# Data Structure
# in
# C++ Language

Lecturer:- Samer Al-khazraji

Reference:- Any C++ text Book

- **Data structure, why?**
- Is a power computer always an efficient solution for a complex problem?
- A computer uses any data structure to solve a problem.
- A data structure is a representation of data.
- A good selection of data structure can lead to an efficient solution for a problem.
- The efficient solution is a method of solving the problem in a limit of resources (storage and time).

# Steps of choosing of data structure

- Investigate the problem to define the supported operations (insert, delete, and search).
- Determine the resource constraints for each operation.
- Choose a data structure that matches these requirements.

# Features of C++ Language

- C++ is an object-oriented programming language and also a low-level programming language.
- It is compatible with C language.
- C++ programmer can have a complete control over memory management.
- It supports different kinds of applications; GUI, 3D graphics, mathematical operations, and etc.
- A good opportunity for getting jobs for C++ programmers.

# File structure in C++

Example:-

```
/*  Multiple lines comment */
//Single line comment

//This is where the execution of program begins


int main()
{

  cout<<"Hello World!";       // displays Hello World! on screen


  return 0;
}
```

# Types of Operators in C++

1- Basic Arithmetic Operators:

- The operators are: +, -, *, /, %
- + is for addition.
- – is for subtraction.
- * is for multiplication.
- / is for division.
- % is for modulo.
  Note: Modulo operator returns remainder, for example 20 % 5 would return 0.

# Types of Operators in C++

2- Assignment Operators

- These operators are: =, +=, -=, *=, /=, %=
- num2 = num1 would assign value of variable num1 to the variable.
- num2+=num1 is equal to num2 = num2+num1
- num2-=num1 is equal to num2 = num2-num1
- num2*=num1 is equal to num2 = num2*num1
- num2/=num1 is equal to num2 = num2/num1
- num2%=num1 is equal to num2 = num2%num1

# Types of Operators in C++

3- Auto-increment and Auto-decrement Operators

- ++ and —
- num++ is equivalent to num=num+1;
- num—- is equivalent to num=num-1;

# Types of Operators in C++

## 4- Logical Operators

- Logical Operators are used with binary variables. They are mainly used in conditional statements and loops for evaluating a condition.

- Logical operators in C++ are: &&, ||, !

# Types of Operators in C++

## 5- Relational operators

- C++ has six relational operators: ==, !=, >, <, >=, <=.
- == returns true if both the left side and right side are equal
- != returns true if the left side is not equal to the right side of operator.
- > returns true if the left side is greater than right.
- < returns true if the left side is less than the right side.
- >= returns true if the left side is greater than or equal to the right side.
- <= returns true if the left side is less than or equal to the right side.

# Types of Operators in C++

## 6- Bitwise Operators

- Bitwise operators in C++ are six: &, |, ^, ~, <<, >>

For example:-

Let **N1** and **N2** two vriavbles

- **N1 & N2**, it is comparison operation between two corresponding bits of N1 and N2 and produces 1 if both bits are equal, otherwise, it returns 0.
- **N1 | N2**, it is comparison operation between two corresponding bits of N1 and N2 and produces 1 if either bit is 1, otherwise, it returns 0.
- **N1 ^ N2**, it is comparison operation between two corresponding bits of N1 and N2 and produces 1 if they are not equal, otherwise, it returns 0.
- **~N1**, the operator ~ changes the bit from 0 to 1 and vice versa.
- **N1 << 2**, the left shift operator << moves the bits of N1 by 2 to the left and ignores the far left bit, and assigns the rightmost bit value of 0.
- **N1 >> 2**, the right shift operator >> moves the bits of N1 to the right and ignores the far right bit, and assigns the leftmost bit value of 0.

# Types of Operators in C++

7- Ternary Operator

This operator assigns the value based on the evaluation of a Boolean expression.

For example:-

- variable N1 = (expression)  ? value1 : value2
- It means; N1 = value1 else N1 = value2

# Types of Operators in C++

## 8) Operator Precedence in C++

It defines which operator should be processed first if the expression has more than one operator. The operator of higher precedence will at the top and the lower precedence will be at the bottom.

- **Unary Operators** ++ – – ! ~
- **Multiplicative** * / %
- **Additive** + –
- **Shift** << >>
- **Relational** > >= < <=
- **Equality** == !=
- **Assignment** = += -= *= /= %= > >= < <= &= ^= |=

# Mathematical standard functions

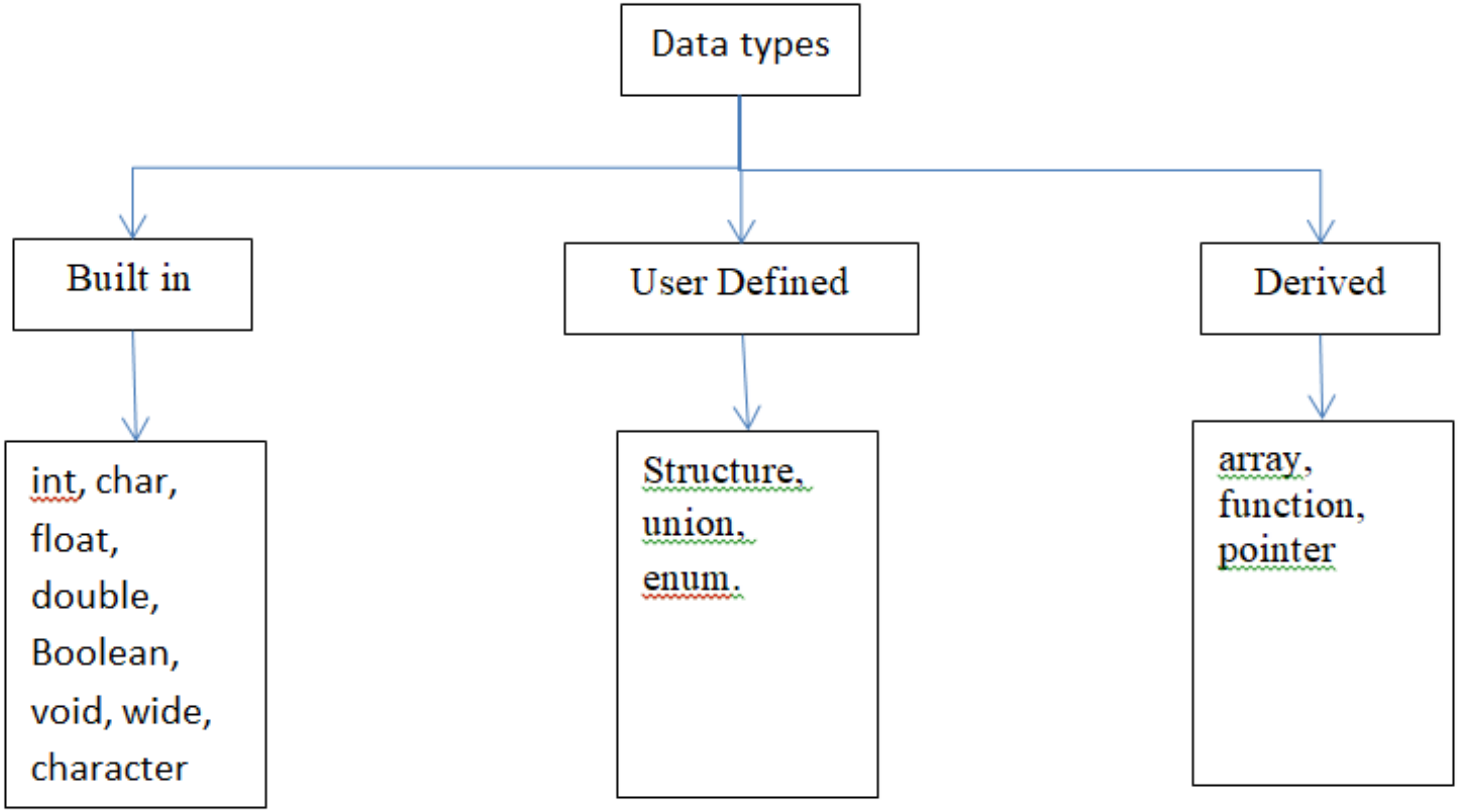
- double sin (double); // Sine
- double cos (double); // Cosine
- double tan (double); // Tangent
- double atan (double); // Arc tangent
- double cosh (double); // Hyperbolic Cosine
- double sqrt (double); // Square Root
- double pow (double, double); // Power
- double exp (double); // Exponential Function
- double log (double); // Natural Logarithm
- double log10 (double); // Base-ten Logarithm

# Types of variables based on their scope

- Global variable: the variable that is defined outside of the main function.

- Local variable: is the variable that is defined between the brace of function, inside of function, loops, control statement. The scope of these variables is limit inside the field that is defined in.

# Data types in C++



# Built in data type

- **char**: For characters. Size 1 byte,

  char ch = ‘A’;
- **int**: For integers. Size 2 bytes.

  int num = 100;
- **float**: For single precision floating point. Size 4 bytes.

  float num = 123.78987;
- **double**: For double precision floating point. Size 8 bytes.

  double num = 10098.98899;
- **bool**: For booleans, true or false.

  bool b = true;

# Statements and Control structures

A general view of C++ statements is that it consists of two types: **simple statement** and **compound statement**.

1- **A Simple statement**, it consists of a single statement which ends with a semicolon.

Examples:

```
a =  34:
a = b + c;
y = sin(x);
```

# Statements and Control structures

2- **A Compound statement**, it can be shown in two variety:

a- **Loop Statements**:

The **while** loop,

While body will be executed when the while loop test is true.
Syntax:

```
while (test Expression)
{
        // codes
}
```

# Statements and Control structures

The **do . . while** loop, it is similar to while loop, but, it tests the expression at the end of the loop.

Syntax:

```
Do
{
        Expressions;
}
While (test);
```

# Statements and Control structures

The **for** loop, it is useful when the loop is repeated many times based on a sequence of values based on a value stored on variable.

Syntax:

```
for ( initial value; condition; counter)
{
        Expression(s);
}
```

# Statements and Control structures

The **nested for** loop, it is a loop of "if" body inside another loop of “if” body.

Syntax:

```
for ( initial value; condition; counter)
{
        for ( initial value; condition; counter)
        {
        Expression(s);
        }
   Expression(s);
}
```

# Statements and Control structures

b- **Conditionals**, this statement executes one or another block depending on the result of the condition.

**“If”** condition

Syntax:

```
If ( condition)
        {
                Execute this block when “if” is true;
        }
else
        {
                Execute this block when “if” is false;
        }
```

# Statements and Control structures

**Nested "if"** condition, it is a body of conditional "if" statement inside another conditional "if" statement.

Syntax:

```
if ( condition)
{
        if
        {
        Statements:
        }

else
   {
        Statements;
    }
}
```

# Statements and Control structures

**Nested "if..else"** condition, it is multiple stages of data checking.

Syntax:

```
if ( condition)
{
Statements;
}
else if (condition)
        {
        Statements;
        }
else
{
Statements;
}
```

# Statements and Control structures

**Switch** statement, It is a good alternative to the nested **if..else** statement.

Syntax:

```
switch (n)          // n is expression
{
        case 1:
                // code to be executed if n = 1;
                        break;

        case 2:
                // code to be executed if n = 2;
                        break;

        default: // code to be executed if n doesn't match any cases
}
```

# Derived data types in C++

- We have three types of derived-defined data types in C++
    1. Array
    2. Function
    3. Pointer

# Array in C++

- An array is a set of the same type elements stored in adjacent memory locations that can be individually referenced by adding an index to a unique identifier.

**Declaration of array**:

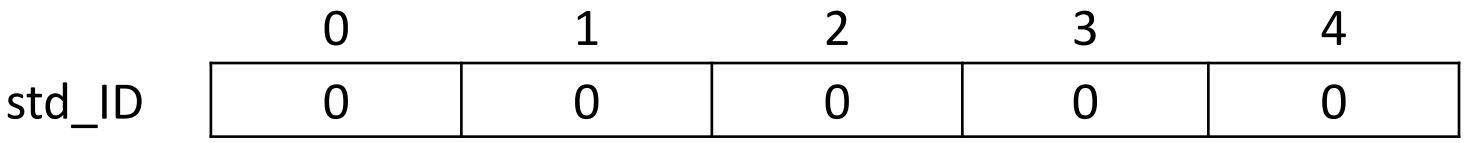
| Type | name | [No. element] |
|---|---|---|
| int | std_ID | [5] |

std_ID, is integer array of 5 elements.

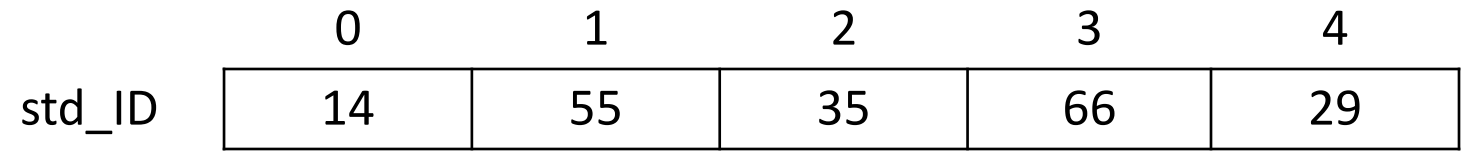
# Array in C++

**Array initailisation**:

```
int  std_ID [5] = {};
```




```
int  std_ID [5] = {14, 55, 35, 66, 29};
```



# Array in C++

## Array initialisation (followed):

- int std_ID [5] = {14, 55, 35};

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| std_ID | 14 | 55 | 35 | 0 | 0 |

## Accessing a value in array:

std_ID[2] = 70; // assign value 70 to the cell ‘2’;

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| std_ID | 0 | 0 | 70 | 0 | 0 |

# Array in C++

- **Insert values to array**:

```
for (i=0; i<5; i++)
{
cin>>var;
std_ID [i] = var;
}
```

- **Print the content of array**:

```
for (i=0; i<5; i++)
{
Cout<< std_ID[i];
}
```

# Two Dimensional Array in C++

It is a list of arrays and it can be represented as a table.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | A | B | C | D |
| 1 | E | F | G | H |
| 2 | I | J | K | L |

# Two Dimensional Array in C++

A two-dimensional array is declared and initialised as follows:

Type array-name [row][column]

```
int arr[3][4] = {
        {A, B, C, D} ,   /*  initializers for row indexed by 0 */
        {E, F, G, H} ,    /*  initializers for row indexed by 1 */
        {I, J, K, L}          /*  initializers for row indexed by 2 */
                };
```

# Two Dimensional Array in C++

- Accessing an element in a second row and third column in a two-dimensional array will be as follows:

```
Array_Name [2][3]
```

- Insert values to array (arr[[][]):

```
for(i=0; i<row; i++)
        {
        for(j=0; j<col; j++)
            {
                cin>>arr[i][j];
            }
        }
```

- Print the content of array (arr[][]):

```
for(i=0; i<row; i++)
        {
        for(j=0; j<col; j++)
            {
                cout<<arr[i][j];
            }
        }
```

# Functions in C++

A function is a part of code that performs a specific task.

Types of functions:

1- Library Functions, which are built–in functions in C++ programming.
Example:

```
#include <iostream>
#include <math>

int main()
{
   double number, squareRoot;
   cout << "Enter a number: ";
   cin >> number;
   squareRoot = sqrt(number);
   cout << "Square root of " << number << " = " << squareRoot;
   cin>>" ";
   return 0;
}
```

Where ‘math’ is a heard file uses to define sqrt() library function used to calculate square root.

# Functions in C++

2- User-Defined Function is a function that is built by a user for a particular task. There are four types of user-defined functions:

**A-** Passed arguments to function and return values:

Example:

```
#include <iostream>
int add(int, int);                          // Function declaration (Function prototype)
int main()
{
   int num1, num2, sum;
   cout<<"Enters two numbers to add: ";
   cin >> num1 >> num2;
    sum = add(num1, num2);   // Function call
   cout << "Sum = " << sum;
   cin>>" ";
   return 0;
}
```

---------- continued ----------

# Functions in C++

```
int add(int a, int b)                    // Function definition
{
   int add;
   add = a + b;
   return add;                                          // Return statement
}
```

The variables *num1* and *num2* are called actual arguments and they initialized the variables *a* and *b* that known as formal arguments.

# Functions in C++

**B-** Function no arguments passed and n return value:

Example:

```
#include <iostream>

void even();
void odd();

void main()
{
   int num;
   cout<<"Enters number to check: ";
    cin >> num;
   if (num%2 == 0)
                even();
   else
                odd();
}
```

```
void even()
{
                cout<<" the number is even";
  cin>>" ";
}
void odd()                      // Function definition
{
    cout<< "The number is odd";
    cin>>" ";
}
```

# Functions in C++

**C-** No arguments passed to function but return value;
Example:

```
#include <iostream>

void even();
void odd();

int read();

void main()
{
   int num;
   num = read();
   if (num%2 == 0)
                even();
   else
                odd();
}
```

---------- continue ----------

# Functions in C++

```
nt read()
{
                int n;
                cout<<"Enters number to check: ";
cin>>n;
   return n;
}

void even()
{
                cout<<" the number is even";
  cin>>" ";
}

void odd()                      // Function definition
{
    cout<< "The number is odd";
    cin>>" ";
}
```

# Functions in C++

**D-** Passed arguments to function but no return values:

Example:

```
#include <iostream>

void check(int);
void even(int a);
void odd(int b);

void main()
{
   int num;
   cout<<"Enters number to check: ";

   cin >> num;
                check(num);
}
```

---------- continue ----------

# Functions in C++

```
void check(int a)
{
   if (a%2 == 0)
                even(a);
   else
                odd(a);
}

void even(a)
{
  cout<<" the number"<< a <<" is even";
  cin>>" ";
}

void odd(b)
{
     cout<< "The number"<< b <<" is odd";
     cin>>" ";
}
```

# C++ Function Overloading

Overloaded functions are a number of functions (two or more) having the same name but different types and/or the number of arguments.

*Example*

```
#include <iostream>
int operatoin (int a, int b)
                { return (a+b); }
float operation (float a, float b)
                { return (a/b); }
int main ()
{
 int x=3,y=6;
 float n=8.0,m=2.0;
cout << operation (x,y);
cout << "\n";
cout << operation (n,m);
cout << "\n"; return 0;
}
```

# User-defined data types

- We have three types of user-defined data types in C++:

  1- struct

  2. union

  3. enum

# Structure(struct) data structure

It is a collection of different data elements (members) gathered under a single name and it declares (creates) as follows:

```
struct stedent                  //std is the struct name
{
        char Name[20];    // struct members
        int Age;                        // or fileds

}; student std;                         //std is the variable
                                        //name of student
```

# Structure(struct) data structure

- Assigning values to struct members:

  cin>>struct_name.member;

- Displaying struct member values:

  cout<<struct_name.member;

# String in C++

- A string is defined in C and C++ as a character array ended by a null (“ “, ‘\0’).

- The world HELLO can be represented in an array as follows

| str | H | E | L | L | O | \0 |
|---|---|---|---|---|---|---|

- Declaration : char str[10]

# String in C++

A number of useful functions can be used to manipulate the string:

- **strcpy**(to_string, from_string): used to copy strings.
- **strcat**(string_1, string_2): used to concatenate two strings.
- **strlen**(str): used to measure the length of a string.
- **strcmp**(string_1, string_2): used to compare two strings.