Chapter 2

C++ Syntax and Structure

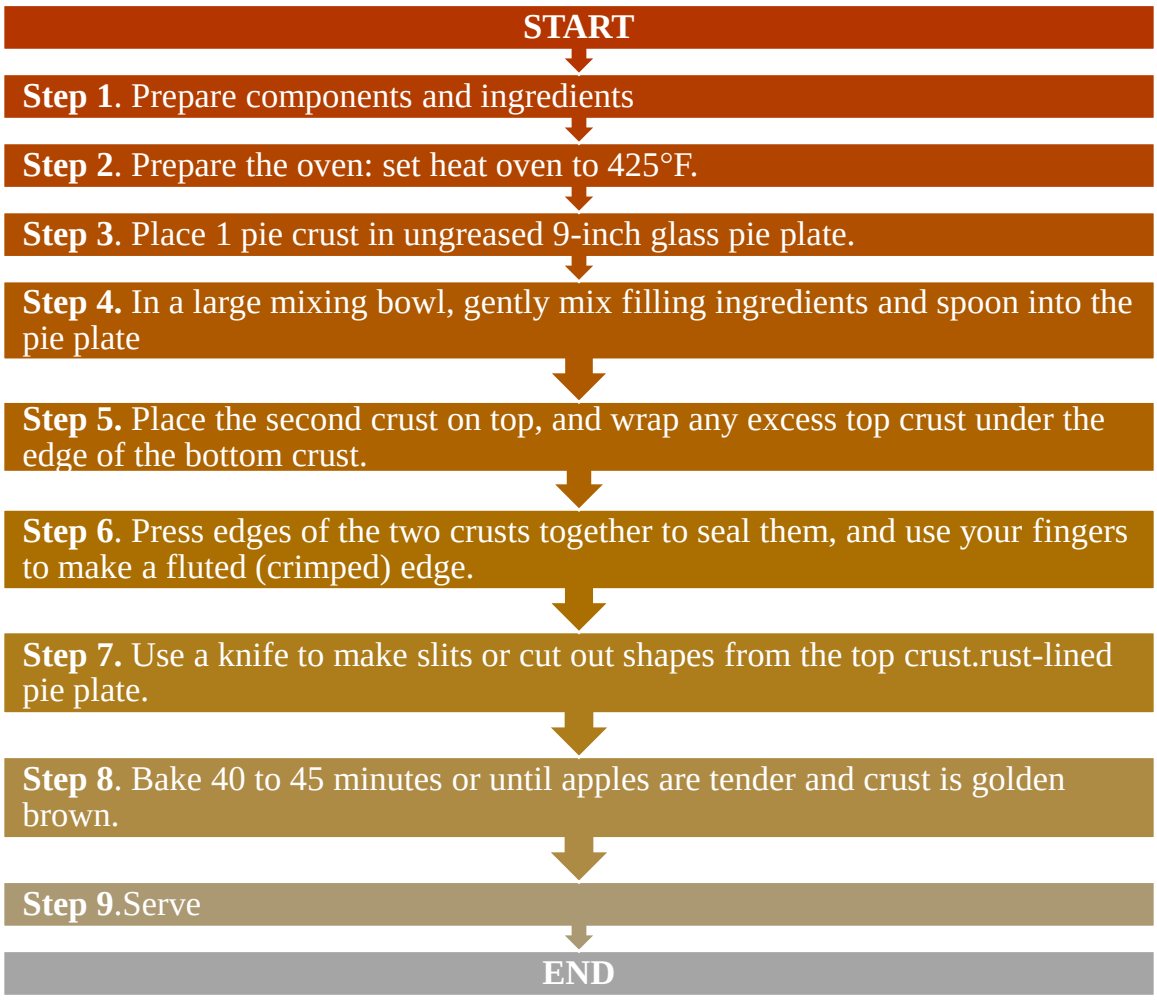
Algorithm

Algorithm in programming is a step-by-step procedure designed to perform an operation. Algorithm have a definite beginning and a definite end, and a set of steps to reach from the beginning to the end.

For example, a recipe to make an apple pie or a problem solving routine are examples of algorithm because a set of steps need to be completed in order to achieve the result.

Algorithm of a program can be organized and presented in steps, pseudocode, or flowchart.

Let's take a look on the steps algorithm of how to make a classic apple pie:



Pseudocode: a semi-programming language used to describe the steps in an algorithm. It uses the structural of a programming language, but is intended for human reading rather than machine reading. Therefore, most of the time, a pseudocode omits the syntax of the programming language.

An example of pseudocode to display a message when a number is greater or less than a number is as:

```
Enter number
If number is greater than 10
Print “Your number is greater than 10”
Else if number is less than 10
Print “Your number is less than 10”
Else
Print “The number is equal to 10”
```

Flowchart: A **flowchart** is a type of diagram that represents an algorithm, workflow or process in a graphical way.

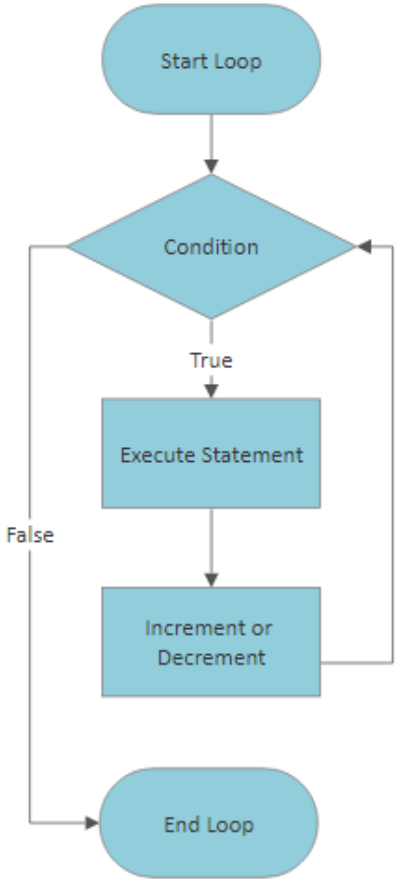


Diagram 1: Flow chart diagram - Image from www.smartdraw.com

Syntax

Syntax refers to the spelling and grammar of a **programming** language. Computers are inflexible machines that understand what you type only if you type it in the exact form that the computer expects. The expected form is called the **syntax**.

Every programming language uses different word sets in different orders, which means that each programming language uses its own syntax. But, no matter the programming language, computers are really exacting in how we structure our syntax.

(What is Syntax in Computer Programming?, n.d.)

C++ syntax

A basic C++ file should have the following structure lines:

- **Header files** are included at the beginning just like in C program. Here `iostream` is a header file which provides us with input & output streams. Header files contained predeclared function libraries, which can be used by users for their ease.
- **Using** `namespace std`, tells the compiler to use standard namespace. Namespace collects identifiers used for class, object and variables. NameSpace can be used by two ways in a program, either by the use of `using` statement at the beginning, like we did in above mentioned program or by using name of namespace as prefix before the identifier with scope resolution (**::**) operator.

```
std::cout << "A";
```

- `main( )` is the function which holds the executing part of program its return type is int.

(Basic Concepts of C++, n.d.)

Comments in C++ Program

Commenting is important in computer programming. Proper use of **commenting** can make **code** maintenance much easier, as well as helping make finding bugs faster. It also help other programmers to follow and read someone else code. Programmers usually use steps or pseudocode to describe each steps in the program.

There are two ways to make comments in C++:

- For single line comments, use // before mentioning comment, like

```
cout<<"single line";   // This is single line comment
```

- For multiple line comment, enclose the comment between /* and */

```
/*this is
  a multiple line
  comment */
```

Libraries and include directives

C++ comes with a number of standard libraries. In fact, it is almost impossible to write a C++ program without using at least one of these libraries. The normal way to make a library available to your program is with an **include** directive. An **include** directive for a standard library has the form

```
#include<library_Name>
```

For example, the library for console I/O is **iostream.** So, most of our demonstration programs will begin

```
#include<iostream>
```

Console input/output

Simple console input is done with the object **`cout`**, **`endl`**, and **`cin`**, all of which are defined in the library iostream.

- ***Output using*** **`cout`**

  The value of variables as well as strings of text may be output to the screen using **`cout`**.

  To display a message in a console, we can just write the message between two double-quotation marks using the object **`cout`**:

  ```
  std::cout << "Hello World!";
  ```

  To display the value of a variable in a console, let's say variable `price`, we write the name of the variable using the object **`cout`**, in this case, we don't write `price` in between two double-quotation mark because `price` is a variable.

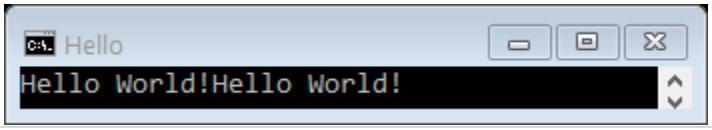
  ```
  price = 10;
  std::cout << price;
  ```

- ***New lines in output*** **`endl`**

  At the end of each line, which is after the semicolon, unless you tell the computer to go to the next line, it will put all the output on the same line.

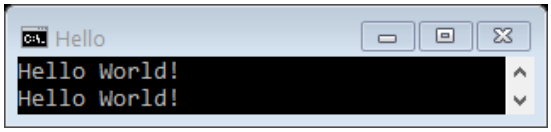
  ```
  std::cout << "Hello World!";

  std::cout << "Hello World!";
  ```



One way to display a new line is using a newline character **\n** Since **\n** is a character, it must be inside the quotes.

```
std::cout << "Hello World!";
std::cout << "\n";
std::cout << "Hello World!";
```



Another way to output a blank line is to use object **endl**, which essentially perform the same as **\n**. Since **endl** is an object, it doesn't be inside the quotes.

```
std::cout << "Hello World!"<<std::endl;
std::cout << "Hello World!";
```

Although **\n** and **endl** performs the same job, they are used slightly different.

Some differences between **endl** and **\n** are:

1. **endl** is an object while **\n** is character.
2. **endl** doesn't occupy any memory whereas **\n** is character so It occupy 1 byte memory.
3. We cannot write **endl** in between double quotation while we can write **\n** in between double quotation like
   cout<<"\n"; it is right but **cout<<"endl";** is wrong.
4. We can use \n both in C and C++ but, **endl** is only supported by C++ and not the C language.

5. The most important difference is **std::endl** flushes the output buffer, and **\n** doesn't. If you don't want the buffer flushed frequently, use **\n** . If you do (for example, if you want to get all the output, and the program is unstable), use **std::endl** .
   (endl vs \n C++, n.d.)

- ***Input using cin***

Object **cin** for input value works almost as **cout** for output. The syntax in similar, except that **cin** uses the double-arrow point in the opposite direction. Since **cin** is used to collect value, the input value must be stored in a variable. Therefore, object **cin** will be covered later in the chapter of variable.

```
int price;
cout << "How much is the tv? \n";
cin >> price;
cout << "The tv price is $ " <<price;
return 0;
```

Example) Write a C++ program to print Hello World!

```
1  /* my first program in C++
2     By: Huixin Wu*/
3
4  #include <iostream>
5
6  int main()
7  {
8    std::cout << "Hello World!";
9    return 0;
10 }
```

Let's examine this program line by line:

Line 1-2: `/* my first program in C++`
`By: Huixin Wu*/`

Enclose the comment between **/*** and ***/** are comments lines inserted by the programmer but which has no effect on the behavior of the program. Programmers use them to include short explanations or observations concerning the code or program. In this case, it is a brief introductory description of the program and its authors.

Line 4: **`#include <iostream>`**

Lines beginning with a hash sign (#) are directives read and interpreted by what is known as the *preprocessor*. They are special lines interpreted before the compilation of the program itself begins. In this case, the directive **`#include <iostream>,`** instructs the preprocessor to include a section of standard C++ code, known as *header iostream*, that allows to perform standard input and output operations, such as writing the output of this program (Hello World) to the screen.

Line 3 and 5: A blank line.

Blank lines have no effect on a program. They simply improve readability of the code.

Line 6: **`int main ()`**

This line initiates the declaration of a function. Essentially, a function is a group of code statements which are given a name: in this case, this gives the name "main" to the group of code statements that follow. Functions will be discussed in detail in a later chapter, but essentially, their definition is introduced with a succession of a type **`int`** a name **`main`** and a pair of parentheses **`( ),`** optionally including parameters.

The function named main is a special function in all C++ programs; it is the function called when the program is run. The execution of all C++ programs begins with the main function, regardless of where the function is actually located within the code.

Lines 7 and 10: **{ }**

The open brace **{** at line 7 indicates the beginning of **main**'s function definition, and the closing brace **}** at line 10, indicates its end. Everything between these braces is the function's body that defines what happens when main is called. All functions use braces to indicate the beginning and end of their definitions.

Line 8: **std::cout << "Hello World!";**

This line is a C++ statement. A statement is an expression that can actually produce some effect. It is the meat of a program, specifying its actual behavior. Statements are executed in the same order that they appear within a function's body.

The **cout** is a predefined object of **ostream** class. It is connected with the standard output device, which is usually a display screen. The **cout** is used in conjunction with stream insertion operator **<<** to display the output on a console.

This statement has three parts: First, **std::cout**, which identifies
The **st**andar**d c**haracter **out**put device (usually, this is the computer screen). Second, the insertion operator **<<** which indicates that what follows is inserted into **std::cout**. Finally, a sentence within quotes **"Hello world!"** is the content inserted into the standard output.

Notice that the statement ends with a semicolon **;** This character marks the end of the statement, just as the period ends a sentence in English. All C++ statements must end with a semicolon character. One of the most common syntax errors in C++ is forgetting to end a statement with a semicolon.

Line 9: **return 0;**

It is not necessary that every time you should use `return 0` to return program's execution status from the `main()` function. But returned value indicates program's success or failure to the operating system and there is only one value that is 0 which can indicate success and other non zero values can indicate failure of execution due to many reasons.

For example - if program's execution fails due to lake of memory we can return -1, if it fails due to file opening we can return -2, if it fails due to any invalid input value we can return -3 and so on. If program's execution is success we should return 0.

Using namespace std

If you have seen C++ code before, you may have seen **cout** being used instead of **std::cout**. Both name the same object: the first one uses its *unqualified name* **(cout)**, while the second qualifies it directly within the **namespace std** (as **std::cout**).

cout is part of the standard library, and all the elements in the standard C++ library are declared within what is called a **namespace**: the **namespace std**.

In order to refer to the elements in the **std namespace** a program shall either qualify each and every use of elements of the library (as we have done by prefixing **cout** with **std::**), or introduce visibility of its components. The most typical way to introduce visibility of these components is by means of *using declarations*:

```
using namespace std;
```

The above declaration allows all elements in the std namespace to be accessed in an *unqualified* manner (without the **std::** prefix).

With this in mind, from the previous example, we can be rewritten to make unqualified uses of **cout** as:

```
1  /* my first program in C++
2     By: Huixin Wu*/
3
4  #include <iostream>
5  using namespace std;
6  int main()
7  {
8    cout << "Hello World!";
9    return 0;
10 }
```

Both ways of accessing the elements of the **std namespace** (explicit qualification and *using* declarations) are valid in C++ and produce the exact same behavior. For simplicity, and to improve readability, the examples in these tutorials will more often use this latter approach with *using* declarations, although note that *explicit qualification* is the only way to guarantee that name collisions never happen.

(Structure of a Program, n.d.)