



Introduction to C++

Georgia Koutsandria

Internet of Things A.Y. 19-20

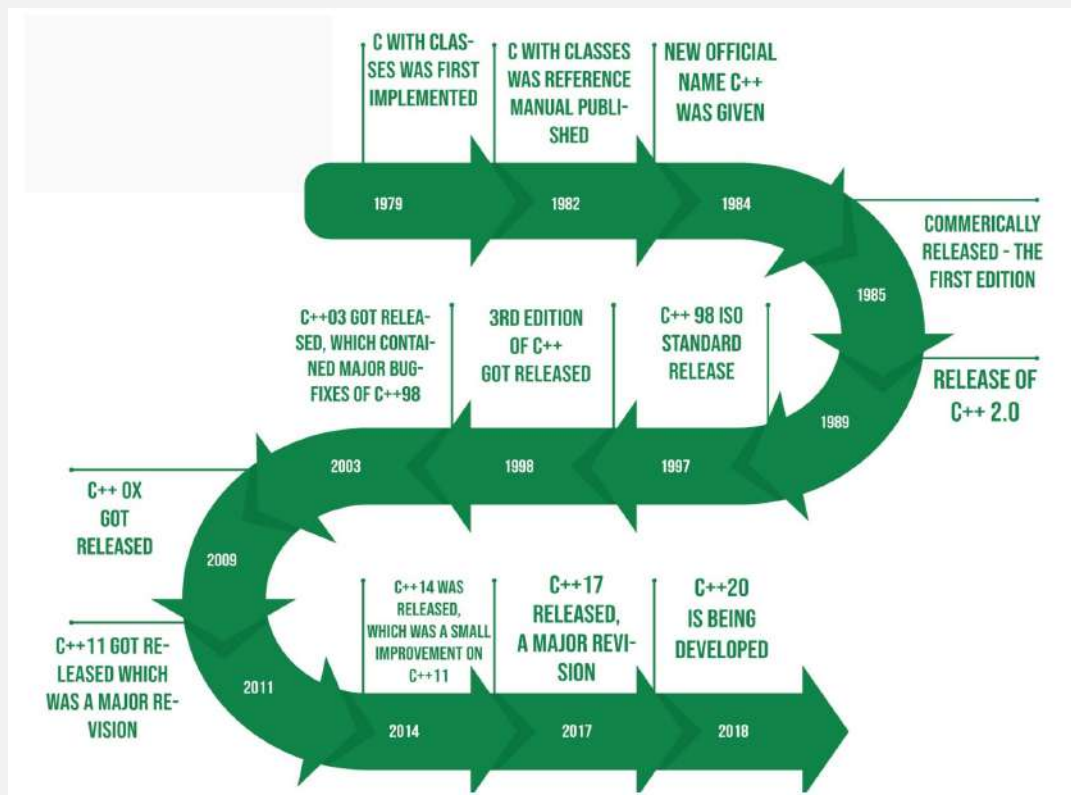
Prof. Chiara Petrioli

Dept. of Computer Science

Sapienza University of Rome

Introduction to C++

-

























What is C++?

- Middle-level programming language; It comprises both high-level and low-level language features.
- Object-oriented; Concepts: object, class, polymorphism, inheritance, abstraction, encapsulation.
- Imperative; Code implements a sequence of steps that change the state (small amount of memory) of the device/computer.
- Low-level memory manipulation; It supports dynamic memory allocation and management techniques.
- Powerful and fast.



Why C++?

Language Rank	Types	Spectrum Ranking
1. Python	  	100.0
2. C++	  	99.7
3. Java	  	97.5
4. C	  	96.7
5. C#	  	89.4
6. PHP		84.9
7. R		82.9
8. JavaScript	 	82.6
9. Go	 	76.4
10. Assembly		74.1

Introduction to C++

-
- The image displays three hexagonal icons arranged in a triangular pattern. The top-left icon is the C++ logo, featuring a blue hexagon with a white 'C' and two white '+' signs. The top-right icon shows a snippet of C++ code in a dark-themed editor, including variable declarations, a function definition, and pointer usage. The bottom-right icon is a word cloud with 'C++' prominently displayed in large yellow letters, surrounded by various programming-related terms like 'programmer', 'bug', 'development', 'algorithm', and 'Haskell'.

Why C++ in IoT?

- System/hardware-specific coding
- Embedded programming
- Resource-constrained devices, such as memory, processing power, etc..
- Large-scale systems
- ...





Compilers



What is a compiler?



- Computers understand only one language which is called *machine language*.
- This language consists of a set of instructions made of ones and zeros.
- Compilers translate high-level programming languages into machine language.



How to compile a C++ program



- **Windows:** Install an Integrated Development Interface (IDE).
 - Dev-C++ <http://www.bloodshed.net/dev/index.html>
- **Mac:** Install Xcode with the gcc/clang compilers.

```
g++ -std=c++11 example.cpp -o example_program OR  
clang++ -std=c++11 -stdlib=libc++ example.cpp -o example_program
```

- **Linux:** Compile your code directly from the terminal using the following command

```
g++ -std=c++0x example.cpp -o example_program
```





Basics of C++



Basic Syntax of C++



- Code is usually written in files with a `.cpp` extension
- Lines of comments are ignored by the compilers
 - Single line: Starts with `//` and proceeds to the end of the line
 - E.g. `//This is an example of a comment`
 - Block of lines: Starts with `/*` and proceeds to first `*/` -- do not nest
 - E.g. `/*This is an example of a comment. */`
- Each line ends with a semicolon(`;`)
- C++ code is case sensitive

INT is not the same as int is not the same as Int



Identifiers



- They used to name and identify entities such types, variables, functions, or any user-defined item.
- A sequence of one or more letters, digits, and underscore characters.
- They do not begin with a digit.
- No punctuation (@, %, etc.) is allowed within identifiers.
- Identifiers that begin with underscore(_) or contain double underscores are, usually, reserved as part of the language and should be avoided.
- Case sensitive.

```
counter    firstCounter  
myvar3     f_o_o_b_a_r  
           sqrt_2     FirstCounter  
myVar      foobar_2
```

Valid identifiers



Identifiers – Reserved keywords



<code>alignas</code> (since C++11)	<code>default</code> (1)	<code>register</code> (2)
<code>alignof</code> (since C++11)	<code>delete</code> (1)	<code>reinterpret_cast</code>
<code>and</code>	<code>do</code>	<code>requires</code> (since C++20)
<code>and_eq</code>	<code>double</code>	<code>return</code>
<code>asm</code>	<code>dynamic_cast</code>	<code>short</code>
<code>atomic_cancel</code> (TM TS)	<code>else</code>	<code>signed</code>
<code>atomic_commit</code> (TM TS)	<code>enum</code>	<code>sizeof</code> (1)
<code>atomic_noexcept</code> (TM TS)	<code>explicit</code>	<code>static</code>
<code>auto</code> (1)	<code>export</code> (1)(3)	<code>static_assert</code> (since C++11)
<code>bitand</code>	<code>extern</code> (1)	<code>static_cast</code>
<code>bitor</code>	<code>false</code>	<code>struct</code> (1)
<code>bool</code>	<code>float</code>	<code>switch</code>
<code>break</code>	<code>for</code>	<code>synchronized</code> (TM TS)
<code>case</code>	<code>friend</code>	<code>template</code>
<code>catch</code>	<code>goto</code>	<code>this</code>
<code>char</code>	<code>if</code>	<code>thread_local</code> (since C++11)
<code>char8_t</code> (since C++20)	<code>inline</code> (1)	<code>throw</code>
<code>char16_t</code> (since C++11)	<code>int</code>	<code>true</code>
<code>char32_t</code> (since C++11)	<code>long</code>	<code>try</code>
<code>class</code> (1)	<code>mutable</code> (1)	<code>typedef</code>
<code>compl</code>	<code>namespace</code>	<code>typeid</code>
<code>concept</code> (since C++20)	<code>new</code>	<code>typename</code>
<code>const</code>	<code>noexcept</code> (since C++11)	<code>union</code>
<code>constexpr</code> (since C++20)	<code>not</code>	<code>unsigned</code>
<code>constexpr</code> (since C++11)	<code>not_eq</code>	<code>using</code> (1)
<code>constinit</code> (since C++20)	<code>null_ptr</code> (since C++11)	<code>virtual</code>
<code>const_cast</code>	<code>operator</code>	<code>void</code>
<code>continue</code>	<code>or</code>	<code>volatile</code>
<code>co_await</code> (since C++20)	<code>or_eq</code>	<code>wchar_t</code>
<code>co_return</code> (since C++20)	<code>private</code>	<code>while</code>
<code>co_yield</code> (since C++20)	<code>protected</code>	<code>xor</code>
<code>decltype</code> (since C++11)	<code>public</code>	<code>xor_eq</code>
	<code>reflexpr</code> (reflection TS)	

- (1) - meaning changed or new meaning added in C++11.
- (2) - meaning changed in C++17.
- (3) - meaning changed in C++20.

Note that `and`, `bitor`, `or`, `xor`, `compl`, `bitand`, `and_eq`, `or_eq`, `xor_eq`, `not`, and `not_eq` (along with the digraphs `<%, %>`, `<:, :>`, `%, :`, and `%, %:`) provide an alternative way to represent standard tokens.

In addition to keywords, there are *identifiers with special meaning*, which may be used as names of objects or functions, but have special meaning in certain contexts.

In addition to keywords, there are *identifiers with special meaning*, which may be used as names of objects or functions, but have special meaning in certain contexts.

```
override (C++11)
final (C++11)
import (C++20)
module (C++20)
transaction_safe (TM TS)
transaction_safe_dynamic (TM TS)
```




Variables and Basic Types



What is a variable?

- A portion of memory to store a value that has a name and is of a specific type.
- **Name:** Distinguishes a variable from other variables.
- **Type:** Determines the meaning of the data and operations.



y	12.5	1001
		1002
		1003
		1004
Temperature	32	1005
Letter	'c'	1006
Number	-	1007
		1008
		1009



Fundamental Data Types



int : integer (7, 1024, ...)

bool : logical (true, false)

float : single precision real number 1.234f, -3.86f

double : double precision real number 1.234f, -3.86f

char : character variable ('a', 'b', 'k', etc..)

- Let's declare an integer variable called 'k' -> *int* k;
- Let's assign an initial value to 'k' -> *int* k = 10;



Fundamental Data Types



Type	Meaning	Min. Size
bool	boolean	NA
char	character	8 bits
short	short integer	16 bits
int	integer	16 bits
long	long integer	32 bits
float	single-precision floating-point	6 significant bits
double	double-precision floating-point	10 significant bits



Signed and Unsigned Types



- A **signed** type represents negative or positive numbers (including zero)
- An **unsigned** type represents only values greater than or equal to zero

`int`
`short` `long`

- The corresponding **unsigned** type is obtained by adding unsigned top the type
- E.x.: `unsigned long`

signed types



Declaring (initialized) variables



A simple variable definition consists of a type specifier, followed by a list of one or more variable names separated by commas, and ends with a semicolon.

```
int k = 123;  
bool flag = true;  
float dinstance = 1.238f;  
double time = 1.0;  
char character = 'b';
```

- *Always initialize your variables! Uninitialized variables have a value which is compiler dependent.*
- *Real constants are always declared as double precision. Use 'f' suffix to specify single precision.*



Literals

- Objects that represent a fixed value.
 - Integer literals
 - Floating-point literals
 - Boolean literals
 - Character literals
 - String literals



Character Literals



- A sequence of characters that are enclosed by single quotes (' ').
- They represent messages or characters.
- A char variable can store only one character; otherwise a character array should be used.



Character	Escape Sequence
Newline	\n
Horizontal tab	\t
Vertical tab	\v
Backspace	\b
Carriage return	\r
Formfeed	\f
Alert	\a
Backslash	\\
Question mark	\?
Single quotation mark	\'
Double quotation mark	\"
Octal number	\ooo
Hexadecimal number	\xhhh
Null character	\0

String Literals



- Same as character literals but string literals are enclosed by the double quote (" ").

```
string mystring;  
mystring = " This is a string ";
```



Constants



- Similar to normal variables.
- A constant variable cannot be changed after it is defined.
- Used when you want to avoid to change the value of a variable by mistake.

```
const int constantInteger = 10;
```



Type deduction: auto



- When a new variable is initialized, compilers can automatically figure out the type of a variable by the initializer.

```
int foo = 0;  
auto bar = foo; //same as int bar = foo;
```

- The type of `bar` is the type of the value used to initialize it, which is the type of `foo` (`int`).





Basic Input/Output



Streams



- C++ uses convenient abstraction to perform input and output operations in sequential media, e.g., screen, keyboard or a file.
- **Stream:** Insert or extract characters to/from.

```
#include <iostream>
```



Standard output (cout)



- Default standard output: screen
- It is used together with the insertion operator (<<)

```
// prints Output sentence on screen
cout << " Output sentence";
// prints number 2 on screen
cout << 2;
// prints the value of x on screen
cout << x;
```



Standard output (cout)



```
// prints Output sentence on screen  
cout << " Output sentence";  
// prints number 2 on screen  
cout << 2;  
// prints the value of x on screen  
cout << x;
```

When the text is enclosed in double quotes ("), the text is printed literally

The << operator inserts the data that follow it into the stream that precedes it



Standard output (cout)



- Multiple insertion operations (<<) may be chained in a single statement:

```
cout << " This " << " is " << " an " << " example. ";  
cout << " I am " << age << " years old. ";
```

- To add line breaks at the end, cout has to be instructed to do so:

```
cout << " First sentence.\n ";  
cout << " Second sentence.\nThird sentence. ";
```

OR

```
cout << " First sentence. " << endl;
```



Standard input (cin)



- Default standard input: keyboard
- It is used together with the extraction operator (>>)

Extracts from cin a value to be stored in the variable age

```
int age;  
cin >> age;
```

Declares a variable of type `int` called age

- The characters introduced using the keyboard are only transmitted to the program when the ENTER (or RETURN) key is pressed.



Structure of a program in C++



```
#include <iostream>
```

```
int main(){
```

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

```
    return 0;
```

```
}
```



Structure of a program in C++



Include files/libraries

A C++ standard library to perform I/O to screen

`main()`: A function of type `int`

```
#include <iostream>
```

```
int main(){
```

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

```
    return 0;
```

```
}
```

return or end the program

A returned '0' indicates success

Structure of a program in C++



```
#include <iostream>
```

```
int main(){
```

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

```
    return 0;
```

```
}
```

special stream object
which 'ends the line'
and flushes the buffer
(more later)

defines the text to be
printed to screen

cout is a 'stream'
which prints variables
and text to screen

std is the 'namespace' for the
standard library and contains a
wide range of functions



Structure of a program in C++



```
#include <iostream>

int main(){

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

    return 0;
}
```

semicolon to end each
statement



Exercise



1. Write a program that prints your name



Exercise 1-Solution



```
#include <iostream>
```

```
int main()  
{
```

```
    std::cout << "Georgia Koutsandria" << std::endl;  
    return 0;
```

```
}
```



String literals – An example



Stores sequences
of characters

```
// my first string  
#include <iostream>  
#include <string>  
using namespace std;
```

includes
the header
<string>

```
int main(){  
    string mystring;  
    mystring = " This is a string ";  
    cout << mystring;  
    return 0;  
}
```

initializes
string



I/O example



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

int main(){
    int i = 0;
    cout << "Please enter an integer value: ";
    cin >> i;
    cout << "The value you entered is " << i;
    cout << " and its double is " << i*2 << ".\n ";
    return 0;
}
```



Standard input and strings



- cin extraction always considers spaces (whitespaces, tabs, new-line,..) as terminating the value being extracted.
- Extracts a single word, not a phrase or an entire sentence.
- Function *getline* takes the stream(cin) as first argument, and the string variable as second.



Standard input and strings



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

int main(){
    string mystr;
    cout << "What's your name? ";
    getline (cin, mystr);
    cout << "Hello " << mystr << "!\n " ;
    return 0;
}
```



Standard input and strings



- The standard header `<sstream>` defines a type called `stringstream`.
- Covert strings to numerical values and vice versa.

```
string mystr ("1204");  
int myint;  
stringstream(mystr) >> myint;
```



Standard input and strings



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

int main(){
    string mystr;
    float price=0;
    int quantity=0;
    cout << "Enter price: ";
    getline (cin, mystr);
    stringstream(mystr) >> price;
    cout << "Enter quantity: ";
    getline (cin, mystr);
    stringstream(mystr) >> quantity;
    cout << "Total price: " << price*quantity << endl;
    return 0;}
```



Exercices



1. Write a program that prompts the user to input two integer numbers, then performs their sum, and prints result.
2. Write a program that prompts the user to input the sentence "This is my first sentence.", and prints that sentence.
3. Write a program that prompts the user to input a float to be stored as a string, converts it to float and prints it.



Exercise 1-Solution



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

int main(){
    int firstNum = 0, secondNum=0, sum=0;
    cout << "Enter the first number: ";
    cin >> firstNum;
    cout << "Enter the second number: ";
    cin >> secondNum;
    sum = firstNum + secondNum;
    cout << "This is the sum: " << sum << ".\n";
    return 0;
}
```



Exercise 2-Solution



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

int main(){
    string sentence;
    cout << "Enter a sentence: ";
    getline(cin, sentence);
    cout << "You entered: " << sentence << ".\n";
    return 0;
}
```



Exercise 3-Solution



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

int main(){
    string mystr;
    float num = 0;
    cout << "Enter a float number: ";
    getline(cin, mystr);
    stringstream(mystr) >> num;
    cout << "You entered: " << num << ".\n";
    return 0;
}
```





Operators



Operators



- The assignment operator (=)
 - `x = 100;`
- Simple arithmetic operations
 - Addition: +
 - Subtraction: -
 - Multiplication: *
 - Division: /
 - Modulo: %



Compound assignment



- They modify the current value of a variable by performing an operation on it :

(+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

expression	equivalent to..
<code>y += x;</code>	<code>y = y + x;</code>
<code>x -= 5;</code>	<code>x = x - 5;</code>
<code>x /= y;</code>	<code>x = x / y;</code>
<code>x *= y+1;</code>	<code>x = x * (y+1);</code>



Example



```
// compound assignment operators
#include <iostream>
using namespace std;

int main ()
{
    int a, b=3;
    a = b;
    a+=2;           // equivalent to a=a+2
    cout << a;
}
```


Increment and decrement



- The increase(++) and the decrease(--) operator, increase or reduce by one the value stored in a variable.
- They can be used both as a prefix and as a suffix (++x or x++).

```
x = 3;  
y = ++x; // y contains 4  
w = x++; // y contains 3  
z = --x; // z contains 2  
k = x--; // k contains 3
```



Relational and comparison operators



- They can be used to compare two expressions.
- The result of such operations is either true or false.

operator	description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to



Logical operators



- The operator `!` is used for the boolean operation NOT.
- The operator `&&` corresponds to the boolean logical operator AND.
- The operator `||` corresponds to the boolean logical operator OR.

```
!true // evaluates to false
!(6 <= 4) // evaluates to true
((5 == 5) && ( 3 > 6 )) // evaluates to false
((5 == 5) || ( 3 > 6 )) // evaluates to true
```





Statements and Flow Control



Statements



- Used for declaration, expression, conditional execution, jump statements, loops etc..
- Most statements end with a semicolon (;)
- Common errors

✗ `int k = 123 //missing semicolon`

✗ `int k = 123;; //extraneous semicolon`

✓ `int k = 123; //single semicolon`



Conditional Statements



1. *if* statement: Determines the flow of control based on a condition.

```
if (condition)  
    statement
```

2. *switch* statement: Evaluates an integral expression and chooses one of several execution paths based on the expression's value.

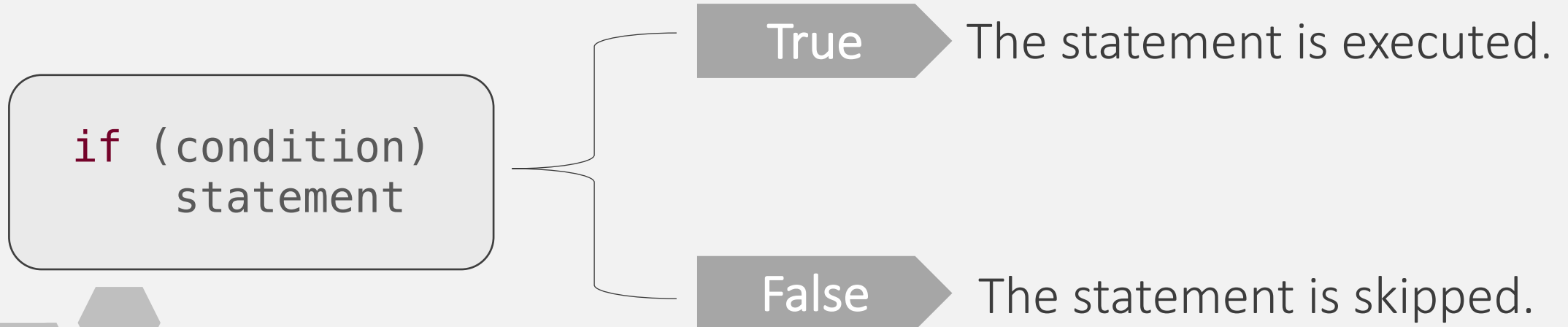
```
switch (condition)  
    statement
```



Condition(s)



- The *Condition* must be enclosed in parenthesis
- It can be an expression or an initialized variable declaration. It must have a type that is convertible to `bool`.



The *if* Conditional Statement



- Conditionally executes another statement based on whether a specified condition is true.

```
int number=0;
cout << "Enter an integer: ";
cin >> number;
// checks if the number is positive
if ( number > 0 ) {
    cout << "You entered a positive integer: " << number << endl;
}
```

simple **if**



Condition



The *if-else* Conditional Statement



```
int number=0;
cout << "Enter an integer: ";
cin >> number;
// checks if the number is positive
if ( number >= 0) {
    cout << "Positive integer: " << number << endl;
}
else{
    cout << "Negative integer: " << number << endl;
}
```



Nested *if* Conditional Statement



```
int number=0;
cout << "Enter an integer: ";
cin >> number;
// checks if the number is positive
if ( number > 0) {
    cout << "Positive integer: " << number << endl;
}
else if ( number < 0) {
    cout << "Negative integer: " << number << endl;
}
else{
    cout << "You entered 0. " << number << endl;
}
```



The *switch* Conditional Statement



- A convenient way of selecting among a (possible large) number of fixed alternatives.

```
switch(x){  
    case 1:  
        cout << "x is 1";  
        break;  
    case 2:  
        cout << "x is 2";  
        break;  
    default:  
        cout << "value of x is unknown";  
}
```



Iterative Statements (loops)



- Repeated execution until a condition is true
- Statements that test the condition before executing the block: `while`, `for`
- Statement that executes the body and then tests the condition: `do while`

```
while (condition)  
    statement
```

```
for (initializer; condition; expression)  
    statement
```

```
do  
    statement  
while (condition);
```



The for loop



- It repeats *statement* while *condition* is true.

```
#include <iostream>
using namespace std;
int main(){
    for (int n=0; n<5; n++)
        cout << n << " ";
    cout << endl;
}
```



The while loop



- It simply repeats *statement* while *condition* is true.
- The loop ends if, after any execution of *statement*, *expression* is no longer true.

```
#include <iostream>
using namespace std;
int main(){
    int n = 10;
    while (n>0){
        cout << n << ", ";
        --n;
    }
    cout << "\liftoff!\n";
}
```



The do-while loop



- It behaves like the *while* loop, except that *condition* is evaluated after the execution of the *statement*.

```
#include <iostream>
using namespace std;
int main(){
    string str;
    do {
        cout << "Enter text: ";
        getline(cin, str);
        cout << "You entered: " << str << "\n";
    } while(str!="ciao");
}
```





Jump Statements



The break statement



- It leaves a loop, even if the condition for its end is not fulfilled
- It can be used to end an infinite loop, or to force it to end before its natural end
- E.g., Let's stop the countdown before its natural end



The break statement



```
//break loop example
#include <iostream>
using namespace std;
int main(){
    for (int n=10; n>0; n--)
    {
        cout << n << ", ";
        if (n==3)
        {
            cout << "Countdown aborted!";
            break;
        }
    }
}
```



The continue statement



- It causes the program to skip the rest of the loop in the current iteration, causing it to jump to the start of the following iteration.
- E.g., Let's skip number 5 in the countdown example



The continue statement



```
//continue loop example
#include <iostream>
using namespace std;
int main(){
    for (int n=10; n>0; n--){
        if (n==5)
            continue;
        cout << n << ", ";
    }
    cout << "liftoff!\n";
}
```



The return statement



- It terminates the function that is currently executing and returns control to the point from which the function was called.
- Two forms of return statements:

```
return;  
return statement;
```



Exercises



1. Write a program that prompts the user to input three integer number and finds the greatest value among them. E.g., if input numbers are 10, 15, and 20, then the greatest number is 20. (use only if statements)



Exercise 1-Solution



```
//find the greatest number among 3
#include <iostream>
using namespace std;

int main(){
    float n1, n2, n3;
    cout << "Enter three numbers: ";
    cin >> n1 >> n2 >> n3;
    if(n1 >= n2 && n1 >= n3)
        cout << "Largest number: " << n1 << endl;
    if(n2 >= n1 && n2 >= n3)
        cout << "Largest number: " << n2 << endl;
    if(n3 >= n1 && n3 >= n2)
        cout << "Largest number: " << n3 << endl;

    return 0;
}
```



Exercises



2. Write a program that prompts the user to input three integer values and finds the greatest value among them. E.g., if input numbers are 10, 15, and 20, then the greatest value is number 20. (use if/else if/else statements)



Exercise 2-Solution



```
//find the greatest number among 3
#include <iostream>
using namespace std;

int main(){
    float n1, n2, n3;
    cout << "Enter three numbers: ";
    cin >> n1 >> n2 >> n3;
    if(n1 >= n2 && n1 >= n3)
        cout << "Largest number: " << n1 << endl;
    else if(n2 >= n1 && n2 >= n3)
        cout << "Largest number: " << n2 << endl;
    else
        cout << "Largest number: " << n3 << endl;

    return 0;
}
```



Exercises



3. Write a program to print the first 10 integer numbers (excluding zero, starting from 1 to 10).



Exercise 3-Solution



```
//print the first 10 integer numbers
//excluding 0 (from 1 to 10)
#include <iostream>
using namespace std;
int main(){
    cout << "These are the first 10 integers: ";
    for (int i=1; i <= 10; i++)
        cout << i << " ";
    cout << endl;
    return 0;
}
```



Exercises



4. Write a program that prints the squares of the numbers from 0 to 20. E.g., 0 1 4 9 16 25 36 ... 400



Exercise 4-Solution



```
//find the squares of numbers from 0 to 20
#include <iostream>
using namespace std;
int main(){
    for (int i=0; i < 21; i++)
        cout << i*i << " ";
    cout << endl;
    return 0;
}
```



Exercises



5. Write a program to find the sum of digits of a given number. E.g., if input number is 1234, then the sum is 10.



Exercise 5-Solution



```
//find the sum of digits of a given number
#include <iostream>
using namespace std;
int main(){
    int num=0, val=0, sum=0;
    cout << "Enter a number: ";
    cin >> val;
    num = val;
    while (num!=0){
        sum += num%10;
        num /= 10;
    }
    cout << "The sum of the digits of " << val << " is ";
    cout << sum << ".\n ";
    return 0;
}
```



Exercises



6. Write a program that prompts the user to enter integer numbers and prints their sum until user enters number 0. Hint: use do..while



Exercise 6-Solution



```
//enter numbers until 0 is given as input
//print the sum of them
#include <iostream>
using namespace std;
int main(){
    int num=0, sum=0;
    do{
        cout << "Enter a number: ";
        cin >> num;
        sum += num;
    }while (num!=0);
    cout << "The sum of the numbers is " << sum;
    cout << ".\n ";
    return 0;
}
```





Functions



Functions Basics



- A *function* is a block of code with a name.
- It is executed by calling the given name, and it can be called from some point of the program.
- Common syntax:

```
type name(parameter1, parameter2, ...){statements}
```



Functions Basics-An example



- Determine the factorial of a given number. The factorial of a number n is the product of the numbers from 1 through n . The factorial of 5, for example, is 120:

$$1 * 2 * 3 * 4 * 5 = 120$$

```
int fact(int val)
{
    int ret = 1;
    while (val > 1)
        ret *= val--;
    return ret;
}
```



Functions Basics



returns an **int**
value

Function named **fact**

takes one **int**
parameter

```
int fact(int val)
{
    int ret = 1; //local variable to hold the result
    while (val > 1)
        ret *= val--; //ret=ret*val and decrement val
    return ret; //return the result
}
```

postfix
decrement
operator (--)

Ends execution of **fact** and
returns the value of ret



Calling a Function



- A *function* call
 - Initializes the parameters from the arguments
 - Transfers control to that *function*.
- Execution of the called *function* begins.

```
int main(){  
    int j = fact(5);  
    cout << "5! is " << j << endl;  
    return 0;  
}
```



Calling a Function



- To call *fact*, we must supply an *int* value.
- The result of the call is also an *int*.

```
int main(){  
    int j = fact(5);  
    cout << "5! is " << j << endl;  
    return 0;  
}
```



Example



```
//function example
#include <iostream>
using namespace std;
int addition(int a, int b){
    int r;
    r=a+b;
    return r;
}
int main(){
    int z;
    z = addition(5,3);
    cout << "The result is " << z << ".\n ";
}
```



Functions with no type



- When a function does not need to return a value, the type to be used is `void`.
- This is a special type to represent the absence of value.
- The `void` can also be used in the function's parameter list to specify that the function takes no actual parameters when called.

`printmessage();`

Note the use of the empty pair of parentheses!



Passing arguments by value



- Arguments can be passed by value

```
int x=5, y=3, z;  
z = addition (x,y);
```

- Only copies of the variables values at that moment are passed to the function.
- Modifications on the values of the variables have not effect on the values of the variables outside the function.



Passing arguments by reference



- Access an external variable from within a function.
- The variable itself is passed to the function.
- Any modification on the local variables within the function are reflected in the variables passed as arguments in the call.
- References are indicated with an ampersand (&) following the parameter type.



Passing arguments by reference



```
//passing parameters by reference
#include <iostream>
using namespace std;
void duplicate(int& a, int& b){
    a*=2;
    b*=3;
}
int main(){
    int x=1, y=3;
    duplicate(x, y);
    cout << "x=" << x << ", y=" << y << "\n";
    return 0;
}
```



Declaring functions



- Functions cannot be called before they are declared.
- Functions should be declared before calling `main`.
- If the `main` is defined before an undeclared function is called, then the compilation of the program will fail.

```
int fact(int a, int b);  
void even(int x);
```

Function
declaration



Recursivity



- The property that functions have to be called by themselves.
- It can be useful for specific tasks such as sorting, factorial etc..
- E.g.: Calculate the factorial of number 3 ($3! = 1 * 2 * 3 = 6$) using a recursive function.



Recursivity



```
//factorial
#include <iostream>
using namespace std;
long factorial(long a){
    if (a > 1)
        return (a * factorial (a-1));
    else
        return 1;
}
int main(){
    long number = 3;
    long fact;
    fact = factorial(number);
    cout << number << "! = " << fact << "\n ";
}
```



Exercise 1



Write a program with a function that swaps (exchanges the values of) two integer numbers. You should pass the arguments to the function by reference. Display the values of the two numbers before and after swapping them.



Exercise 1-Solution



```
#include <iostream>
using namespace std;
void swap(int& n1, int& n2) {
    int temp;
    temp = n1;
    n1 = n2;
    n2 = temp;
}
int main(){
    int a = 10, b = 20;
    cout << "Before swapping: ";
    cout << "a = " << a << ", b = " << b << endl;
    swap(a, b);
    cout << "After swapping: ";
    cout << "a = " << a << ", b = " << b << endl;
    return 0;
}
```



Exercise 2



Write a recursive function that computes the sum of the first 10 integer numbers ($1+2+\dots+10 = 55$)



Exercise 2-Solution



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

int sum(int n){
    if (n == 0)
        return n;
    else
        return n+sum(n-1);
}

int main(){
    int n = 10;
    cout << "Sum of first 10 numbers: " << sum(n) << endl;
    return 0;
}
```



Exercise 3



Write a function that takes a single integer argument (height) and displays a pyramid of this height made up of "*" characters on the screen. E.g.: when height=6 the following should be displayed on the screen.

Enter the height of the pyramid: 6

```
      *
    * * *
  * * * * *
* * * * * * *
* * * * * * * * *
* * * * * * * * * *
```



Exercise 3-Solution



```
#include <iostream>
using namespace std;
void pyramid(int height){
    int space = 0;
    for(int i = 1, k = 0; i <= height; ++i, k = 0){
        for(space = 1; space <= height-i; ++space)
            cout << " ";
        while(k != 2*i-1){
            cout << "* ";
            ++k;}
        cout << endl;}}
int main(){
    int height = 0;
    cout <<"Enter the height of the pyramid: ";
    cin >> height;
    pyramid(height);
    return 0;
}
```





Arrays



Why do we need arrays?



```
#include <iostream>
```

```
int main(){
```

```
    int a;
```

```
    int b;
```

```
    int c;
```

```
    return 0;
```

```
}
```

Your program needs
3 int variables

```
#include <iostream>
```

```
int main(){
```

```
    int a;
```

```
    int b;
```

```
    . . .
```

```
    int z;
```

```
    return 0;
```

```
}
```

Your program needs
1000 int variables!

Arrays



- A structure which stores many variables of the same type, e.g., `int`, `double`, `bool`, etc..
- Uses an 'index' to access each variable or 'element' of the array.
- C++ includes static arrays, allocated arrays, and standard library containers.



Array declaration



- Arrays are declared like normal variables.
- Square brackets [] indicate the number of variables which the array can store.

```
#include <iostream>
```

```
int main()  
{
```

```
    int array[5];
```

```
    double array_d[2]={1.0,2.0};
```

```
    return 0;
```

```
}
```

array can store five
int variables

array_d can store 2
double variables,
initialized

Accessing array elements



- Array 'index' starts from zero
- Can be used for arithmetic, copied to other variables etc..

```
#include <iostream>
```

```
int main()  
{
```

```
    int array[5];
```

```
    array[0] = 3;
```

```
    array[1] = array[0]+5;
```

```
    return 0;
```

```
}
```

array can store five
int variables

set the first
element to 3

set the second
element to 8

Common memory errors



```
#include <iostream>
```

```
int main()  
{
```

```
    int array[5];
```

```
    array[0] = 3;
```

```
    array[5] = 5;
```

```
    return 0;
```

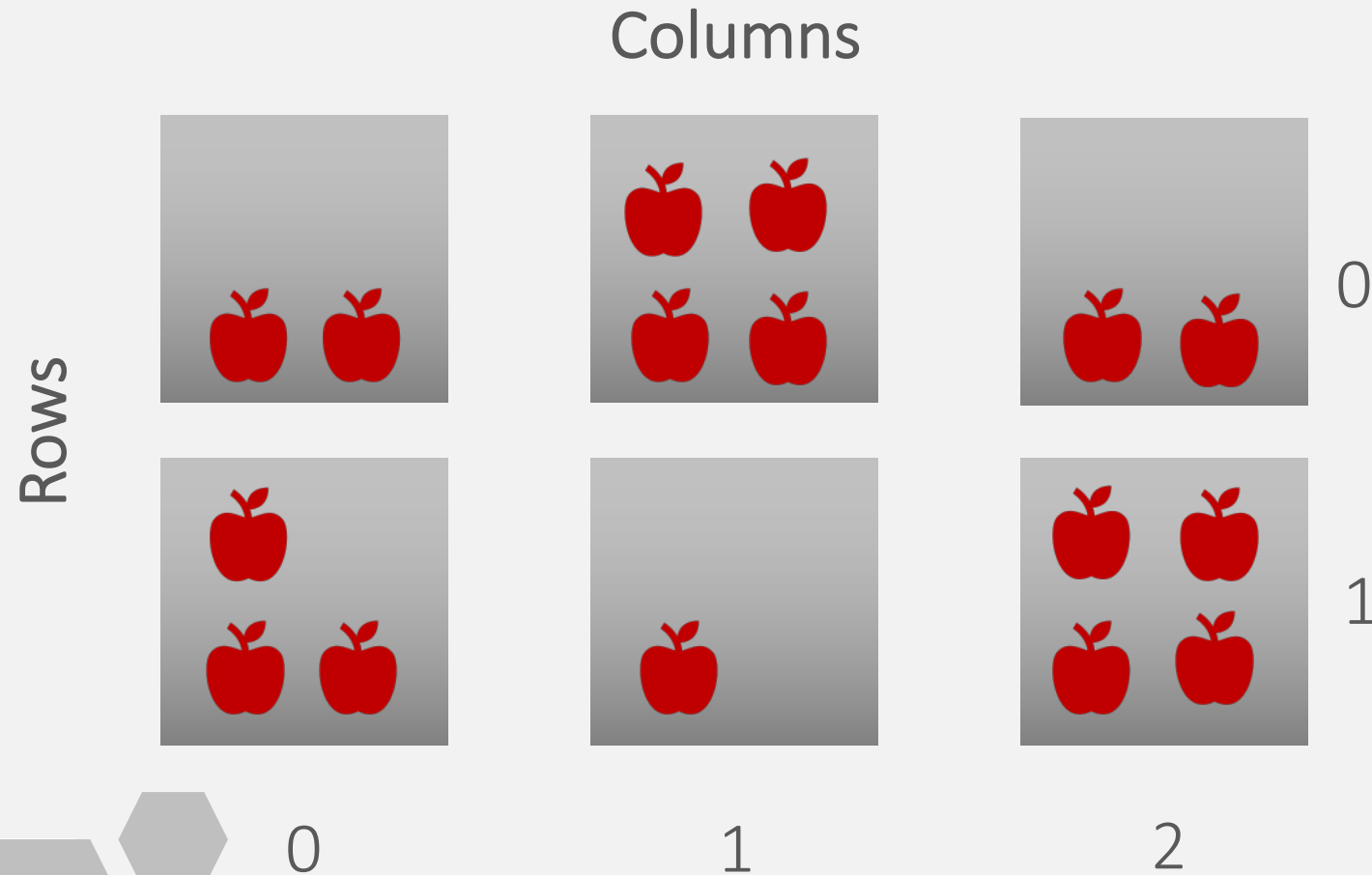
```
}
```

array can store five
int variables

set the first
element to 3. OK

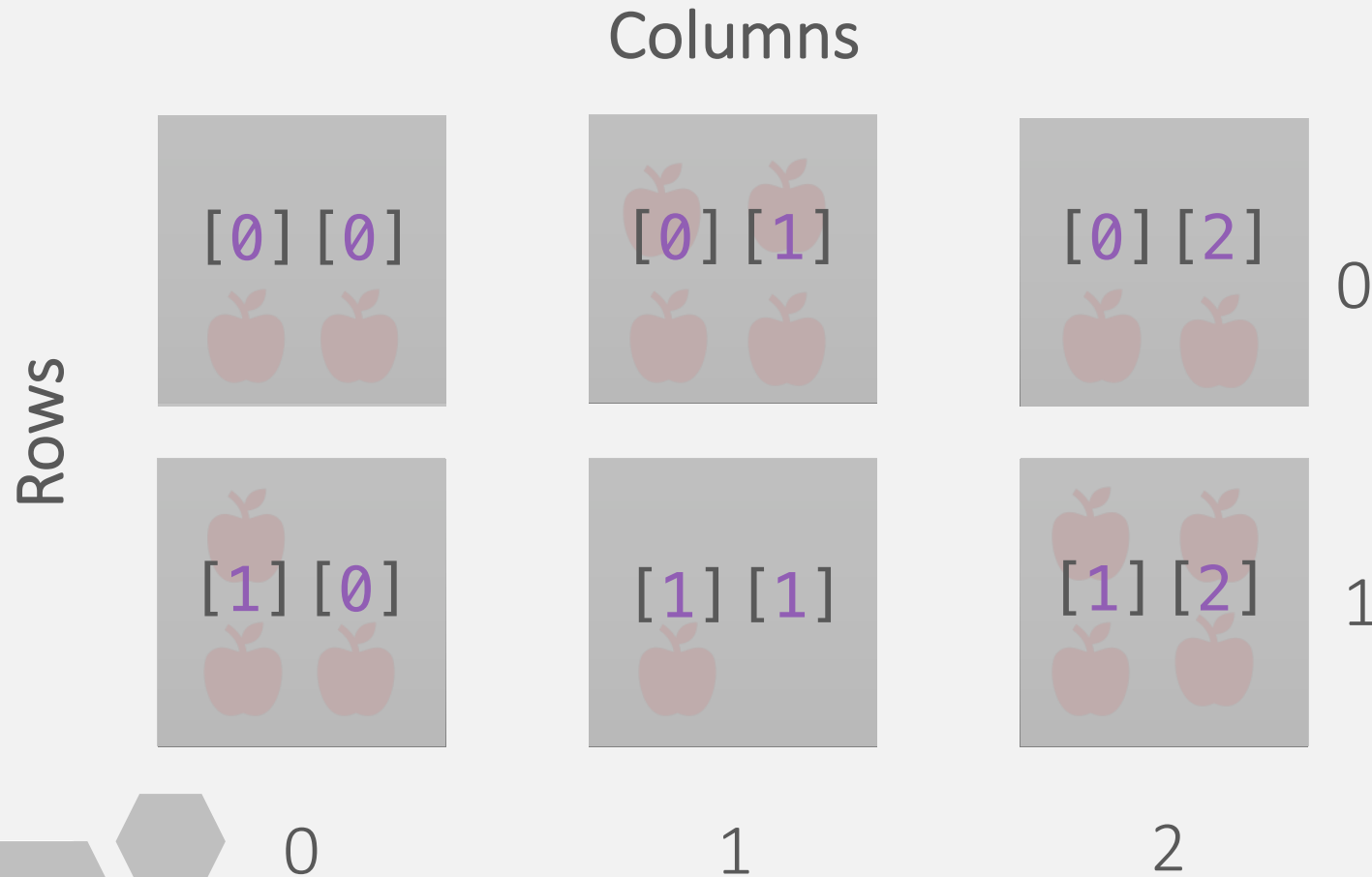
set the fifth
element to 5. Oops!

Multidimensional Arrays



```
int array[2][3];
```


Multidimensional Arrays



```
int array[2][3];
```

```
array[0][0] = 2;  
array[0][1] = 4;  
array[0][2] = 2;  
array[1][0] = 3;  
array[1][1] = 1;  
array[1][2] = 4;
```


Accessing Arrays using loops



```
#include <iostream>
```

```
int main()  
{
```

```
    int array[5];
```

```
    for(int i=0; i<5; ++i)  
        array[i] = 0;
```

```
    for(int i=0; i<5; ++i)  
        array[i] += i;
```

```
    return 0;
```

```
}
```

loop over all
elements and set
to zero

Add *i* to each
element of the
array

Accessing 2D Arrays using loops



```
#include <iostream>
```

```
int main()  
{
```

```
    int array[2][2];
```

array 2x2

```
    for(int i=0; i<2; ++i){  
        for(int j=0; j<2; ++j){  
            array[i][j] = 0;  
        }  
    }
```

Nested loop over
all elements, set
them to zero

```
    return 0;
```

```
}
```



Arrays and functions



- Arrays can be passed to a function as an argument.
- Only the name of the array is used as argument.
- When passing an array to a function, you need to mention the name of the array during the function call. Syntax:

```
function_name(array_name);
```



Exercise 1



Write a program that stores and calculates the sum of 3 integer numbers entered by the user (use arrays).



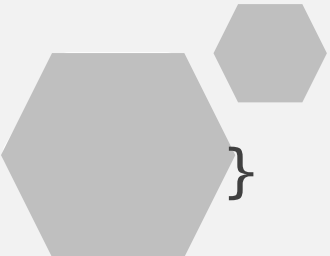
Exercise 1-Solution



```
#include <iostream>
using namespace std;
int main(){
    int num[3], sum = 0;
    cout << "Enter 3 numbers: " << endl;

    for(int i=0; i< 3; ++i){
        cin >> num[i];
        sum += num[i];
    }
    cout << "Sum is: " << sum << endl;

    return 0;
}
```



Exercise 2



Write a program that finds the smallest and largest element of a given array. E.g., if $a[5]=\{6, 13, 8, 25, 100\}$ then smallest number is $a[0]=6$ and largest number is $a[4]=100$.



Exercise 2-Solution



```
#include <iostream>
using namespace std;
int main(){
    int num[5]={6, 13, 8, 25, 100};
    int largest = 0, smallest = 100 ;

    for(int i=0; i< 5; ++i){
        if(num[i]>largest)
            largest = num[i];
        if(num[i]<smallest)
            smallest = num[i];
    }
    cout << "Smallest number is: " << smallest << endl;
    cout << "Largest number is: " << largest << endl;

    return 0;
}
```


Exercise 3



Write a program to reverse the elements of an array. Print the elements of the array before and after reversing them.



Exercise 3-Solution



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

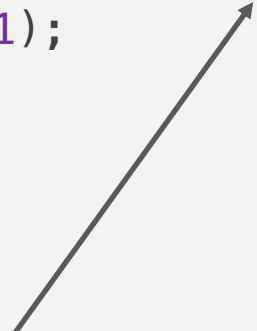
```
void rvereseArray(int arr[], int start, int end) {
    if (start >= end)
        return;
```

```
    int temp = arr[start];
    arr[start] = arr[end];
    arr[end] = temp;
    rvereseArray(arr, start + 1, end - 1);
}
```

```
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}
```

```
int main() {
    int arr[] = {1, 2, 3, 4, 5};

    cout << "Original array is: " << endl;
    printArray(arr, 5);
    rvereseArray(arr, 0, 4);
    cout << "Reversed array is: " << endl;
    printArray(arr, 5);
    return 0;
}
```



Exercise 4



Write a program to implement the Tic Tac Toe game.

Rules:

- The game requires two players and it evolves on a 3x3 grid.
- Each player chooses to fill a square in the grid with a symbol assigned to him (X or O).
- In order to win, the players need to align 3 identical symbols vertically, horizontally, or diagonally.
- If no one wins, then the game is said to be draw.

0	0	X
0	x	x
X	0	0



Exercise 4



Write a program to implement the Tic Tac Toe game.

Hints:

- Assign to a variable player the value 1 or 2 based on whether the number is even or odd. E.g.: `player=(player%2)?1:2;`

This expression translates to :

 If (player%2)

 player = 1;

 else

 player = 2;

- Assign symbols based on whether the number of the player is even or odd.

E.g.: `mark = (player==1) ? 'X' : 'O' ;`

0	0	X
0	x	x
X	0	0



Exercise 4-Solution



```
1 #include <iostream>
2 using namespace std;
3 char square[10] = {'0','1','2','3','4','5','6','7','8','9'};
4 int checkwin();
5 void board();
6 int main(){
7     int player = 1,i,choice;
8     char mark;
9     do{
10         board();
11         player=(player%2)?1:2;
12         cout << "Player " << player << ", enter a number: ";
13         cin >> choice;
14         mark=(player == 1) ? 'X' : '0';
15         if (choice == 1 && square[1] == '1')
16             square[1] = mark;
17         else if (choice == 2 && square[2] == '2')
18             square[2] = mark;
19         else if (choice == 3 && square[3] == '3')
20             square[3] = mark;
21         else if (choice == 4 && square[4] == '4')
22             square[4] = mark;
23         else if (choice == 5 && square[5] == '5')
24             square[5] = mark;
25         else if (choice == 6 && square[6] == '6')
26             square[6] = mark;
27         else if (choice == 7 && square[7] == '7')
28             square[7] = mark;
29         else if (choice == 8 && square[8] == '8')
30             square[8] = mark;
31         else if (choice == 9 && square[9] == '9')
32             square[9] = mark;
33         else{
34             cout<<"Invalid move ";
35             player--;
36         }
37         i=checkwin();
38         player++;
39     }while(i==1);
40     board();
41     if(i==1)
42         cout<<"==>\aPlayer " <<--player<<" win ";
43     else
44         cout<<"==>\aGame tie!";
45     return 0;
46 }
```


Exercise 4-Solution(Cont.)



```
47  /*****
48  FUNCTION TO RETURN GAME STATUS
49  1 FOR GAME IS OVER WITH RESULT
50  -1 FOR GAME IS IN PROGRESS
51  0 GAME IS OVER AND NO RESULT
52  *****/
53
54  int checkwin()
55  {
56      if ((square[1] == square[2] && square[2] == square[3]) || (square[4] == square[5] && square[5] == square[6])
57          || (square[7] == square[8] && square[8] == square[9]) || (square[1] == square[4] && square[4] == square[7])
58          || (square[2] == square[5] && square[5] == square[8]) || (square[3] == square[6] && square[6] == square[9])
59          || (square[1] == square[5] && square[5] == square[9]) || (square[3] == square[5] && square[5] == square[7]))
60          return 1;
61      else if (square[1] != '1' && square[2] != '2' && square[3] != '3'
62              && square[4] != '4' && square[5] != '5' && square[6] != '6'
63              && square[7] != '7' && square[8] != '8' && square[9] != '9')
64          return 0;
65      else
66          return -1;
67  }
68  /*****
69  FUNCTION TO DRAW BOARD OF TIC TAC TOE WITH PLAYERS MARK
70  *****/
71  void board(){
72      cout << "Player 1 (X) - Player 2 (O)" << endl;
73      cout << " | | " << endl;
74      cout << " " << square[1] << " | " << square[2] << " | " << square[3] << endl;
75      cout << "——|——|——" << endl;
76      cout << " " << square[4] << " | " << square[5] << " | " << square[6] << endl;
77      cout << "——|——|——" << endl;
78      cout << " " << square[7] << " | " << square[8] << " | " << square[9] << endl;
79      cout << "——|——|——" << endl;
80      cout << " " << square[7] << " | " << square[8] << " | " << square[9] << endl;
81      cout << " | | " << endl << endl;
82  }
```


Additional Resources



- <http://www.cplusplus.com/doc/tutorial/>
- <https://en.cppreference.com/w/>
- Programming: Principles and Practice Using C++, Bjarne Stroustrup (Updated for C++11/C++14)
- C++ Primer, Stanley Lippman, Josée Lajoie, and Barbara E. Moo (Updated for C++11)

