Computer Programming

Overview

Prof. Amit Agarwal

amitfce@iitr.ac.in amit.agarwal@mfs.iitr.ac.in

slido

What do you think of programming (one word)?

ⓘ Start presenting to display the poll results on this slide.

Introduction

- Computer: (Definition by Oxford Dictionary)

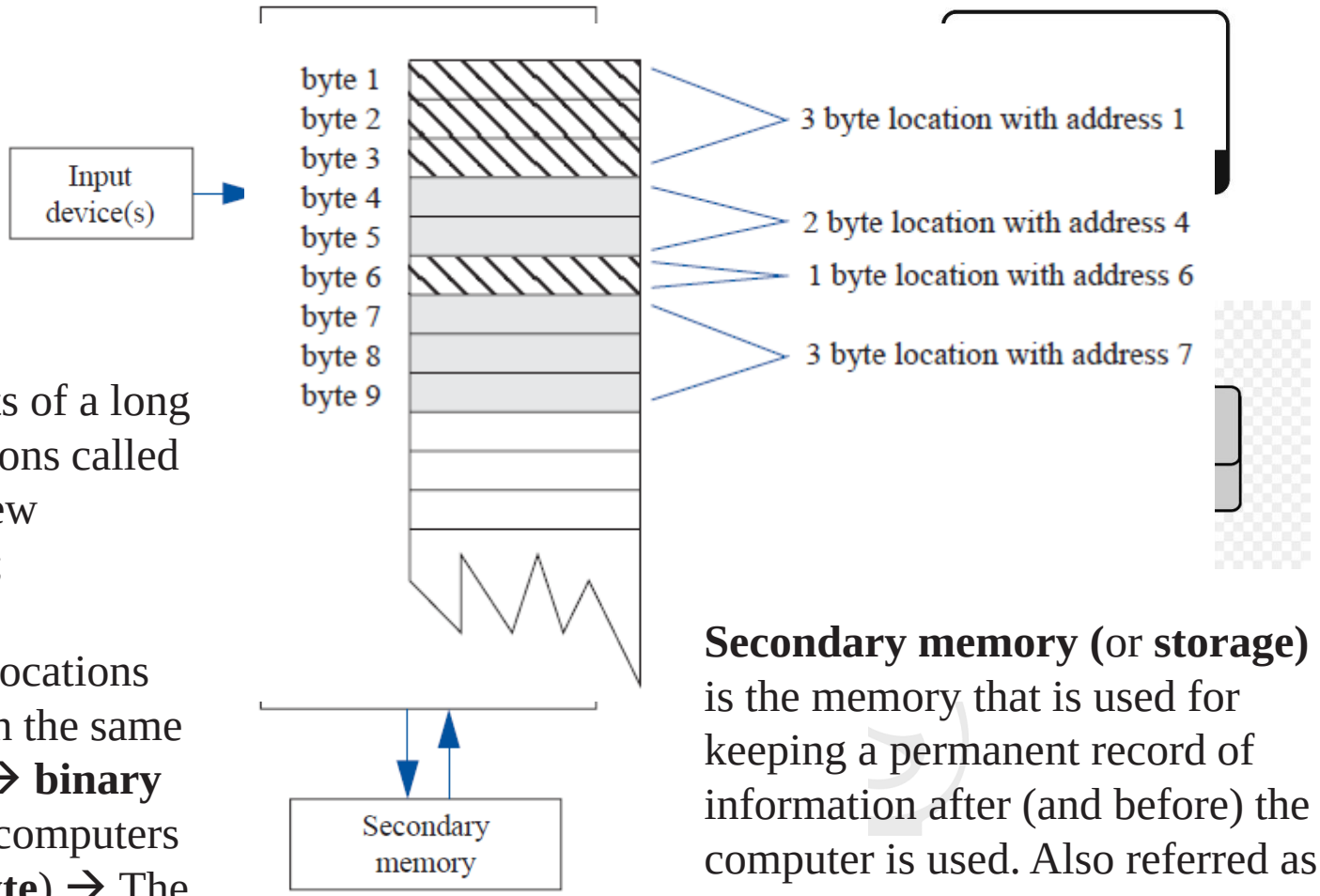
"an electronic **device** for storing and processing **data**, …, according to **instructions given** to it in a **variable program.**"

- Complex system consisting of hardware and software
  - Hardware: physical parts (e.g., input media, motherboard, etc.)
  - Software: a set of programs used by a computer

- Programming is a process of writing instructions (a set of commands) to solve a specific problem.

- Algorithm is a step-by-step problem-solving process.

Basics components

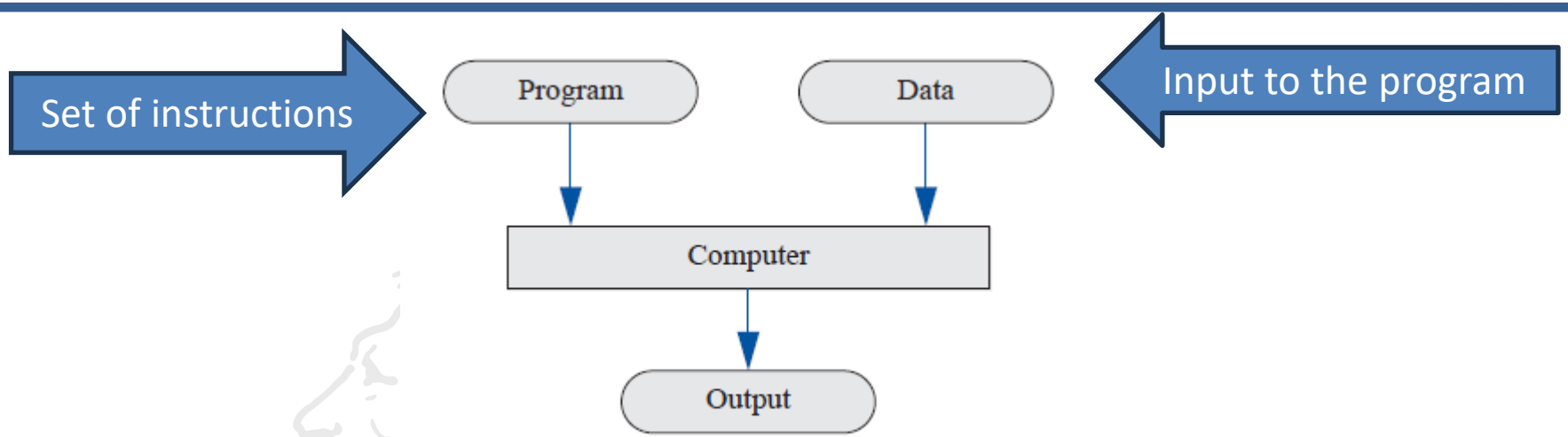


Main memory consists of a long list of numbered locations called *memory locations* (a few thousands to millions);

The contents of these locations can change and contain the same number of *0s* and *1s* → **binary digit** or a **bit** ➔ most computers contains 8 bits (i.e., **byte**) → The number that identifies a byte is called its **address.**

Secondary memory (or **storage)** is the memory that is used for keeping a permanent record of information after (and before) the computer is used. Also referred as *auxiliary memory*, *auxiliary storage*, *external memory*, and *external storage*

Programs



- Algorithm
  - A set of **explicit and unambiguous finite steps**, which, when carried out for a given set of **initial conditions** to produce the corresponding **output** and terminate in **finite time**.
  - Basically, they are **a set of steps** to be performed, which can be used to perform a task/solve a problem.
- Program
  - An implementation of an algorithm in a programming language that computer can understand.

Programs

Algorithm

```
Input A, B, C
Sum = A + B + C
Print Sum
```

Program (in C++)

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

int main() {
    int A, B, C, Sum;
    cin>>A>>B>>C;
    Sum = A + B + C;
    cout<<Sum;
    return 0;
}
```

High and low level languages

High-level

- Resemble human languages
- Are designed to be easy to read and write
- Use more complicated instructions than the CPU can follow
- Must be translated to zeros and ones for the CPU to execute a program
- C, C++, Java, Visual Basic, etc.

- → Source Code

Low-level

- It is difficult to write and debug code in Low-Level Languages.
- Assembly

  ADD X Y Z

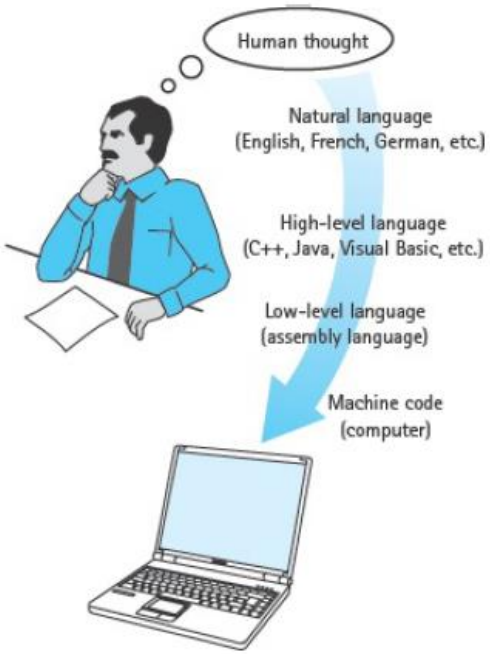
- Machine language
  - CPU can follow machine language
  - Assembly must be converted to machine language

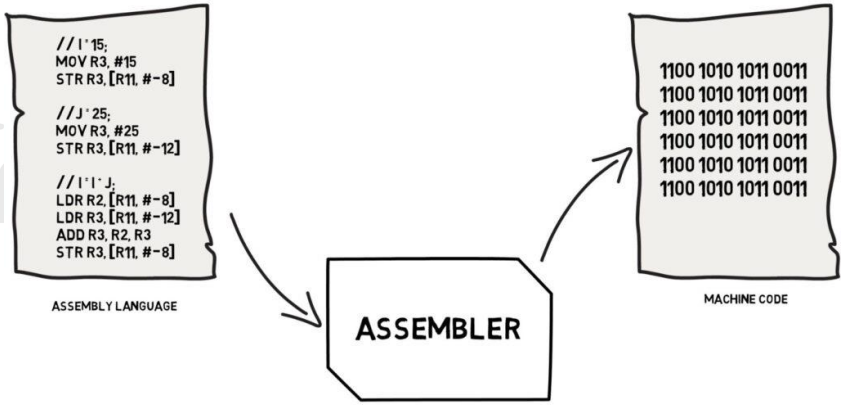
0110 1001 1010 1011

→ Object Code

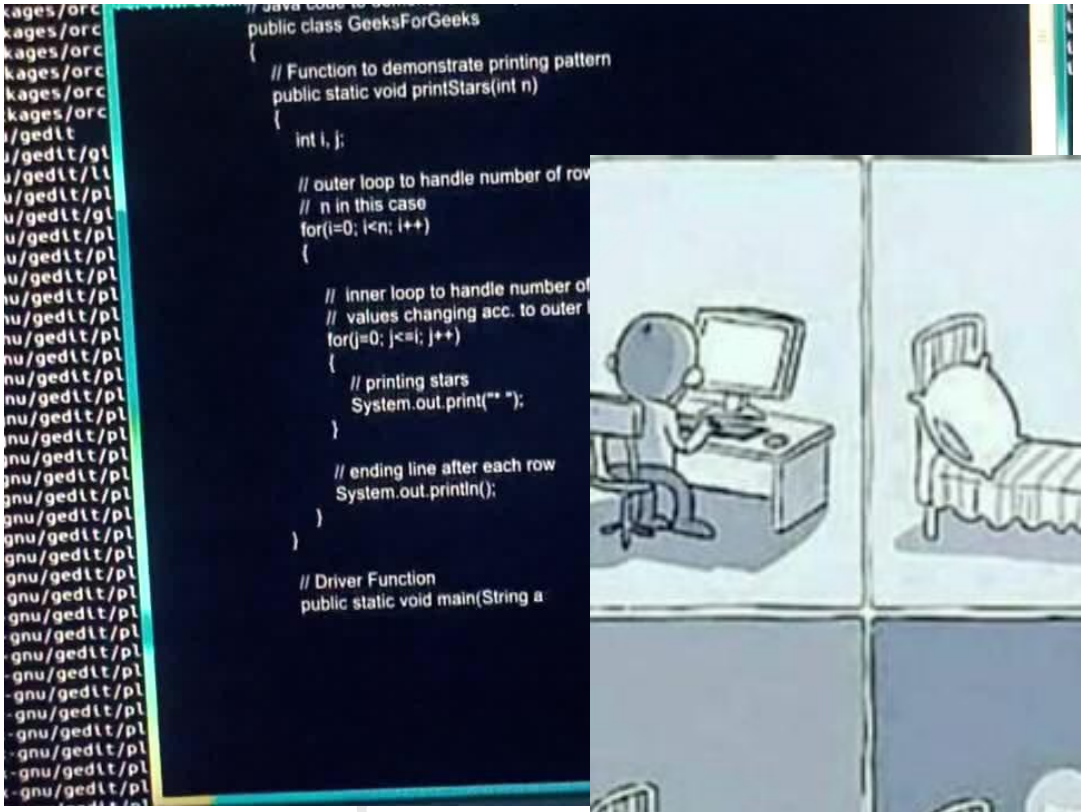
High and low-level languages

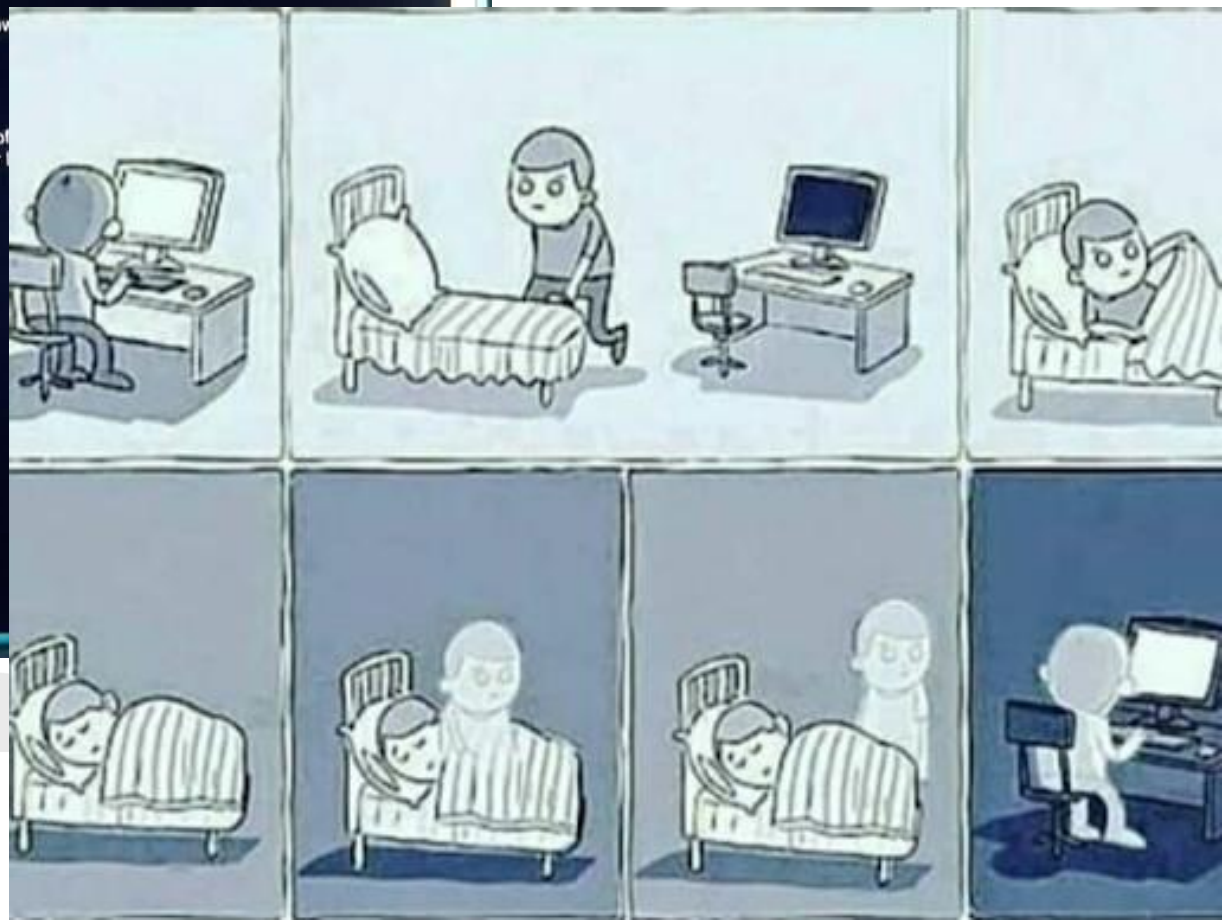
C = A + B;	C, C++, JAVA — High Level Language
ADD A , B	Assembly Language
100100111	Machine Language
✔	Hardware





Programmers?





In this course

- How to write programs?

- Program → a set of instructions → special notation or syntax (i.e., programming language)

- C++
  - Derived from C
  - C++ designer: Bjarne Stroustrup (1979)
  - Powerful and complex

- We will cover only a part of C++….

C++ basics

- Syntax: language-specific statements (instructions/ notations/ rules) which are legal

- Comments:
  - used to explain the code or increase the
  - Comments are ignored by compiler
  - Single/ multi-lined

  // looping over the array

  /* looping over

  the array*/

- Every instruction/ command is terminated by semi-colon “;”

Processing a C++ program

- Editor → simply writing the set of instructions as per the syntax (many text editors support syntax/ color highlighting)

- Preprocessor → these are directives which give instructions to the compiler to pre-process the information
  - Starts with # (only whitespace is allowed before this)
  - these are filenames in the Standard C++ Library.

#include <iostream> ← this is a C++ standard library header for input-output stream

#define PI 3.14

Processing a C++ program

- Compiler →
  - checks the program for syntax errors (set of rules which are language specific)
  - Compiler accepts almost any pattern of line breaks and indentation
  - Translate into machine language (low-level language);
  - Binary of “Hello World.”

```
01001000 01100101 01101100 01101100 01101111
00100000 01010111 01101111 01110010 01101100
01100100
```

- Execution
  - from top to bottom
  - from left to right

- Output

Program errors

- Syntax errors
  - Violation of the **grammar rules (syntax)** of the language
  - Discovered by the compiler
    - Error messages may not always show the correct location of errors
- Run-time errors
  - Error conditions detected by the computer at run-time
- Logic errors
  - Errors in the program's algorithm
  - Most difficult to diagnose
  - Computer does not recognize an error

Hello World!

```
#include <iostream>

int main( ) {

// hello world program
        std::cout << "Hello world";
        return 0;

}
```

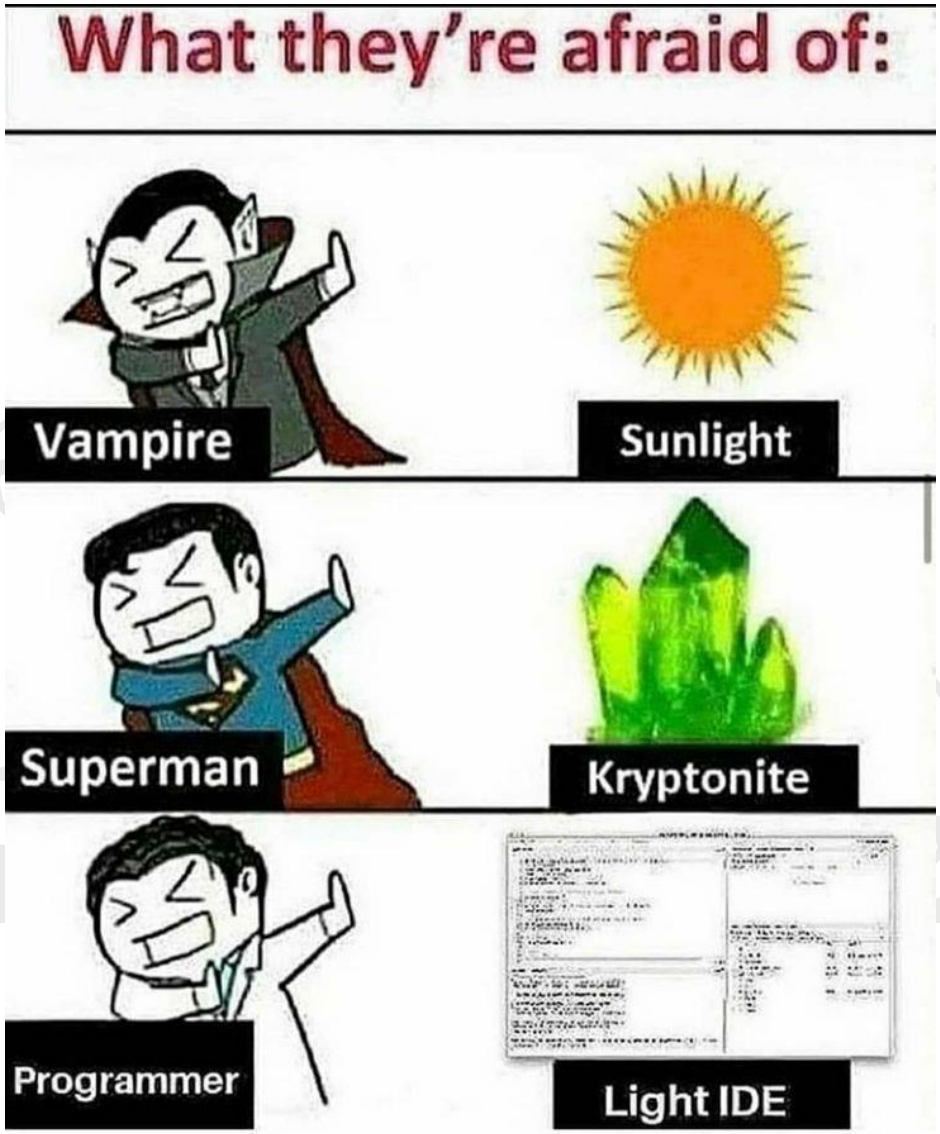
Where to run the program?

- Offline:
  - Dev-C++
  - Visual Studio
  - …
- Online:
  - https://www.programiz.com/cpp-programming/online-compiler/
  - https://www.tutorialspoint.com/compile_cpp_online.php
  - https://www.w3schools.com/cpp/cpp_compiler.asp

[[let's run the first program]]

Try to write a few programs, which prints your name/ enrolment number/ branch/ institute, etc.

Where to run the program?



Why C++ (or programming) in Civil Engineering?

- AutoCAD → C++
- SUMO (open-source): https://sumo.dlr.de/ → C++
- 12 D solutions → C++
- FreeFEM: https://freefem.org/ → C++
- …
- Others (general) → C++
  - Amazon
  - Google web search engine
  - Apple OS X (a few important parts)
  - Adobe systems
  - Bloomberg
  - Sun: Open Office
  - …
- Many other languages…

Special symbols

- +
- -
- *
- /
- .
- ;
- ?
- ,
- <=
- !=
- ==
- >=

Reserved keywords

- int
- float
- double
- char
- const
- void
- return
- switch
- while
- do
- for
- this
- catch
- enum
- public
- false
- true
- break
- continue
- namespace
- throw
- static
- private
- …

All small-case

C++ basic input, output

- ***cout*** → sends the output from main memory to standard devices (e.g., console screen)
- ***cin*** → takes the input from standard devices (e.g., keyboard, mouse) to the main memory
- header (required): ***#include <iostream>***
- For cout, insertion (**<<**) operators are used, whereas for cin extraction (**>>**) operators are used
- Let's look on the Hello World example

TODO: try ***clog*** and ***cerr***

cout

```
#include <iostream>

int main( ) {

// write the comment here…
        std::cout << "Hello world";
        return 0;

}
```

header (remember directives)!!

Name of the function; return type; every program must have exactly one main() function

Check cout, insertion operator, semi colon

What is this? Can we get rid of it?

Return statement; optional but some compilers expect it to be included as the last line of the main() function

cout

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

int main( ) {

// write the comment here…
        cout << "Hello world";
        return 0;

}
```

This means, all functions under std namespace are available in the scope of this program without explicitly prefixing "std::"

Let's look on another example

cout

What will be the output of this?

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

int main( ) {

        cout << "Hel" << "lo" <<"Wo" << "rld";
        return 0;

}
```

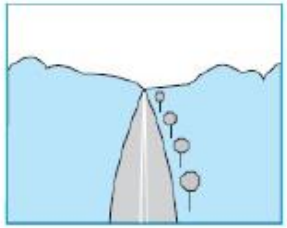
Let's look on another example

cin

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

int main( ) {

        int num ;
        cout << "Enter an Integer";
        cin >> num; // taking input
        cout << "Square of  the number is " << num*num;
        return 0;

}
```

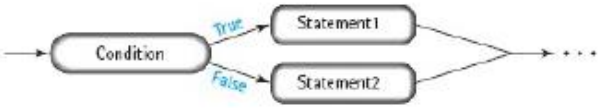
C++ Program

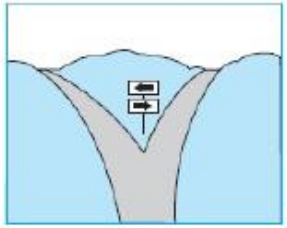
SEQUENCE



SELECTION (also called *branch* or *decision*)

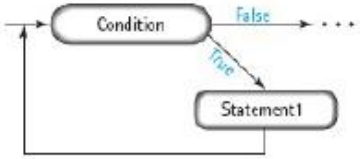
IF condition THEN statement1 ELSE statement2

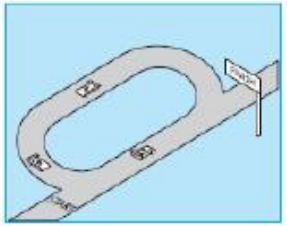




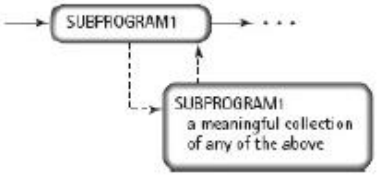
LOOP (also called *repetition* or *iteration*)

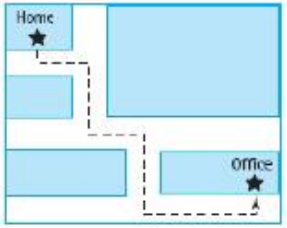
WHILE condition DO statement1





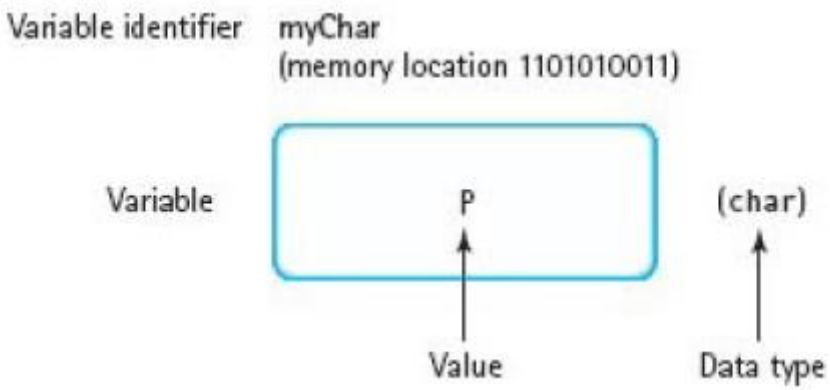
SUBPROGRAM (also called *procedure, function, method,* or *subroutine*)



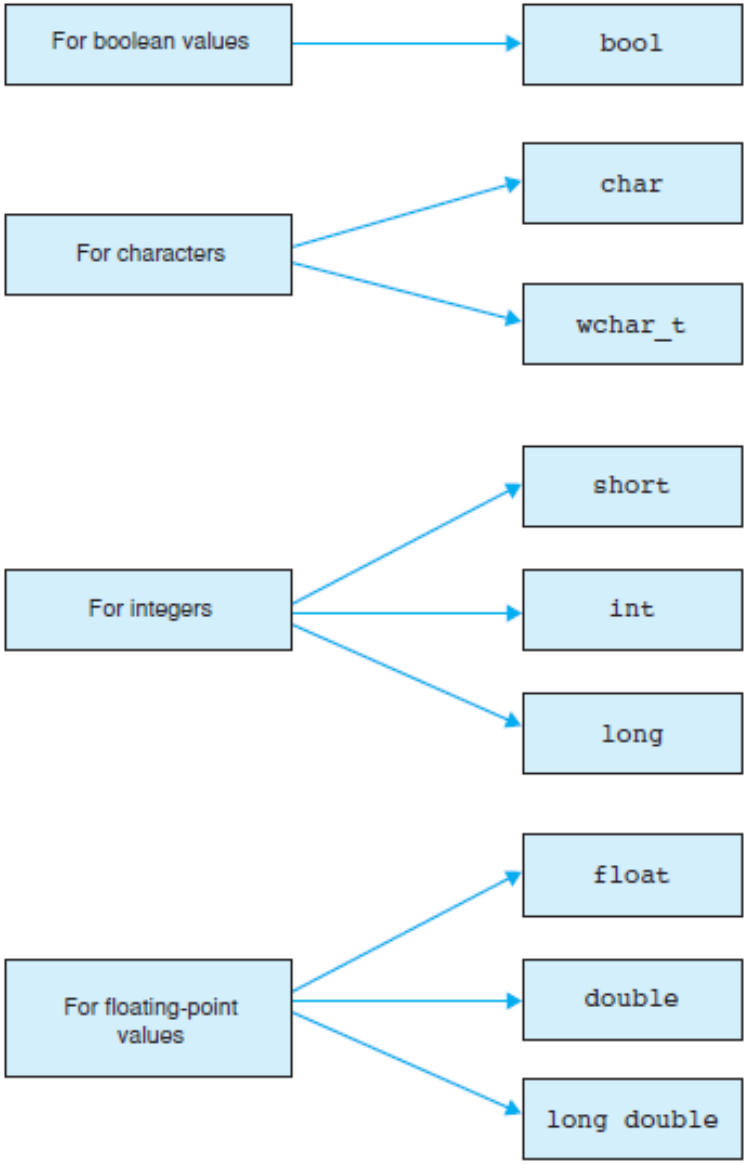


Data types

- Primitive data types
- Derived data types
- User defined data types



Data types


For boolean values
bool
For characters
char
wchar_t
For integers
short
int
long
For floating-point values
float
double
long double

Primitive Data types: char

- “`hello`” is a `string literal`
- `char` is stored in the computers as an integer
- enclosed in single quotes
- American Standard Code for Information Interchange (ASCII)
- Visit:
  - https://www.ascii-code.com/
  - https://en.cppreference.com/w/cpp/language/ascii
- Newline character ‘`\n`’ is one of the non-printing characters (e.g., `\n`, `\t`).
- Size 1 byte
- Range: -128 to 127 or 0 to 255 (on most compilers)

Primitive Data types

Data type	Meaning	Size (in bytes)	Other details
int *int age = 24;*	Integer	4	Range: -2147483648 to 2147483647; usually of 4 bytes
float *float speed = 60.5;*	Floating point (decimals)	4	Precision = 7 places
double *double dist = 4.45;*	Double floating point	8	Higher precision than float (14)
char *char te = 't';* *std::cout << int(te) ;*	Character	1	Inside single quotes; an integer value is stored rather than the character itself (see https://en.cppreference.com/w/cpp/language/ascii and https://www.ascii-code.com/) Range: -127 to 127 or 0 to 255
bool *bool val = false;*	Boolean	1	Two possible values: true, false

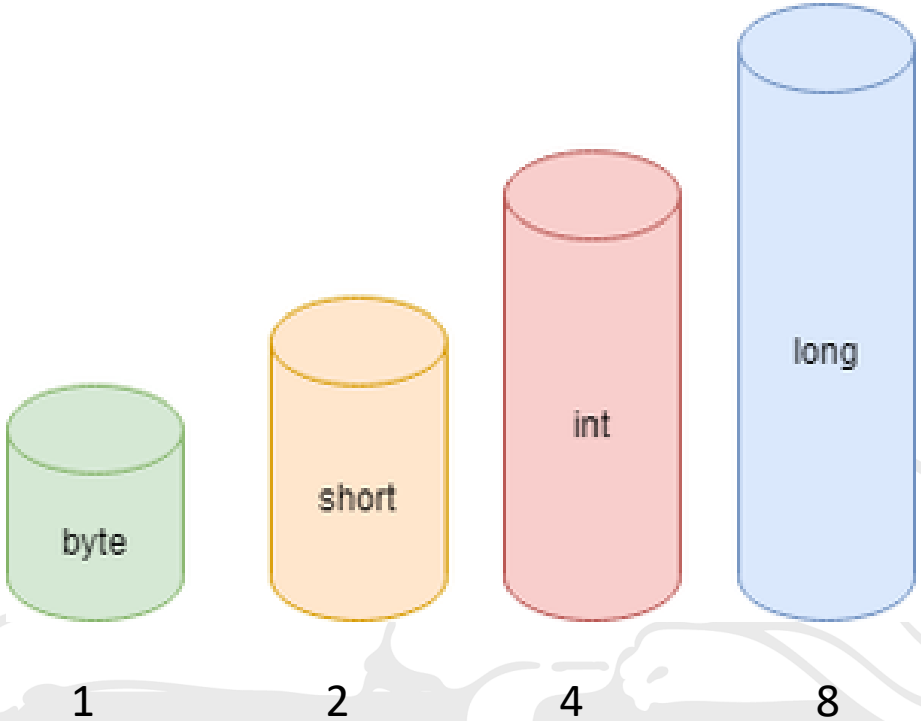
Do you really need 1 byte for bool?

Data types modifiers

- **signed**: int, char, long-prefix
- **unsigned**: int, char, long-prefix
- **short**: int
- **long**: int, double
- Execute the following and try to understand the results

```
cout << "Size of char : " << sizeof(char) << endl;
cout << "Size of int : " << sizeof(int) << endl;
cout << "Size of short int : " << sizeof(short int) << endl;
cout << "Size of long int : " << sizeof(long int) << endl;
cout << "Size of signed int : " << sizeof(signed int) << endl;
cout << "Size of unsigned int : " << sizeof(unsigned int) << endl;
cout << "Size of unsigned long int : " << sizeof(unsigned long int) << endl;
cout << "Size of float : " << sizeof(float) << endl;
cout << "Size of double : " << sizeof(double) << endl;
cout << "Size of long double : " << sizeof(long double) << endl;
cout << "Size of signed char : " << sizeof(signed char) << endl;
cout << "Size of unsigned char : " << sizeof(unsigned char) << endl;
```

Data types modifiers



Other Data types

- Derived data types: array, pointers, functions, etc.
- User defined: Class, Structure, Union, Enum, etc.

Variable names in C++

- Unique names (called as identifiers)
- This is basically, a name given to the memory location
- Variables must be declared before use.
- It could be short as (i,j,k or long as volume, speed)
- It may contain letters, digits, underscores (_)
- It must begin with a letter or underscore (i.e., it cannot begin with a number)
- Names are case sensitive
- Names cannot contain white spaces or special characters
- Reserved words cannot be used.
- Examples:

age	***_age***	***A_ge***	***a_g_e***
age23	***a23ge***	***a***	***ageOfPerson***

What will happen if you write wrong names? Try it.

Variable names in C++

- Syntax

  datatype variable_name;

- Though, it is a matter of personal preference, **descriptive names** are recommended for better **readability** of the code
- Unlike reserved words, predefined identifiers (e.g., cout, cin) can be redefined, but **<u>don't do that.</u>**

- Cases:
  - Camel ageOfPerson
  - Pascle AgeOfPerson
  - Snake age_of_person

Different style for local, instance, static, variables may be adopted for clarity.

Variable names in C++

int age = 24;

data_type variable_name value (initialize)

int time, speed;

What will be output of the following?

```
float number = 1/10;
std::cout << number <<std::endl;
```

What will be output of the following?

```
float number = 1.0/10.0;
std::cout << number <<std::endl;
```

What will be output of the following?

```
int number = 1.0/10.0;
std::cout << number <<std::endl;
```

Variable names in C++

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

int main( ) {

        int num ;          // declaration
        num = 23;          // initialization or assignment
        return 0;

}
```

A variable must be initialized before it is used (not necessarily during declaration).

A variable can be initialized by assignment and by taking from inputs.

Variable names in C++

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

int main( ) {

        int num1;
        int num2;
        num1=34;
        num2=90;
        cout << num1 << endl;
        cout << num2;
        return 0;
}
```

Variable names in C++

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

int main( ) {

        int num1 =34, num2=90; // declaration and initialization
        cout << num1 << endl;
        cout << num2;
        return 0;
}
```

Variable names in C++

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

int main( ) {

        int num1 =34, num2=90;
        cout << num1 << endl;
        cout << num2 <<endl;
        int num3 = num1 +num2; /* expression (on the right) is
evaluated first and then assigned to num3 */
        cout << num3;
        return 0;
}
```

Variable names in C++

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

int main( ) {

        int hours, minutes, seconds;
        cin>> hours >> minutes >> seconds;      // order is important
        cout<< hours << ":" << minutes << ":" << seconds;
        return 0;
}
```

Three variables can be entered separated by space or one number in each line (newline separator).

Operators

1. Arithmetic
2. Assignment
3. Relational
4. Logical
5. Other

Arithmetic Operators

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

Arithmetic Operators

```
int main( ) {
        int a =34, b=90;
        cout  <<  "a+b="  << (a+b) << endl;
        cout  <<  "a-b="  << (a-b) << endl;
        cout  <<  "a*b="  << (a*b) << endl;
        cout  <<  "a/b="  << (a/b) << endl;
        cout  <<  "a%b="  << (a%b) << endl;
        return 0;
}
```

```
int main( ) {
        int a =3;
        int x = a++;
        cout  <<  x << endl;
        cout  <<  a << endl;
        x = ++a;
        cout  <<  x << endl;
        cout  <<  a << endl;
        return 0;
}
```

Unary operators

- Pre-increment ++x (also --x)
- Post-increment x++ (also x--)

Assignment Operators

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

Assignment Operators

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

int main( ) {

        int a =3, b=9;
        cout  <<  “Old value of a:"  << a << endl;
        a += b
        cout  <<  “New value of a:"  << a << endl;
        return 0;

}
```

Order of Precedence

- All operations inside of () are evaluated first
- *, /, and % are at the same level of precedence and are evaluated next
- + and – have the same level of precedence and are evaluated last
- When operators are on the same level
  - Performed from left to right (associativity)

Mixed expression

- Integral expression 2 + 3 * 6
- Floating expression 2.5 + 3.1 * 3.2
- Mixed expression evaluation: 2 + 3.5 / 2
  - If operator has same types of operands
    - Evaluated according to the type of the operands
  - If operator has both types of operands
    - Integer is changed to floating-point
    - Operator is evaluated
    - Result is floating-point
  - Entire expression is evaluated according to precedence rules

Relational Operators

Operator	Meaning	Example
==	Is Equal To	3 == 5 gives us **false**
!=	Not Equal To	3 != 5 gives us **true**
>	Greater Than	3 > 5 gives us **false**
<	Less Than	3 < 5 gives us **true**
>=	Greater Than or Equal To	3 >= 5 give us **false**
<=	Less Than or Equal To	3 <= 5 gives us **true**

Logical Operators

Operator	Example	Meaning
&&	expression1 **&&** expression2	Logical AND. True only if all the operands are true.
||	expression1 **||** expression2	Logical OR. True if at least one of the operands is true.
!	**!**expression	Logical NOT. True only if the operand is false.

Logical Operators

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

int main( ) {

        bool out = (3 != 5) && (3 < 5); // 1 (true)

        bool out2 = !(5 == 2);    // 1 (true)

        bool out3 = !(5 == 5);   //0 (false)
        return 0;

}
```

other Operators

Operator	Description	Example
sizeof	returns the size of data type	sizeof(int); // 4
?:	returns value based on the condition	string result = (5 > 0) ? "even" : "odd"; // "even"
&	represents memory address of the operand	# // address of num
.	accesses members of struct variables or class objects	s1.marks = 92;
->	used with pointers to access the class or struct variables	ptr->marks = 92;
<<	prints the output value	cout << 5;
>>	gets the input value	cin >> num;

Mathematical Functions

- Large number of mathematical functions are available in standard C++

- To use them, include:

```
#include<cmath>

#include<math.h>
```

- Examples:
  - sin, cos, tan …
  - asin, acos, atan …
  - sqrt, abs, pow, floor, ceil …
  - log, exp, …
  - See https://www.cplusplus.com/reference/cmath/

Mathematical Functions

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

int main( ) {

        int num = -10;
        cout << abs(num) << endl;

        int a = 25;
        cout << sqrt(a) << endl;

        int b = 5;
        int c = 3;
        cout << pow(5,3);
        return 0;
}
```

Conditional Statements

- Remember Relational and logical operators
- Use **if** to execute a block of code, if a condition is met (i.e., true)
- Use **else** to execute a block of code, if the same condition is false
- Use **else if** to test a new condition, if the first condition is false
- Use **switch** to execute many alternative blocks based on different criteria of a variable

```
if (condition) {
  // block of code to be executed if the condition is true
}

if (35 >= 20) {
  cout << “35 is greater than or equal to 20";
}
```

Conditional Statements

Truth Tables

AND

Exp_1	Exp_2	Exp_1 && Exp_2
true	true	true
true	false	false
false	true	false
false	false	false

OR

Exp_1	Exp_2	Exp_1 | | Exp_2
true	true	true
true	false	true
false	true	true
false	false	false

NOT

Exp	!(Exp)
true	false
false	true

Conditional Statements

The unary operators +, –, ++, --, and !.
The binary arithmetic operations *, /, %
The binary arithmetic operations +, –
The Boolean operations <, >, <=, >=
The Boolean operations ==, !=
The Boolean operations &&
The Boolean operations ||

Highest precedence (done first)

↓

Lowest precedence (done last)

Conditional Statements

```
int speed = 30;
if (speed < 40) {
  cout << “Good going…";
} else {
  cout << “Too fast, slow down…";
}
```

```
int speed = 30;
if (speed < 40) {
  cout << “Good going…";
} else if (speed < 50) {
  cout << “Just right …";
}  else {
  cout << “Too fast, slow down…";
}
```

Conditional Statements

```
if ( (kids != 0) && ((pieces/kids) >= 2) )
    cout << "Each child may have two pieces!";
```

```
if (!time > limit)
    Something
else
    Something_Else
```

```
if (!(time > limit))
    Something
else
    Something_Else
```

```
if (time <= limit)
    Something
else
    Something_Else
```

Conditional Statements

```
int speed = 30;
if (speed < 40) {
  cout << “Good going…";
} else {
  cout << “Too fast, slow down…";
}
```

```
int speed;
cin >> speed;
if (speed < 40) {
  cout << “Good going…";
} else if (speed < 50) {
  cout << “Just right …";
}  else {
  cout << “Too fast, slow down…";
}
```

Conditional Statements

```
if (fuel_gauge_reading < 0.75)
    if (fuel_gauge_reading < 0.25)
        cout << "Fuel very low. Caution!\n";
    else
        cout << "Fuel over 3/4. Don't stop now!\n";
```

Conditional Statements

```
int speed = 30;
string message = (speed < 40) ? "Good going…" : “Too fast, slow down…";
cout << message;
```

Conditional Statements

```
int day = 4;
switch (day) {
  case 1:
    cout << "Monday";
    break;
  case 2:
    cout << "Tuesday";
    break;
  case 3:
    cout << "Wednesday";
    break;
  case 4:
    cout << "Thursday";
    break;
  case 5:
    cout << "Friday";
    break;
  default:
    cout << “Enjoying the Weekend";
}
```

It breaks out of the switch block, i.e., no more case testing inside the switch block.

Conditional Statements: nested

```
int speed = 30;
int vehicle = 1; // 1→ car, 2→ MTW
if (speed < 40) {
        if (vehicle == 1) {
                cout << “Good going…";
        } else {
                cout << “Be careful, out there.";
        }
} else {
  if (vehicle == 1) {
                cout << “Too fast, slow down …";
        } else {
                cout << “Dangerous speed…";
        }
}
```

Loops

- Repetition
- To execute a block of code as long as a specific condition is met
- Loops are handy, saves time, reduce errors, and are more readable

```
while (condition) {
  // code block to be executed
}
```

The while loop loops through a block of code as long as a specified condition is true.

```
do {
  // code block to be executed
}
while (condition);
```

This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
for (initialization; condition; update) {
  // code block to be executed
}
```

If it is known that a block of code is to be executed 'n' times, use the for loop instead of a while loop:

Loops

```
int i = 0;
int max;
cin >> max;
while (i < max) {
  cout << i << "\n";
  i++;
}
```

```
int i = 0;
int max;
cin >> max;
do {
  cout << i << "\n";
  i++;
} while (i < max);
```

```
for (int i = 0; i < max; i++) {
  cout << i << "\n";
}
```

Loops

- Compute the average of numbers as long as a user enters the numbers

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

int main() {
    int x, count = 0;
    float sum = 0.0;
    cout << "Please enter some integers:\n (Break with any letter)" << endl;

    while( cin >> x ) {
        sum += x;
        ++count;
    }

    cout << "The average of the numbers: " << sum / count << endl;
    return 0;
}
```

Loops

- Multiple variables in the loop

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

int main(){
    int i, j;

    for (i = 1, j = 5; i < 5; i++, j++);

    cout << "Value of i = " << i << endl;
    cout << "Value of j = " << j << endl;

    return 0;
}
```

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

int main(){
    int i, j;

    for (i = 1, j = 5; i < 5; i++, j++) {
        cout << "Value of i = " << i << endl;
        cout << "Value of j = " << j << endl;
    }


    return 0;
}
```

Loops

- Multiple variables in the loop

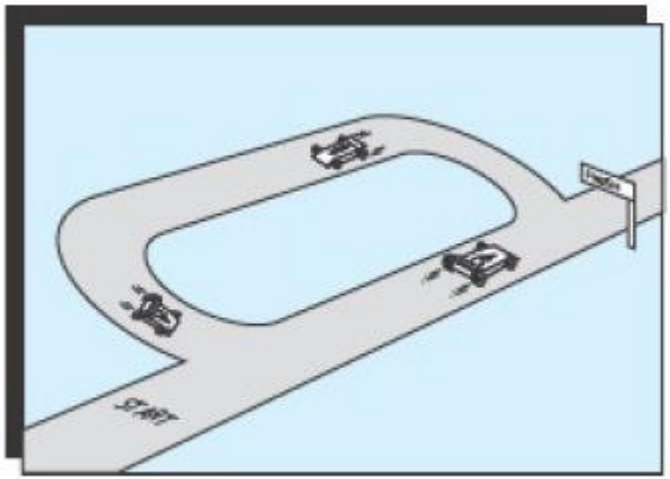
```
#include <iostream>
using namespace std;
int main()
{
    int sum = 0;
    int j = 100;
    for(int i = 1; i<=100/2 && j>100/2; i++){
        sum += i+j;
        j--;
    }

    cout<<sum;

}
```

```
#include <iostream>
using namespace std;
int main()
{
    int sum = 0;
    for(int i = 1; i<=100; i++){
        sum += i;
    }

    cout<<sum;
    return 0;

}
```

Loops

- Infinite loop

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

int main(){
    int i, j;

    for (;;) {
        cout << "dont do this";
}

    return 0;
}
```



Loops: nested

```
for (int i = 0; i < max; i++) {
        for (int j = 0; j < max; j++) {
                cout << i << "\n";
        }
}
```

How can we get the following?

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

How can we get the following?

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

Jumps with *break*, *continue* and *goto*

- ***break***: The break statement exits from a switch or loop immediately. You can use the break keyword to jump to the first statement that follows the switch or loop.

- ***continue***: The continue statement can be used in loops and has the opposite effect to break, that is, the next loop is begun immediately. In the case of a while or do-while loop, the program jumps to the test expression, whereas a for loop is reinitialized.

- ***goto***: C++ also offers a goto statement and labels. This allows you to jump to any given point marked by a label within a function. For example, you can exit from a deeply embedded loop construction immediately. A label is a name followed by a colon.

```
for( . . . )
   for( . . . )
     if (error) goto errorcheck;
 . . .
errorcheck: . . .        // Error handling
```

break

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  cout << i << "\n";
}
```

```
int i = 0;
while (i < 10) {
  cout << i << "\n";
  i++;
  if (i == 4) {
    break;
  }
}
```

break

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {

    int ac = 32;

    while(true) {
        cout << "\n Character \t Decimal \t Hexadecimal \n\n";
        int upper;
        for( upper = ac + 20; ac < upper && ac < 256; ++ac)
            cout << " " << (char) ac << "\t\t" << dec << ac << "\t\t" << hex << ac << endl;

        if( upper >= 256) break;

        cout <<"Continue -> y; End -> n \n";

        char answer;
        cin >> answer;

        if( answer == 'n' || answer == 'N' )
            break;

        }
    return 0;
}
```

continue

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  cout << i << "\n";
}
```

```
int i = 0;
while (i < 10) {
  if (i == 4) {
    i++;
    continue;
  }
  cout << i << "\n";
  i++;
}
```

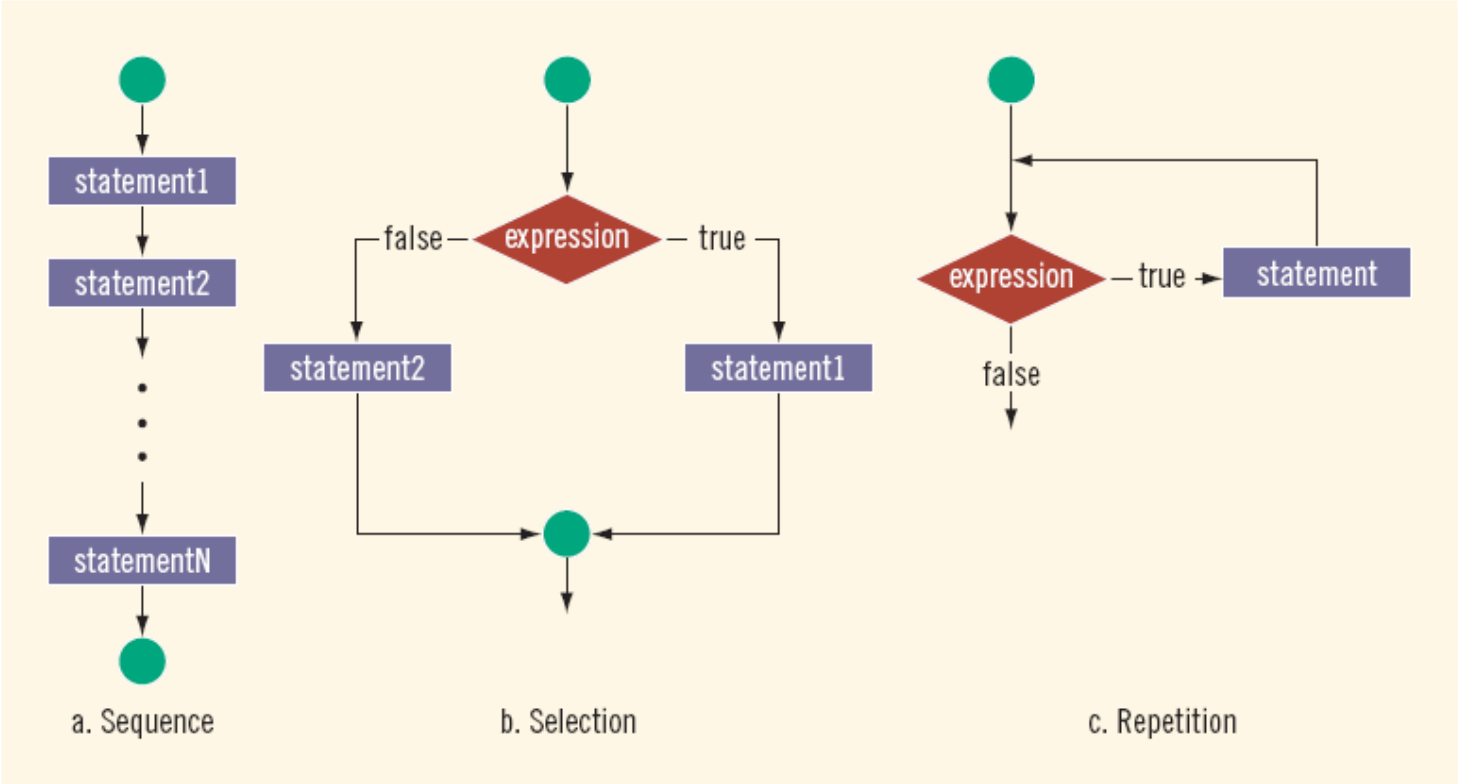
goto

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

int main()
{
    float num, average, sum = 0.0;
    int i, n;

    cout << "Maximum number of inputs: ";
    cin >> n;

    for(i = 1; i <= n; ++i)
    {
        cout << "Enter n" << i << ": ";
        cin >> num;

        if(num < 0.0)
        {
            // Control of the program move to jump:
            goto jump;
        }
        sum += num;
    }

jump:
    average = sum / (i - 1);
    cout << "\nAverage = " << average;
    return 0;
}
```

goto

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

int main () {
   // Local variable declaration:
   int a = 10;

   // do loop execution
   LOOP:do {
      if( a == 15) {
         // skip the iteration.
         a = a + 1;
         goto LOOP;
      }
      cout << "value of a: " << a << endl;
      a = a + 1;
   }
   while( a < 20 );

   return 0;
}
```

Control structure in C++



string in C++

- string is a variable, which is a collection of characters
- Surrounded by double quotes (“my string”)
- Example

```
string age = “34";
string message("Good morning!");
```

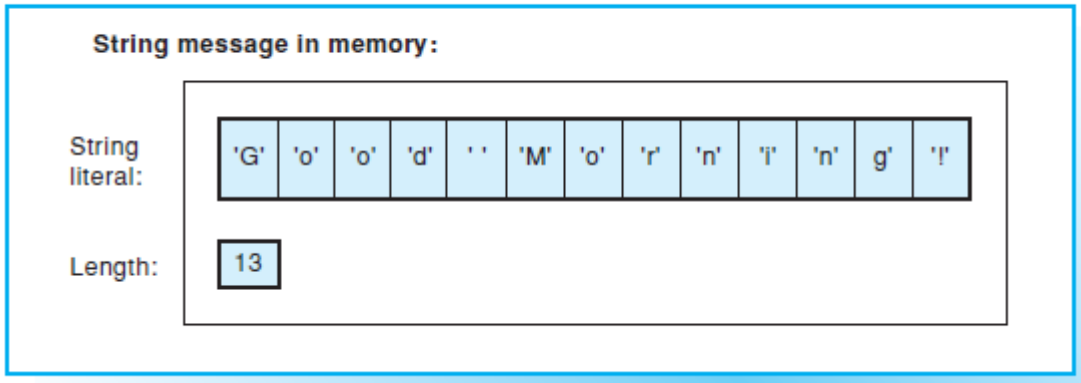
- Size of string

```
age.size();
age.length();
```

Try to use sizeof(age) and see the output. Does it make sense?

Note that: sizeof only reports the memory occupied by the variable

```
string message = "Good Morning!";
```

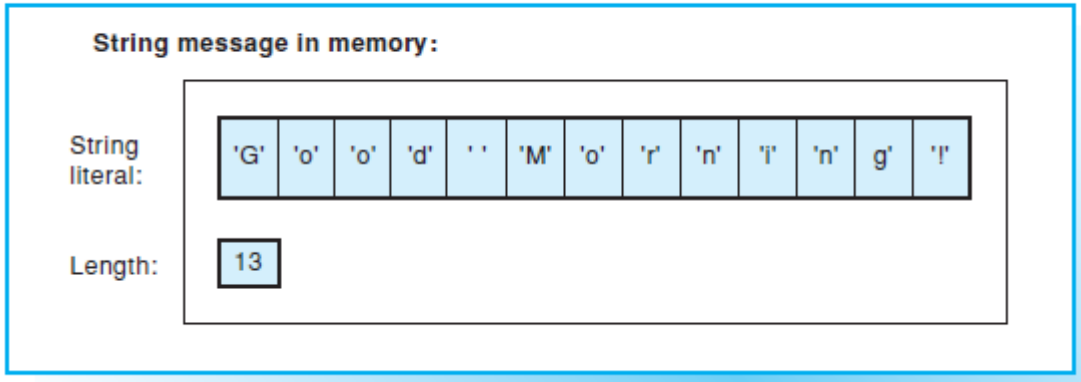


string in C++

- Each character has a relative position in the *string*
  - Position of the first character is 0

- The length of a string is the number of characters in it
  - Example: length of "`Good Morning!`" is 13

```
string message = "Good Morning!";
```



string in C++

- endl → causes insertion point to move to beginning of next line
- Escape sequence:

	Escape Sequence	Description
\n	Newline	Cursor moves to the beginning of the next line
\t	Tab	Cursor moves to the next tab stop
\b	Backspace	Cursor moves one space to the left
\r	Return	Cursor moves to the beginning of the current line (not the next line)
\\	Backslash	Backslash is printed
\'	Single quotation	Single quotation mark is printed
\"	Double quotation	Double quotation mark is printed

string in C++

- Relational operators can be applied to strings
- Strings are compared character by character, starting with the first character
- Comparison continues until either a mismatch is found, or all characters are found equal
- If two strings of different lengths are compared and the comparison is equal to the last character of the shorter string
  - The shorter string is less than the larger string

string in C++

```
string str1 = "Hello";
string str2 = "Hi";
string str3 = "Air";
string str4 = "Bill";
string str5 = "Big";


str1 < str2; //  true
str1 > “Hen”; // false
str3 < “An”; //  true
str1 ==“hello”; // false
str3 <= str4; // true
str2 > str4; // true
Str4 >= “Billy”; //false
Str5 <= “Bigger” // true
```

What will be the output of…

```
bool out = 'H' < 'h';
std::cout << std::boolalpha<< out << std::endl;
```

You may need https://en.cppreference.com/w/cpp/language/ascii

String Concatenation

- The + operator can be used between two strings to make a new string
- White space can also be concatenated.

```
string firstName = “Amit ";
string lastName = “Agarwal";
string fullName = firstName +
lastName;
cout << fullName;
```

- A string is an object, thus append() can also be used to concatenate strings.

```
string firstName = “Amit ";
string lastName = “Agarwal";
string fullName
= firstName.append(lastName);
cout << fullName;
```

String Concatenation

- Append allows part of a string.

  str1.append(str2, 0, 5);

What will be the output of…

```
int x = 10;
int y = 20;
int z = x + y; // output →??


string x = "10";
string y = "20";
string z = x + y; // output →??
```

String Access

- String index starts at 0 (i.e., first character)
- The characters in the string can be accessed using index numbers inside [ ]

What will be the output of…

```
string myString = "Hello";
cout << myString[1]; // output →??
```

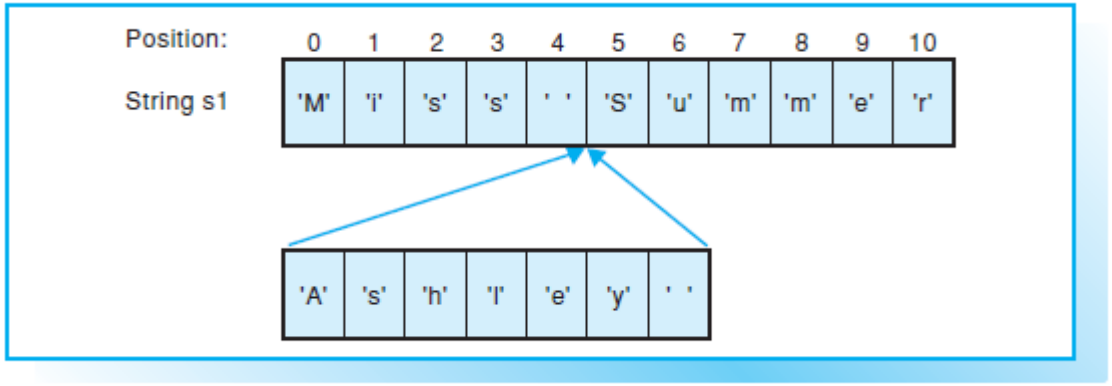
- It can also replace the existing character

What will be the output of…

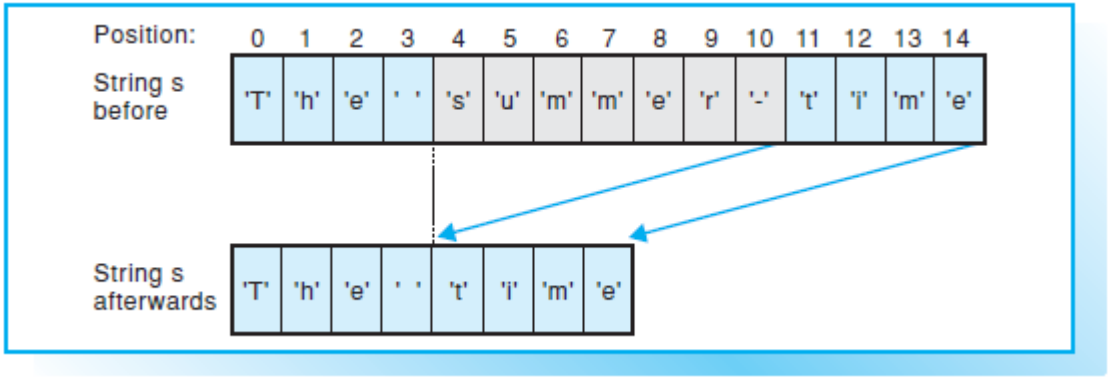
```
string myString = "Hello";
myString[0] = 'J';
cout << myString;
```

String inserting and erasing

```
string s1("Miss Summer");
s1.insert(5, "Ashley "); // Insert at position: 5
```




```
string s("The summer-time");
s.erase(4,7); // position: 4, quantity 7
```



String Access

- Taking a string as input with whitespace in it.

What will be the output of…

```
string name;
cin >> name;   // Enter “Amit Agarwal”
cout << name;
```

- cin considers a space (whitespace, tabs, etc.) as a terminating character, which means that it can only display a single word

What will be the output of…

```
string name;
cout << "Enter your name" <<endl;
getline (cin, name);
cout << name;
```

Thanks …

amitfce@iitr.ac.in