.

```
#include <stdio.h>
bool find(int arr[],int size, int num)
{
 for(int i=0;i<num;i++)
 {
  if(arr[i]==num)
   break;
 }
 if(i==num)
  return(false);
 else
  return(true);
}
void main()
{
  //program to find whether the given number is present in the array
  int x=5; //given number
  int arr[]={32, 67, 123, 987};
  int n=sizeof(arr)/sizeof(arr[0]);
  find(arr,n,x);
}
```

In the above program, consider the function 'find' in which focusing on the 'for' loop where the variable 'i' is initialized and declared that is the scope of this variable is only within this 'for' loop and therefore by accessing the variable out of this 'for' loop would result in Variable out of scope error.

### 11.2.4 Unterminated string constant

If there is an unescaped new line in a string then the compiler considers this as a lexical error and reports as unterminated string constant error.

```
#include <stdio.h>
void main()
{
 char str[]="The language C+ is cool
 but it is just a prototype"
}
```

The above program would result in Unterminated string constant error as while initializing string 'str', the new line after cool is not escaped.

### 11.2.5 EOF in comment

If there is EOF inside a comment then the compiler considers this as a lexical error and reports as EOF in the comment error.

```
#include <stdio.h>
int length_of_str(char str[])
{
  int i=0;
  while(str[i]!='\0')
   i++;
  return(i);
  /*in this language the end of every string is indicated with a
  special character named null character('\0')*/
}
void main()
{
  char str[]="Mahesh is whitehat Jr's chintu";
  //program to find the length of the given string;
  length_of_str(str);
  //final value of i is the length of the given string
  /*The function 'length_of_str' will return the length of given
  string
}
```

The above program would result in EOF in the comment error because the program reached end of the file even before the multi-line comment in the 'main' function is closed.

### 11.2.6 EOF in string constant

If there is EOF inside a string then the compiler considers this as a lexical error and reports as EOF in the string constant error.

```
#include <stdio.h>
void main()
{
  char str[]="There are not many good anime to watch during this
  COVID-19 pandemic
}
```

The above program would result in EOF in string constant error because the program reached end of the file even before the string which is to be initialised to 'str' is closed or completed.

### 11.2.7 Unbalanced Parentheses

Parentheses have special meaning in regular expressions supported by the lexer of C+. If the parentheses are not balanced considering the rules of lexer then the compiler would report this as Unbalanced parentheses error.

```
#include <stdio.h>
void main()
{
```

```
      for(int i=0;i<10;i++)
      {
       //some code

   }
```

In the above program, the no.of opening parenthesis and no.of closing parenthesis are not equal that is not balanced which results in lexical error named Unbalanced parentheses.

## 12 Standard I/O

C+ supports 2 pre-defined functions 'printf'-to write output to the console and 'scanf'-to read input from from the console. These functions (printf and scanf which are similar to as that of C language) enables the user to control the input and output stream according to their desire.

## 13 Summary

The Features that differs C+ from C are:

1. Object Oriented Programming
2. Nested Comments
3. Lexical Scoping
4. Function Overloading
5. Range specific for loop
6. and, or, not keywords for Logical AND, OR, NOT
7. Predefined Constants