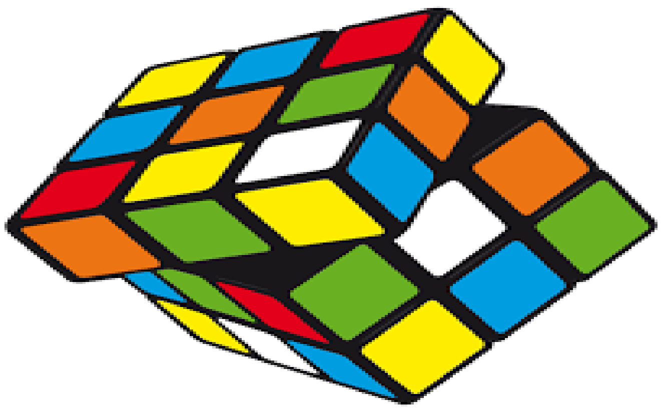
# Week #1

## | Introduction to problem solving

Okay let's take a step backwards, what is a **problem**? it's a difficulty that must be resolved or dealt with.
Maybe you have other definitions for a "problem", but let's all agree it's something challenging that we're trying to overcome and ***solve.*** That leads us to the title "what is problem solving?"



**Problem solving** is the process of working through details of a problem to reach a solution.
It may include mathematical or systematic operations, and there are four stages involved in this process.

### So what are the stages of problem solving?

- **Define the problem**
  Identify the issue that you're dealing with. Observe the problem area closely to form a detailed image of what's wrong.

- **Generate alternative solutions/ Brainstorm**
  This is one of the most important stages of problem solving. Think of all possible alternatives, be creative and think logically.

- **Evaluate and choose the best strategy**
  Consider all your options, try to choose an alternative that will solve the problem without causing other unanticipated problems (bugs).

- **Implement and follow up on the solution**
  This is where you draw up an action plan and follow through with your chosen approach. It requires planning and execution.

## | What are X-CPCs?

CPC stands for collegiate programming contest, where competitive programmers gather and compete to solve a given number of problems.
The teams are ranked on how many problems they solve and how fast they do that.

### There are four tiers to it:

1- **Local contest** (Between contestants of the same university)
2- **ECPC** (Egypt)
3- **ACPC** (Arab Region)
4- **ICPC** (International)

# | Introduction to Programming

## -What is computer programming?

Computer programming is a way of giving computers instructions about what they should do next. These instructions are known as code, and computer programmers write code to solve problems or perform a task.

The end goal is to create something: that could mean anything from a web page, or a piece of software, or even just a pretty picture. That’s why computer programming is often described as a mix between art and science; it’s technical and analytical, yet creative simultaneously.

## -What are computer programming languages?

Computer Programming Languages are the tools we use to communicate with a computer(allowing us to give instructions to a computer in a language a computer understands).

# | Introduction to competitive programming

Competitive programming is an intellectual sport, you compete by solving algorithmic/programming problems, within a limited amount of time in a ***contest***.

**A programming contest** is a competition where you have a set of problems, and a time limit, and you try to solve ***as many*** problems as you can, ***as fast*** as you can .

## Why competitive programming ?

- **You will learn :**
  - Many useful algorithms / mathematical insights.
  - How to code / debug* quickly and accurately.
  - How to work in a team (there are team contests as well).

**__debug:__ detecting bugs and problems within your code that makes it behave unexpectedly.*

- **Then you can rock in classes, job interviews, etc.**

- **It's fun!**

Okay, there's one more thing you need to know: ***online judges*** .

**An online judge** is an online system to test programs in *programming contests*, they're also used to practice for such contests.
So an online judge compiles and runs your code, then shows you a verdict: is your code **Accepted** or **not** .

## | Programming And Problem Solving :

Programming is the process of taking an algorithm and encoding it into a notation, a programming language, so it can be executed by a machine. Without an algorithm there can be no program. Programming is often the way that we create a representation for our solutions.

Algorithms describe the solution to a problem in terms of the data needed to represent the problem instance and the set of steps necessary to produce the intended result. Programming languages must provide a notational way to represent both the process and the data. To this end, languages provide control constructs and data types.

# | Variables

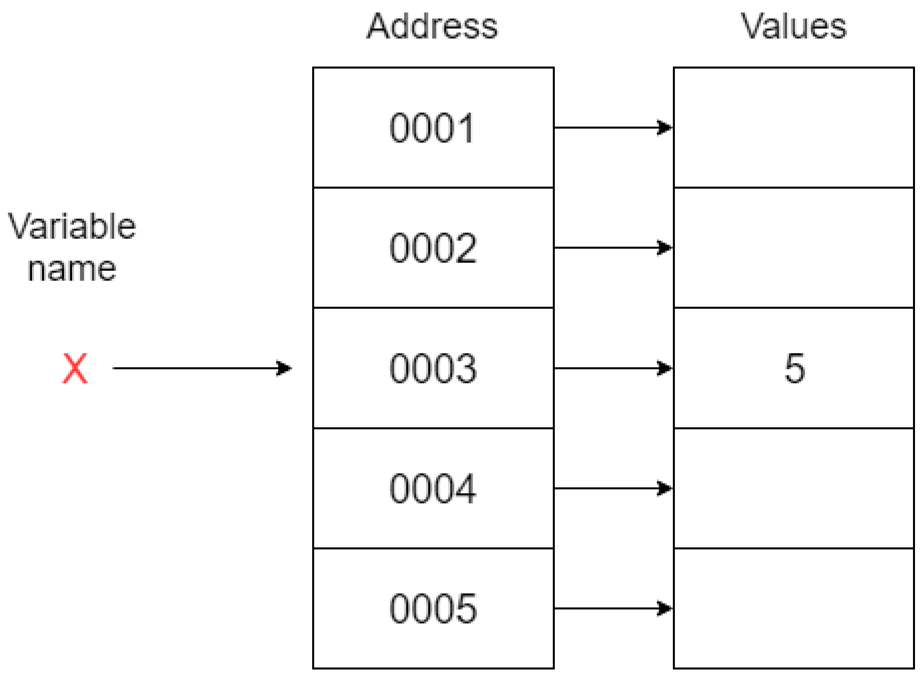
## Memory

Every computer has a very fast memory (RAM) which can be accessed by programs. Whenever you create a program, a portion of the RAM is reserved for your program to use.
The memory works much like an inventory where each storage unit has a unique address.

## What are variables?

Take a look at this statement `x = 5`

It means:
take the value of 5 and put it in `x`, But what exactly is `x` ? It's a variable.
More specifically, it's a place in the memory with a name of `x`, that holds an integer of value `5`.



## Variable types

Variables come in different types and sizes, we'll discuss the integer variable for now. An integer is a whole number written without its fractional component.

*For example: 3, 0, -50 are integers, while 0.6, 1.536 are not.*

## Variable declarations and initializations, assignment operator `=`

To create a variable in c++, we do a process called **Variable Declaration**

```
//Data_Type  Variable_Name;
  int x;
```

We define a variable `x` that holds values of type `int`
In other words, The variable x is given a location in the memory, and whenever the program sees `x`, it knows it should look in that location to get the value of `x`.

You can also declare multiple variables of the same type in a single line

```
int x, y, z;
```

Now we want to put some value in that variable, this is called **Variable Assignment**

```
//Variable_Name =  Value;
  x = 5;
```

We can even combine the two processes in one called **Variable Initialization**

```
 //Data_Type   Variable_name = initial value;
  int x = 5; (This is called copy initialization)
  int x(5); (This is called direct initialization)
  int x{5}; (This is called uniform initialization)
```

**Note:** copy initialization is the most common way

## L-Values and R-Values

look at these statements:

```
int x;
x = 4;
```

`x` is called an L-value (left side of the assignment operator) and `4` is called R-value (right side of the assignment operator)
It means: put the R-values inside the L-value. And by now we should know that to put a value in something, it must be a defined variable.

**SO ALL L-VALUES MUST BE DEFINED VARIABLES.**

These statements will not work:

```
int x, y;

//5 isn't a defined variable, we can't store any value in it
5 = x;
5 = 4 + 1;

//L-values must evaluate to defined variables, we can't assign any
value to "x+y" because it simply doesn't exist in the memory
x + y = 4;
```

but these will:

```
int x, y = 4;
//puts the value of y (4) + 2 inside the variable x
x = y + 2;
//x now equals 6

//puts the value of x(6) + the value of x(6) inside the variable y
y = x + x;
//y now equals 12
```

### Uninitialized variables (Garbage!)

When a variable is declared, it’s given what was previously stored in its memory location.
It can be something like `5277592` or `4309726`. You can never know.

```
int x;
cout << x; //who knows what we’ll get!
```

**Note:** you should always try to avoid uninitialized variables.

### Variable types (again)

In addition to integers, there are a few more data types that come in handy.

*chars are data types representing characters like ‘a’, ‘Z’ , ‘#’ , ‘@’ or even a white space ‘ ‘.
But each of these have a number according to the ASCII table ranging from [0,127]

### Naming convention

There are some rules and conventions you need to follow when naming your variables:

1- You can’t use C++ reserved keywords (this, const, friend, case..etc).
2- Your variable names can only start with a letter/ underscore.

```
int 1stNum;  //invalid
```

3- Your variable names can't contain spaces or special characters.

```
int my variable name;  //invalid
int my@variable#name;  //invalid
```

4- Use a consistent style to name your variables (camelCase, separate_by_underscores)

```
int my_variable_name;  //correct
int myVariableName;  //correct
int Myvariable_name;  //valid but not prefered
```

## | Cin

Cin is the  “standard input stream”, it’s used to read the text or numbers entered by the user from the keyboard on the console.
It is defined in <iostream> header file.
It’s used along with this operator “ >> “ , the extraction operator .
The general syntax is shown in the following example.

```
int x; // variable of type integer
cin >> x; //
```

this line reads the value of x that the user enters
This means if the user entered 5 for example , then x now has a value = 5 .
Also, the extraction operator can be used more than once to accept multiple inputs and can be used with any data type as:

```
char x, y, z ; //variables of type character
cin >> x >> y >> z ; //the first character the user enters will be
x , the second one y and so on
```

*But take care, the input stops by a space or a new line, which means:
If for example you have a variable x of type int and you want x to have a value = 123
But then while you enter this value you typed : 12 -> space/enter -> 3
Your input stops at the space/enter , so x now has a value = 12 **not** 123.

## | Cout

The predefined object cout is an instance of ostream class. The `cout` object is said to be "connected to" the standard output device, which usually is the display screen. The `cout` is used in conjunction with the stream insertion operator, written as “ << “ which are two less than signs as shown in the following example.
It is defined in <iostream> header file.

**Example:**

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

int main() {
   cout << "Hello acmASCIS"<< endl;
}
```

the insertion operator << may be used more than once in a single statement.

```
cout << "Hello acmASCIS" << endl;
```

can be written as
`cout << "Hello" << " " << "acmASCIS" << endl;`
and endl is used to add a new-line after the last thing written on the console.

## Operators

You can use different arithmetic operators with variables

`-` : Subtraction.
`+` : Addition.
`*` : Multiplication
`/` : Division.
`%` : Modulo operator (finding the remainder after division).

```
int x = 5;
x = x + 2;         //puts the value of x (5) + 2 inside x
                   //X equals to 7
int y = 4;
y = x * y;         //puts the value of x(7)*value of y(4) inside y
                   // y equals 28

y = y / 2;        // puts the value of y(28) / 2 inside y
                  // y equals 14
int t = 3;
int rem = t%2;   // puts the remainder of t/2 in rem
                 // rem = 1
```

## Operators precedence

Just like they do in math, arithmetic operators follow a standard precedence
There are many operators in C++, but for now you need to know that
`*` , `/` and `%` operators have **higher** precedence than `+` and `-`

For example:

```
int x = 5;
cout << x * 5 + x / 5;
//prints 26
```

## Compound assignment operators

```
int x = 5;
x -= 2;          //equivalent to x = x - 2
                 //X equals to 3
int y = 4;
y *= x;          //equivalent to y = y*x
                 // y equals 12
```

```
y /= 2;          //equivalent to y = y/2
                 // y equals 6
```

### Increment/ decrement operators

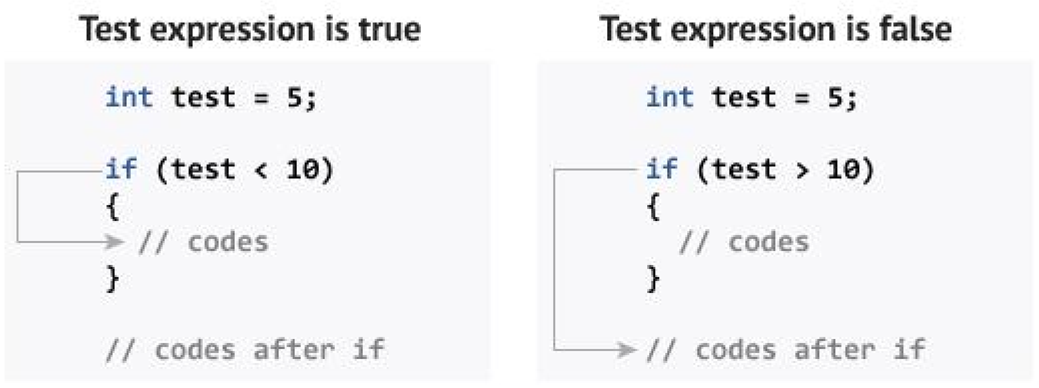
```
int x = 5;
cout << x++;        //means print the value of x (5) to the console
then increment it
                    //X equals to 6, the console prints 5
int y = 4;
cout << --y;        //means decrement the value of y(4) then print
it to the console      // y equals to 3, the console prints 3
```

There are other types of operators called “comparison operators”
You’ll get to know them in the if/ else statements section.

## | If Condition

The if keyword is used to execute a statement or block of code, if and only if, a condition is fulfilled.

```
if ( TRUE )
{
  // Execute all statements inside the braces
}
```



```
int test = 5;

if (test < 10)
{
  // codes
}

// codes after if
```

```
int test = 5;

if (test > 10)
{
  // codes
}

// codes after if
```

* in the first example :`int test = 5`,  so think of the if statement that way, is `test < 10` ?

**Yes**, `test = 5 < 10`, **this expression is true**, so the piece of code within the braces will be executed.

In the second example, ask the same question : is `test > 10` ?

**No**, `test = 5` and 5 is not > 10, **this expression is false**, so the code within the braces will be skipped, and the code after it will be executed normally.

## Relational Operators:

Relational Operators are used to compare between values to set a condition.
Here are the relational operators, as they are known, along with examples:

```
>     greater than               5 > 4 is TRUE
<     less than                  4 < 5 is TRUE
>=    greater than or equal      4 >= 4 is TRUE
<=    less than or equal         3 <= 4 is TRUE
```

## Logical operators:

We use logical operators to obtain a single relational result out of one or more expressions.

### “!” - NOT operator: Inverts the expression to its right.

```
!(4 == 5) //4 == 5 evaluates to false, then !(false) evaluates to true.
!(5 == 5) //5 == 5 evaluates to true, then !(true) evaluates to false.
```

### “&&” - AND operator: evaluates to true ONLY when both expressions are true.

```
(4 == 4) && (5 == 5) //evaluates to true, because both expressions are
true
(4 == 5) && (5 == 5) //evaluates to false, because the first expression
is false
(4 == 5) && (5 == 6) //evaluates to false, because both expressions are
false
```

### “||” - OR operator: evaluates to true when ANY expression is true.

```
(4 == 4) || (5 == 5) //evaluates to true, because both expressions are
true
(4 == 5) || (5 == 5) //evaluates to true, because the second expression
is true
(4 == 5) || (5 == 6) //evaluates to false, because the both expressions
are false
```

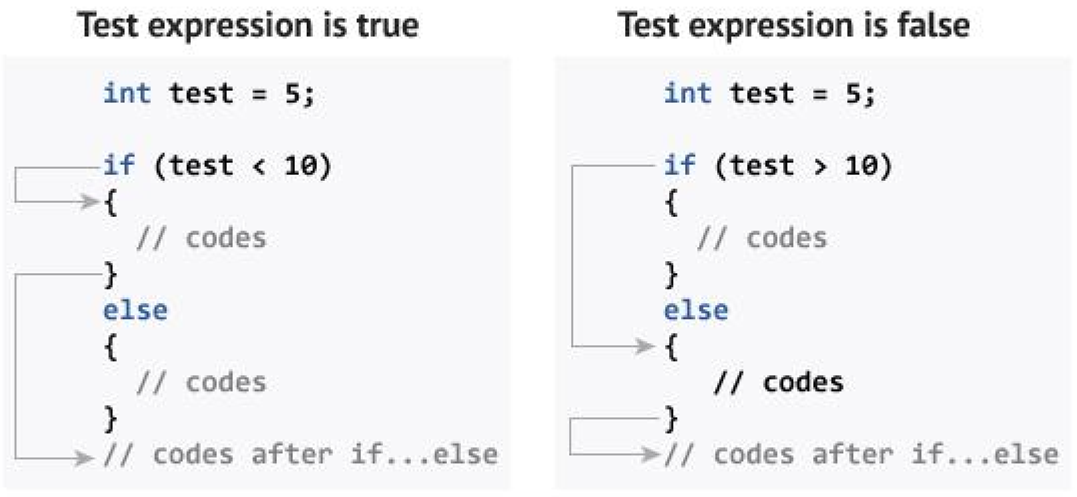
## if / else

Sometimes when the condition in an if statement evaluates to false, it would be nice to execute some code instead of the code executed when the statement evaluates to true. The "else" statement effectively says whatever code after it.

**Example :**

```
if ( condition ) // Execute these statements if TRUE
{

}
else // Execute these statements if FALSE
{
}
```



Test expression is true

```
int test = 5;

if (test < 10)
{
  // codes
}
else
{
  // codes
}
// codes after if...else
```

Test expression is false

```
int test = 5;

if (test > 10)
{
  // codes
}
else
{
   // codes
}
// codes after if...else
```

**Let’s consider this example :**

```
int test = 20 ;
if ( test  <= 10 )
{
     cout << "the condition was true" << endl ;
}
else
{
    cout << "the condition was false" << endl ;
}
cout << "done" ;
```

```
Output : the condition was false
         done
```

* But if the value of `test` was `10`,

```
the output will be : the condition was true
                     done
```

The code after the if/else statement will be executed anyways, so “done” will be displayed  in both cases.

## else  if

Another use of “else” is when there are multiple conditional statements that may all evaluate to true, yet you want only one if-statement's body to execute. You can use an "else if" statement following an if-statement and its body; that way, if the first statement is true, the "else if" will be ignored, but if the if-statement is false, it will then check the condition for the “else if” statement. If the if-statement was true the else-statement will not be checked. It is possible to use many “else if” statements to ensure that only one block of code is executed.

**Example :**

```
if ( testExpression1 )
{
  // statements to be executed if testExpression1 is true
}
else if( testExpression2 )
{
 // statements to be executed if testExpression1 is       false
and testExpression2 is true
}
else if ( testExpression3 )
{
 // statements to be executed if testExpression1 and
testExpression2 is false and testExpression3 is true
}
else
{
   // statements to be executed if all test expressions are false
}
```

## Example On Comparing using Conditions

What is The Output of the following Code ?

```
int age = 105;
if ( age < 100 )           // If the age is less than 100
  {
     cout << "You are pretty young! \n";
  }
  else if ( age == 100 )
  {
     cout << "You are old\n" ; // Executed if the first condition
is false and the second condition is true
  }
  else
  {
      cout<<"You are really old\n";  // Executed if no other
statement is true
 }
```

**This reference belongs to ACMASCIS  ( Ain shams University )**