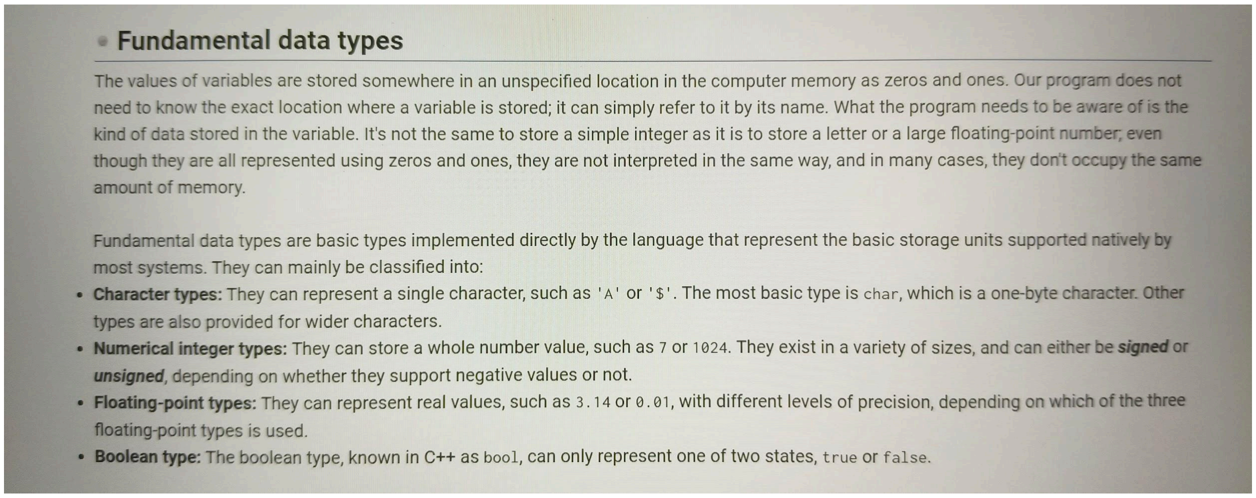
🔴🔴DATA TYPES

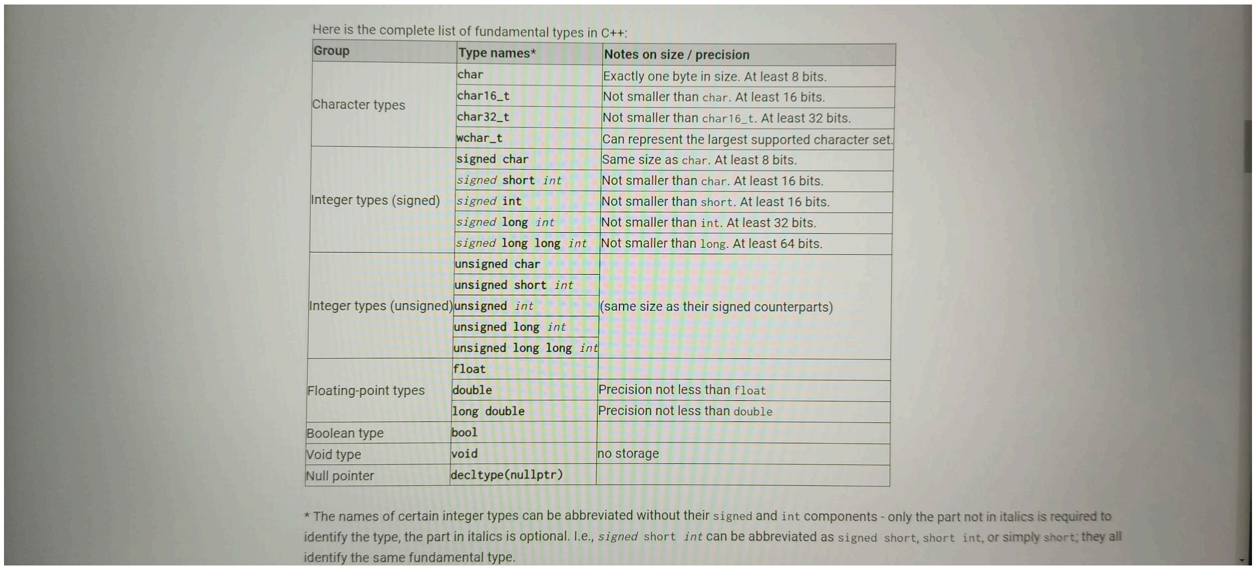


Fundamental data types

The values of variables are stored somewhere in an unspecified location in the computer memory as zeros and ones. Our program does not need to know the exact location where a variable is stored; it can simply refer to it by its name. What the program needs to be aware of is the kind of data stored in the variable. It's not the same to store a simple integer as it is to store a letter or a large floating-point number; even though they are all represented using zeros and ones, they are not interpreted in the same way, and in many cases, they don't occupy the same amount of memory.

Fundamental data types are basic types implemented directly by the language that represent the basic storage units supported natively by most systems. They can mainly be classified into:

- **Character types:** They can represent a single character, such as `'A'` or `'$'`. The most basic type is `char`, which is a one-byte character. Other types are also provided for wider characters.
- **Numerical integer types:** They can store a whole number value, such as `7` or `1024`. They exist in a variety of sizes, and can either be ***signed*** or ***unsigned***, depending on whether they support negative values or not.
- **Floating-point types:** They can represent real values, such as `3.14` or `0.01`, with different levels of precision, depending on which of the three floating-point types is used.
- **Boolean type:** The boolean type, known in C++ as `bool`, can only represent one of two states, `true` or `false`.

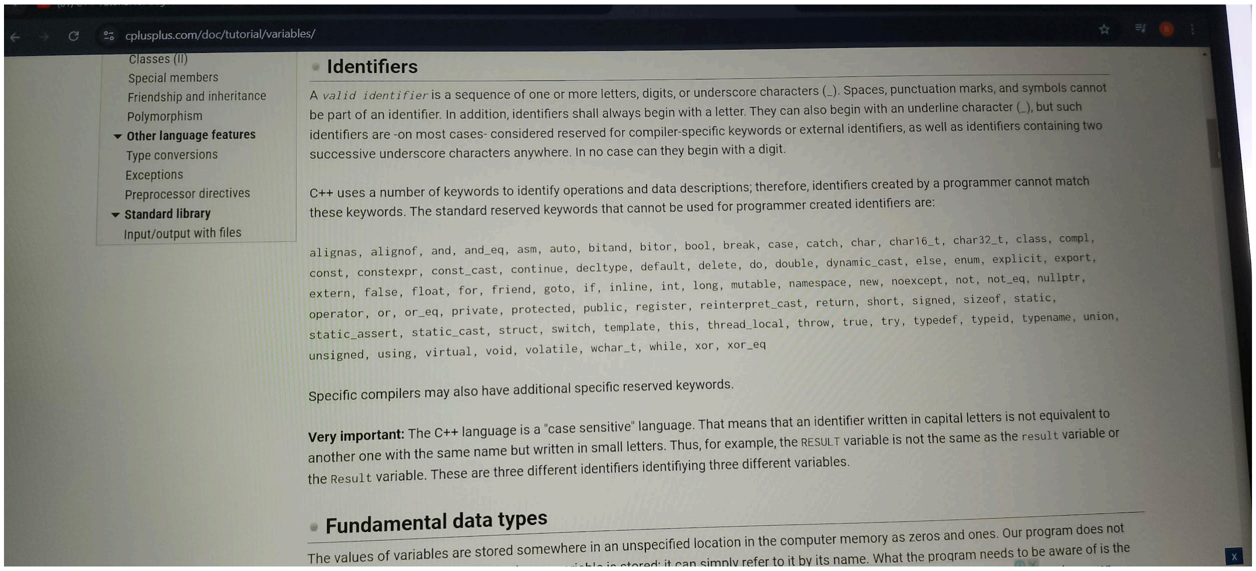


Here is the complete list of fundamental types in C++:

Group	Type names*	Notes on size / precision
Character types	`char`	Exactly one byte in size. At least 8 bits.
	`char16_t`	Not smaller than `char`. At least 16 bits.
	`char32_t`	Not smaller than `char16_t`. At least 32 bits.
	`wchar_t`	Can represent the largest supported character set.
Integer types (signed)	`signed char`	Same size as `char`. At least 8 bits.
	`signed` `short` *`int`*	Not smaller than `char`. At least 16 bits.
	`signed` `int`	Not smaller than `short`. At least 16 bits.
	`signed` `long` *`int`*	Not smaller than `int`. At least 32 bits.
	`signed` `long long` *`int`*	Not smaller than `long`. At least 64 bits.
Integer types (unsigned)	`unsigned char`	(same size as their signed counterparts)
	`unsigned short` *`int`*	
	`unsigned` *`int`*	
	`unsigned long` *`int`*	
	`unsigned long long` *`int`*	
Floating-point types	`float`	
	`double`	Precision not less than `float`
	`long double`	Precision not less than `double`
Boolean type	`bool`	
Void type	`void`	no storage
Null pointer	`decltype(nullptr)`	

* The names of certain integer types can be abbreviated without their `signed` and `int` components - only the part not in italics is required to identify the type, the part in italics is optional. I.e., *`signed`* `short` *`int`* can be abbreviated as `signed short`, `short int`, or simply `short`; they all identify the same fundamental type.

🔴🔴 identifiers



Identifiers

A `valid identifier` is a sequence of one or more letters, digits, or underscore characters (_). Spaces, punctuation marks, and symbols cannot be part of an identifier. In addition, identifiers shall always begin with a letter. They can also begin with an underline character (_), but such identifiers are -on most cases- considered reserved for compiler-specific keywords or external identifiers, as well as identifiers containing two successive underscore characters anywhere. In no case can they begin with a digit.

C++ uses a number of keywords to identify operations and data descriptions; therefore, identifiers created by a programmer cannot match these keywords. The standard reserved keywords that cannot be used for programmer created identifiers are:

```
alignas, alignof, and, and_eq, asm, auto, bitand, bitor, bool, break, case, catch, char, char16_t, char32_t, class, compl,
const, constexpr, const_cast, continue, decltype, default, delete, do, double, dynamic_cast, else, enum, explicit, export,
extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, noexcept, not, not_eq, nullptr,
operator, or, or_eq, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static,
static_assert, static_cast, struct, switch, template, this, thread_local, throw, true, try, typedef, typeid, typename, union,
unsigned, using, virtual, void, volatile, wchar_t, while, xor, xor_eq
```

Specific compilers may also have additional specific reserved keywords.

Very important: The C++ language is a "case sensitive" language. That means that an identifier written in capital letters is not equivalent to another one with the same name but written in small letters. Thus, for example, the `RESULT` variable is not the same as the `result` variable or the `Result` variable. These are three different identifiers identifiying three different variables.

Fundamental data types

The values of variables are stored somewhere in an unspecified location in the computer memory as zeros and ones. Our program does not

🔴If statement $ ...

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

int main()
{

int age ;

Cout<< "Enter your age:";
cin  >> age;

If (age >=100){
Cout<<"you are too old to enter";
}
Else if(age>=18){
Cout<<"welcome to the site";
}
Else if(age<=10){
Cout<<"you haven't been born yet";
}
Else{
Cout<<"you are not old enough to enter ";
}
Return 0;
}
```

Output

```
Enter your age: -10
you haven't been born yet
```

🔴switch

Alternative to using many "else if" statements compare one value against matching cases

In the case of months..we want to write a long prgrm with else ...if... statements, instead,we can use switch.

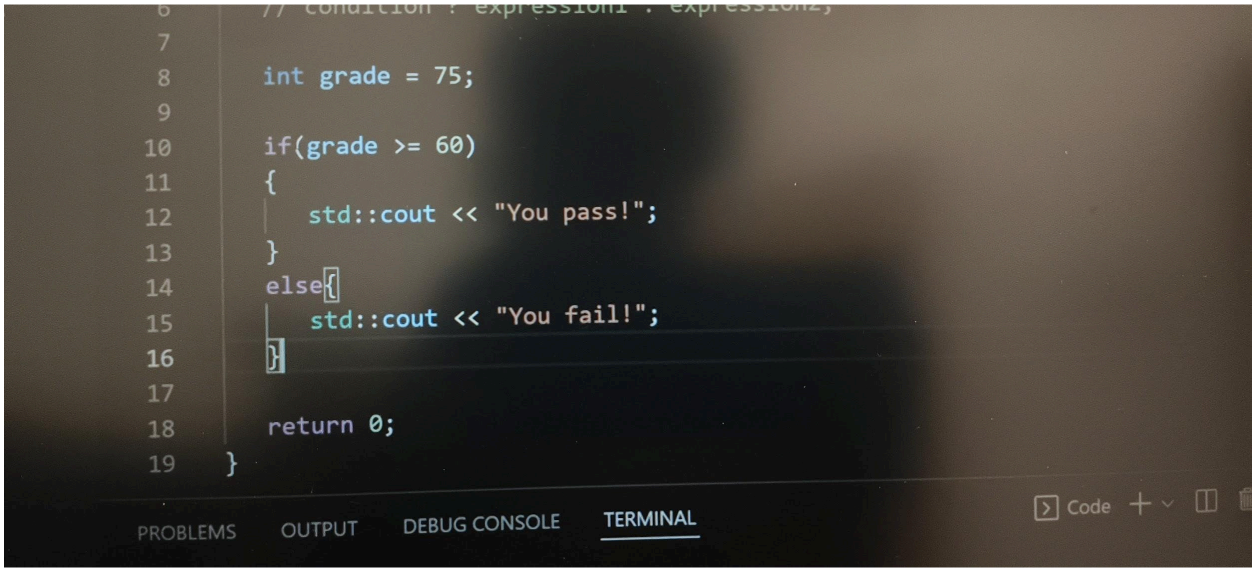
```
#include  < iostream >
using namespace std;

int main()
{

      char grade;

      cout<<"what letter grade?:";
      cin>>grade;

     switch (grade){
          case 'A':
                  cout<<"you did great";
                  Break;
          case 'B':
                  cout<<"you did good";
                  Break;
           case 'c':
                  cout<<"you did it";
                  Break;
           case 'D':
                  cout<<"bad";
                  Break;
           case 'E':
                  cout<<"you fail";
                  Break;
       Default:
                  Cout<<"please only enter a letter
grade (A-F)";
     }
Return 0;
}
```

```
Output

What letter grade?:P
Please only enter a letter grade (A-F)

C
You did it
```

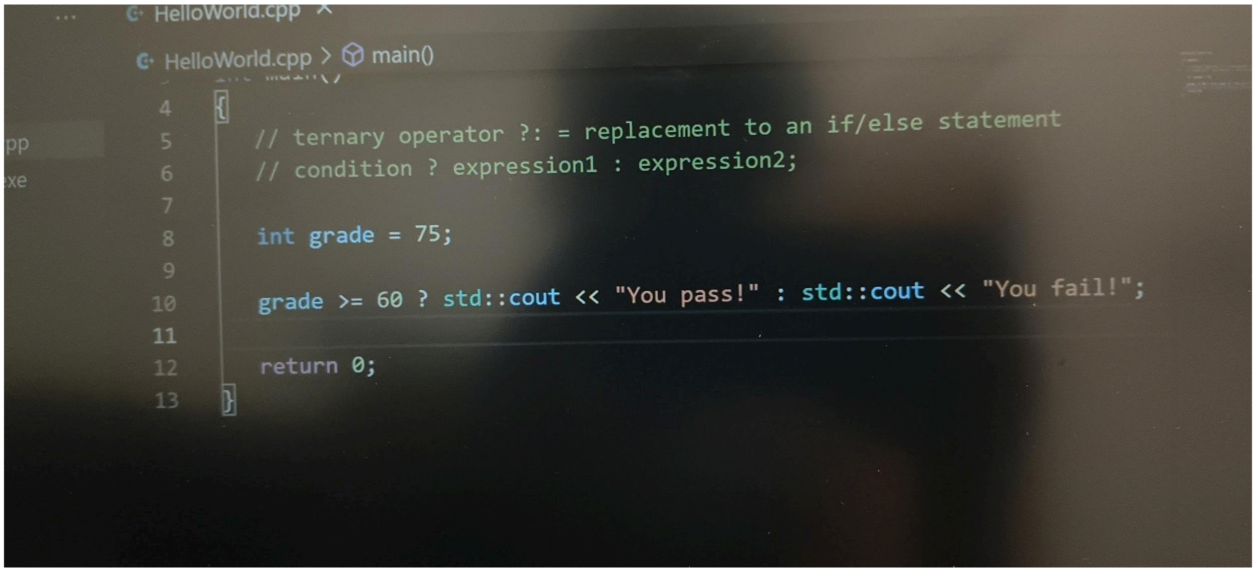
🔴🔴Ternary operator 🔴🔴

?:

replacement to if/else statement
Condition ? expression1 : expression2;



```
    // condition ? expression1 : expression2;

    int grade = 75;

    if(grade >= 60)
    {
        std::cout << "You pass!";
    }
    else{
        std::cout << "You fail!";
    }

    return 0;
}
```



```
{
    // ternary operator ?: = replacement to an if/else statement
    // condition ? expression1 : expression2;

    int grade = 75;

    grade >= 60 ? std::cout << "You pass!" : std::cout << "You fail!";

    return 0;
}
```

Here, we can see that,@2nd simply use Ternary operator instead of using if/else statements.

```
Int number  =9;
Grade % 2 ? Std::cout<<("ODD" : "EVEN");
```

```
Bool hungry = true;
Hungry ? Std::cout<<("YOU ARE HUNGRY"
: " YOU ARE NOT HUNGRY");
```

🔴🔴logical operators

&& = Check the two given conditions are true ..

```
int main()
{

  Int temp;

  Std::cout << " Enter the temperature: ";
Std::cin >> temp;

If( temp > 0 && temp < 30 ){
Std :: cout << " The temperature is good!\n";
}
Else{
  Std::cout << " The temperature is bad!\n";
}

Return 0;

}

Output
Enter the temperature: 100
The temperature is bad!
```

| | = Check if at least one of two conditions is true

Here only one will be true and the other will be false..

…

```
int main()
{

Int temp;

  Std::cout << " Enter the temperature: ";
Std::cin >> temp;

If( temp <= 0 | | temp >= 30 ){
Std :: cout << " The temperature is bad!\n";
}
Else{
  Std::cout << " The temperature is good!\n";
}

Return 0;

}

Output
Enter the temperature:-100
The temperature is bad!
```

! = If a condition is true ,it becomes false .if the condition is false,it becomes true.

```
int main()
{

  Int temp;
  Bool sunny = false;

  Std::cout << " Enter the temperature: ";
Std::cin >> temp;

If( temp <= 0 | | temp >= 30 ){
Std :: cout << " The temperature is bad!\n";
}
Else{
  Std::cout << " The temperature is good!\n";
}

If (!sunny){
Std:: cout << "It is cloudy outside!";
}
Else{
```

```
Std::cout << " It is sunny outside!";
}

Return 0;

}

Output

Enter the temperature:50
The temperature is bad!
It's is cloudy outside!
```

String methods..

- .length()

Length method means length of a string

```
#include <iostream>

Int main{

String name;
Std::cout<<"enter your name : ";
std::getline(std::cin,name);

If (name.length()  > 12){
   Std::cout << "your name can't be over 12 characters";
}
Else{
  std::cout << "welcome" << name;
}

Return 0;

}

Output
Enter your name: bhagath sen123
Your name can't be over 12 characters
…………………………………………………….

Enter your name: bhagath sen
Welcome bhagath sen
```

- .empty()

```
#include <iostream>

Int main{

String name;
Std::cout<<"enter your name : ";
std::getline(std::cin,name);

If (name.empty()){
   Std::cout << "you didn't enter your name";
}
Else{
  std::cout << "Hello" << name;
}

Return 0;

}

Output
Enter your name:
You didn't enter your name
……………………………………………….

Enter your name: bhagath sen
Hello bhagath sen
```

- .clear()

```
#include <iostream>

Int main{

String name;
Std::cout<<"enter your name : ";
std::getline(std::cin,name);

name.clear();

Std:: cout<<"Hello"<<name;


Return 0;

}
```

```
Output
Enter your name: bhagath sen
Hello
.............................................
...........
```

- .append()

```
#include <iostream>

Int main{

String name;
Std::cout<<"enter your name : ";
std::getline(std::cin,name);

name.append("@gmail.com");

Std::cout<<"your username is now "<< name;

Return 0;

}
```

Output

```
Enter your name: bhagath9037
Your username is now bhagath9037@gmail.com
…………………………………………………..
```

- .at()

```
#include <iostream>

Int main{

String name;
Std::cout<<"enter your name : ";
std::getline(std::cin,name);

Std::cout<< name.at(0);

Return 0;

}

Output
Enter your name: bhagath
b

If ……at.(2)
It will gives back a

at.(6)
h
……………………………………….
…………..
```

- . insert ()

```
#include <iostream>

Int main{

String name;
Std::cout<<"enter your name : 
";
std::getline(std::cin,name);

name.insert(1, "@");
Std::cout<<name;

Return 0;

}

Output
Enter your name: bhagath
b@hagath
…………………………………
……………..
```

- .find()

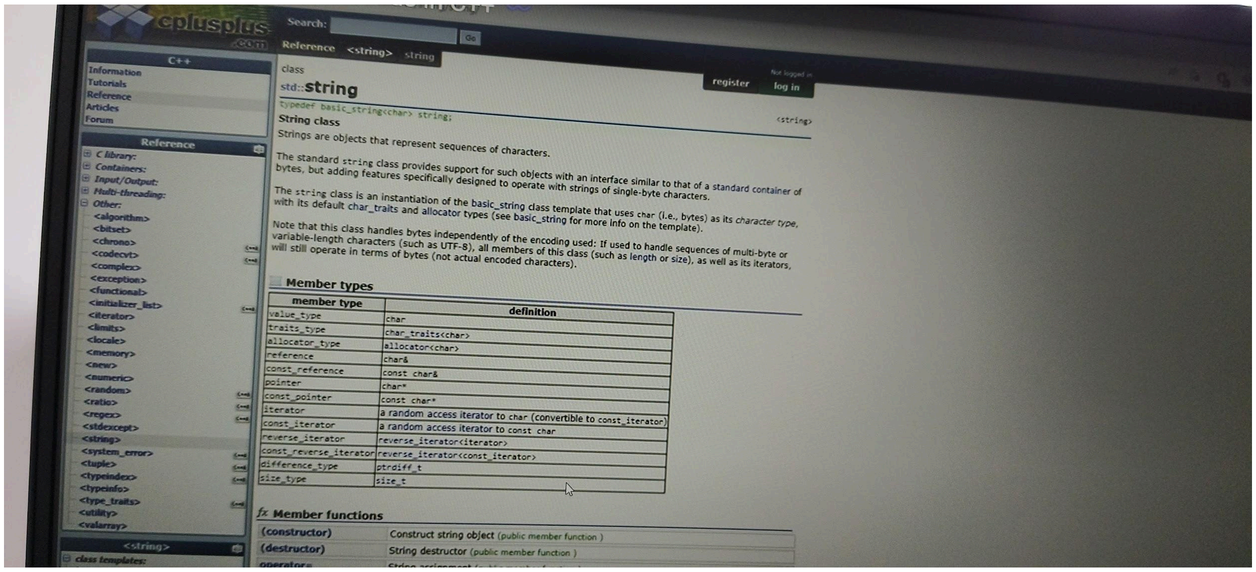
```
#include <iostream>

Int main{

String name;
Std::cout<<"enter your name : ";
std::getline(std::cin,name);

Std::cout<< name.find('  ');

Return 0;

}

Output
Enter your name: bhagath sen
7
…………………………………….
…………..
```

- .erase()

```
#include <iostream>

Int main{

String name;
Std::cout<<"enter your name : ";
std::getline(std::cin,name);

name.erase(0, 3);
Std::cout<< name;

Return 0;

}

Output
Enter your name: bhagath
gath
.........................................
..............
```

More information
!!!

!



class
std::**string**

`typedef basic_string<char> string;`

String class <string>

Strings are objects that represent sequences of characters.

The standard `string` class provides support for such objects with an interface similar to that of a standard container of bytes, but adding features specifically designed to operate with strings of single-byte characters.

The `string` class is an instantiation of the basic_string class template that uses `char` (i.e., bytes) as its *character type*, with its default char_traits and allocator types (see basic_string for more info on the template).

Note that this class handles bytes independently of the encoding used: If used to handle sequences of multi-byte or variable-length characters (such as UTF-8), all members of this class (such as length or size), as well as its iterators, will still operate in terms of bytes (not actual encoded characters).

Member types

member type	definition
value_type	char
traits_type	char_traits<char>
allocator_type	allocator<char>
reference	char&
const_reference	const char&
pointer	char*
const_pointer	const char*
iterator	a random access iterator to char (convertible to const_iterator)
const_iterator	a random access iterator to const char
reverse_iterator	reverse_iterator<iterator>
const_reverse_iterator	reverse_iterator<const_iterator>
difference_type	ptrdiff_t
size_type	size_t

fx **Member functions**

(constructor)	Construct string object (public member function)
(destructor)	String destructor (public member function)

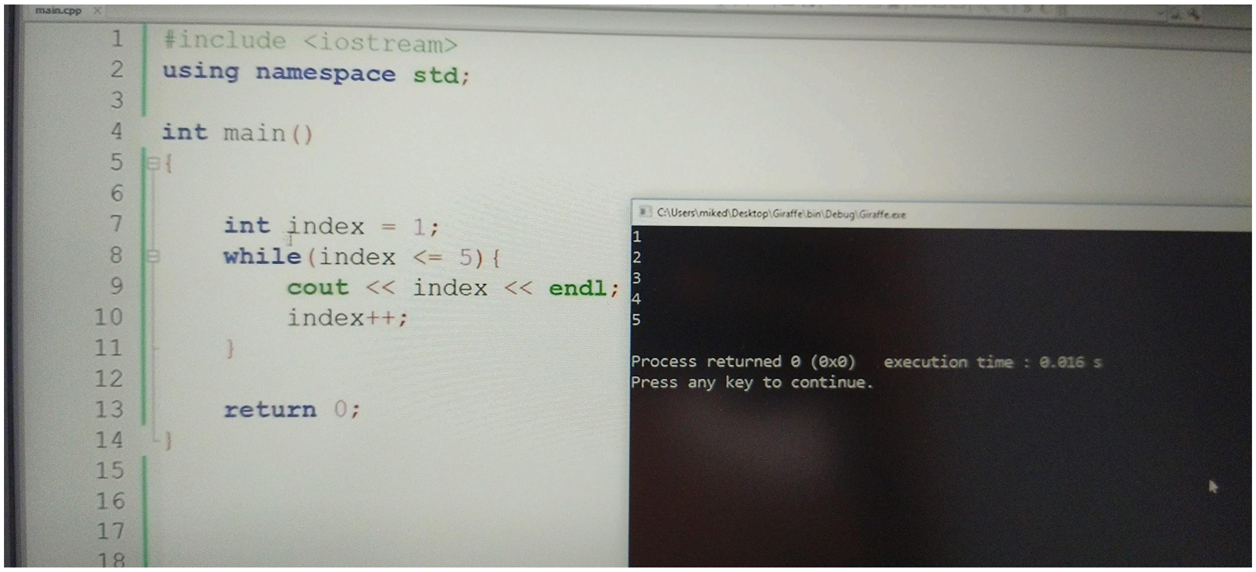
!

🔴🔴While loop

Basically speaking and while loop is a kind of an if statement accept it can repeat some code Infinitely as this condition remains true.

```
#include <iostream>

Int main{

String name;
while(name.empty()){
Std::cout<<"enter your name : ";
std::getline(std::cin,name);
}

Std::cout <<"Hello"<< name;

Return 0;

}
```

```
Output
Enter your name:
Enter your name:
Enter your name: bhagath
Hello bhagath
……………………………………
…………..
```



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

int main()
{

    int index = 1;
    while(index <= 5){
        cout << index << endl;
        index++;
    }

    return 0;
}
```

```
1
2
3
4
5

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

```
#include <iostream>

Int main{

String name;

While(1==1);
{

Std::cout<<"HELP ! I AM STUCK IN THE INFINIT LOOP: ";
}
```

Return 0;

}

Output
………….
….
……
…

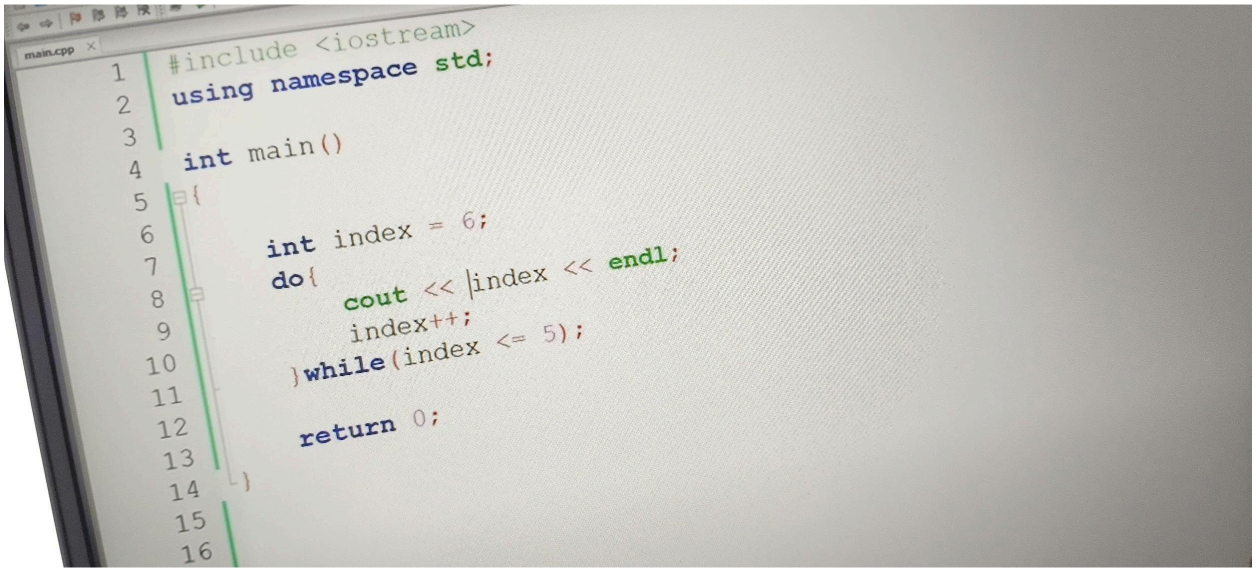
….

…..
………………………………………………….

Here the condition,will never become false and it is repeatly loop..loop...loop..
It's called infinity loop

🔴🔴Do..while..

At first,do some block of code then,repeat again,if the condition is true.



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

int main()
{
        int index = 6;
        do{
            cout << index << endl;
            index++;
        }while(index <= 5);

        return 0;
}
```

It's output will 6

Because first check code ,then condition
But in while loop, first check condition then,code

If it is while loop,output will be ………
(nothing/blank)

🔴🔴 for loops

For loop is also a loop,which will execute a block of code a specified amount of times

(Index (i)= counting sort is an algorithm for sorting a collection of objects according to keys)

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

Int main(){

For(int i =0,i <=10,i ++){
Cout<< i <<'\n';
}

Cout<<"Good morning 🌞! \n";

Return 0;
}

Output
0
1
2
3
4
5
6
7
8
9
10
Good morning 🌞!
```

for(int i =0;i<=10;i++)

🔴🔴 Break and continue keywords

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

```
Int main(){

For(int i =1,i <=20,i ++){
  If (i==13){
        Break;
}
Cout<< i <<'\n';
}

Cout<<"Good morning 🌞! \n";

Return 0;
}
```

Output

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

If we use continue;(((((skip current iteration))))) instead of Break((((break out of loop))))

Output

```
1
2
3
4
5
6
7
8
9
10
11
12
14
15
16
17
18
19
```

```
for (int i=1;i<=20;i++)
```

20
Good morning 🌞!

🔴🔴 nested loop

Nested loop is a loop that's just inside of another loop

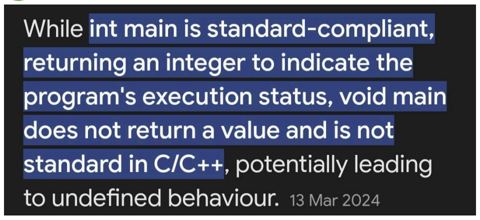
```
#include <iostream>
using namespace std;

Int main(){

Int rows;
Int columns;
Int symbol;

Cout<<"how many rows: ";
Cin>>rows;

Cout<<"how many columns: ";
Cin>>columns;

Cout<<"enter the symbol: ";
Cin>>symbol;

For (int i =1;i<= rows; i++){
    For (int j=1;j<=columns; j++){
        Cout<< symbol;
}
Cout<<'\n';
}
```

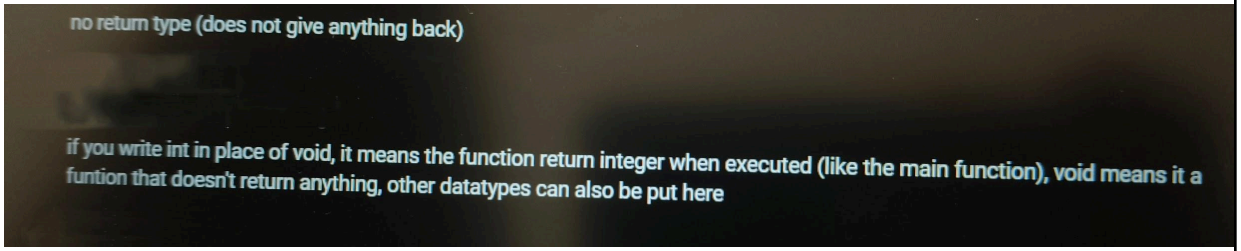
```
Output
How many rows:3
How many columns:4
Enter the symbol:#

###
###
###
###
```

🔴🔴User defined functions

C++ allows the programmer to define their own function.

🟢Void

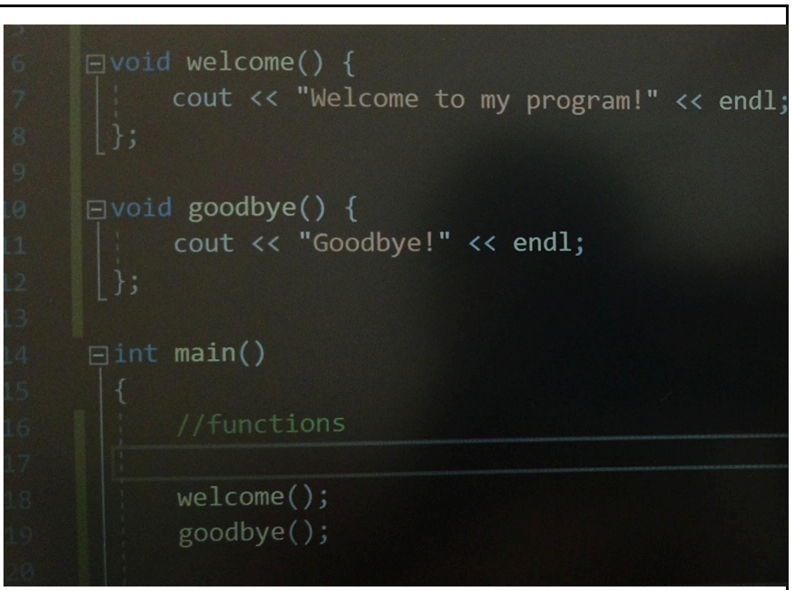

While int main is standard-compliant, returning an integer to indicate the program's execution status, void main does not return a value and is not standard in C/C++, potentially leading to undefined behaviour. 13 Mar 2024


no return type (does not give anything back)

if you write int in place of void, it means the function return integer when executed (like the main function), void means it a funtion that doesn't return anything, other datatypes can also be put here

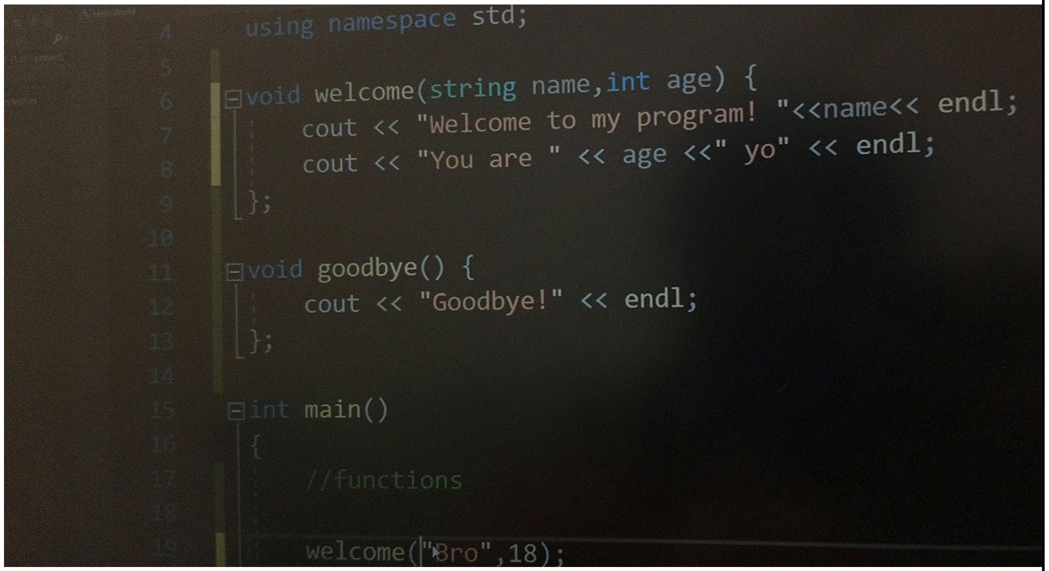
If we use your own function 🟢 void bhagath();🟢here function name start with a lowercase 'b' .

Below picture is an example,there we type a void welcome() function and goodbye();. now going back to our main method we can call this function whenever we want to perform this task and in this instance we're displaying one line code or one message so in order to call a function we can type in the name of the function followed by (); (inside the main function)



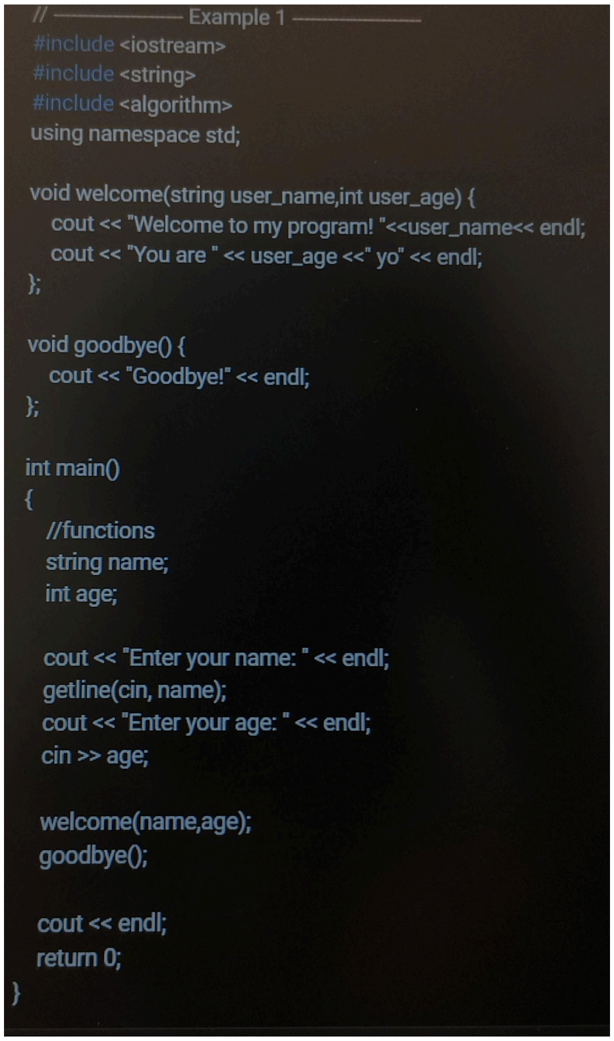
```cpp
void welcome() {
    cout << "Welcome to my program!" << endl;
};

void goodbye() {
    cout << "Goodbye!" << endl;
};

int main()
{
    //functions

    welcome();
    goodbye();
```

Argument 🟢The values that are declared within a function when the function is called.. ((((((((welcome(bro,18);((((here the two arguments bro and 18))))))))))
Parameter 🟢(string name,int age) the data type((string,int)) of this argument followed by a unique name for this value .



```cpp
using namespace std;

void welcome(string name,int age) {
    cout << "Welcome to my program! "<<name<< endl;
    cout << "You are " << age <<" yo" << endl;
};

void goodbye() {
    cout << "Goodbye!" << endl;
};

int main()
{
    //functions

    welcome("Bro",18);
```

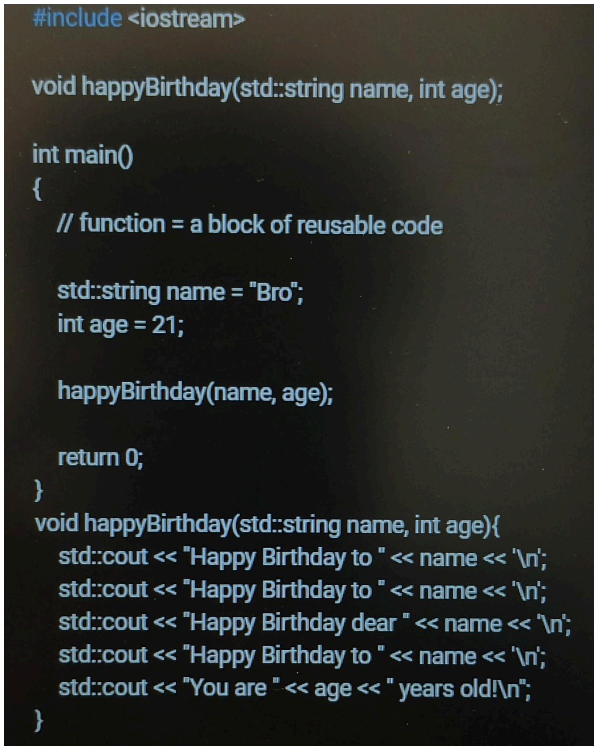
🟢Here we create a function called add(number1,number2) as arguments too.



```cpp
// ——————— Example 1 ———————
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

void welcome(string user_name,int user_age) {
    cout << "Welcome to my program! "<<user_name<< endl;
    cout << "You are " << user_age <<" yo" << endl;
};

void goodbye() {
    cout << "Goodbye!" << endl;
};

int main()
{
    //functions
    string name;
    int age;

    cout << "Enter your name: " << endl;
    getline(cin, name);
    cout << "Enter your age: " << endl;
    cin >> age;

    welcome(name,age);
    goodbye();

    cout << endl;
    return 0;
}
```

Output

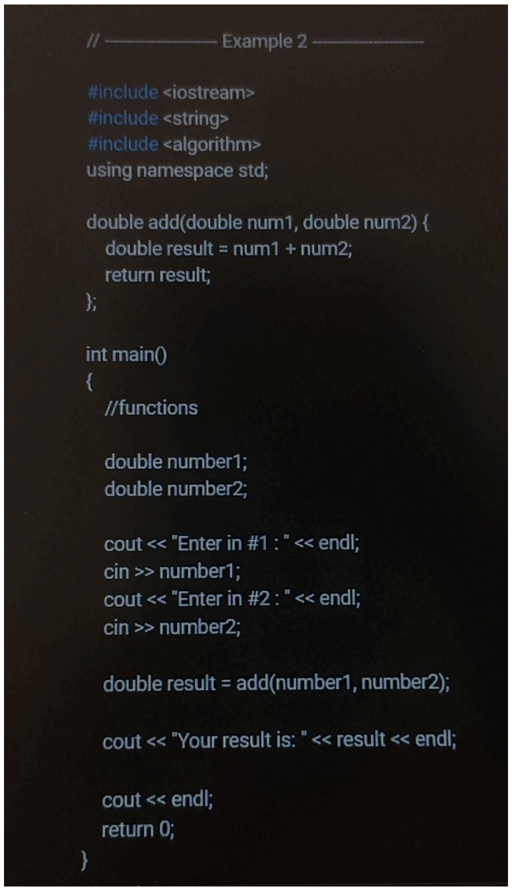
```
Enter your name: bhagath
Enter your age:18
Welcome to my program! bhagath
You are 18 yo
```



```cpp
#include <iostream>

void happyBirthday(std::string name, int age);

int main()
{
    // function = a block of reusable code

    std::string name = "Bro";
    int age = 21;

    happyBirthday(name, age);

    return 0;
}
void happyBirthday(std::string name, int age){
    std::cout << "Happy Birthday to " << name << '\n';
    std::cout << "Happy Birthday to " << name << '\n';
    std::cout << "Happy Birthday dear " << name << '\n';
    std::cout << "Happy Birthday to " << name << '\n';
    std::cout << "You are " << age << " years old!\n";
}
```

Output

```
Happy birthday to bro
Happy birthday to bro
Happy birthday dear bro
Happy birthday to bro
You are 21 years old!
```



```
// ―――――――― Example 2 ――――――――

#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

double add(double num1, double num2) {
   double result = num1 + num2;
   return result;
};

int main()
{
   //functions

   double number1;
   double number2;

   cout << "Enter in #1 : " << endl;
   cin >> number1;
   cout << "Enter in #2 : " << endl;
   cin >> number2;

   double result = add(number1, number2);

   cout << "Your result is: " << result << endl;

   cout << endl;
   return 0;
}
```

Output

```
Enter in #1 :3.4
Enter in #2 :7.1
Your  result is : 10.5
```

Here we use double instead of void, because we want a return value.
if we use void ,we can't get a return value.