AMBITION GURU

Object-Oriented programming in C++ (CSIT 202)

II semester, BScCSIT

Extra Notes for Introduction to OOP in C++

Compiled by Ankit Bhattarai
Ambition Guru College

Compiled by ab

AMBITION GURU

Tokens, Keywords and Identifiers, Variable declaration, the const Qualifier, endl, datatypes

Tokens

Tokens are the smallest units in a C++ program. The compiler breaks a program into these tokens while parsing.

Types include:

- **Keywords**
- **Identifiers**
- **Constants**
- **Operators**
- **Punctuation/Symbols**

Example C++ program

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

int main() {
      int age = 25;          // 'int' is a keyword, 'age' is an identifier
     float height = 5.9;      // 'float' is a keyword, 'height' is an identifier

    const float PI = 3.14159; // 'const' is a keyword, PI is a constant identifier

     cout << "Age: " << age << endl;
     cout << "Height: " << height << endl;
     cout << "PI (constant): " << PI << endl;

     return 0; // 'return' is a keyword
}
```

Output:
Age: 25
Height: 5.9
PI (constant): 3.14159

Keywords

Keywords are reserved words in C++ with special meaning, like int, return, if, while, etc. You cannot use them as variable names.

asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
catch	float	public	try
char	for	register	typedef
class	friend	return	union
const	goto	short	unsigned
continue	if	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while

Identifiers

identifiers are the names you assign to variables, functions, arrays, classes, etc. To ensure the compiler recognizes them correctly, you must follow specific **rules**:

Rules for Defining Identifiers in C++

- Can only contain : Letters (A–Z or a–z), Digits (0–9), Underscore (_)
- Must begin with a letter or underscore (_). *Example: total, _value, sum1 are valid, 1sum is invalid*
- Cannot be a C++ keyword. *Example: int, while, class cannot be used as identifiers.*
- Case-sensitive. *Example: score, Score, and SCORE are treated as different identifiers*
- No special characters or spaces. *Example: Invalid: user-name, total$, first name*

Variable Declaration

Declaring a variable means specifying its name and type so the compiler can reserve memory for it.

Syntax: `data_type variable_name;`

Example: `int count;`

const Qualifier

The const qualifier is used to declare a constant value that cannot be changed after initialization.

Example:
```
const float PI = 3.14;
```

endl

In C++, endl is a stream manipulator used with cout to:

- Insert a newline character (\n)
- Flush the output buffer

Example:

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

int main() {
    cout << "Line 1" << endl;
    cout << "Line 2" << endl;
    return 0;
}
```

Output:
Line 1
Line 2

Note:

Datatype in C++

Datatype in C++

AMBITION GURU

1. int (Integer)

Use: Stores whole numbers (no decimal point).

Size: Typically 4 bytes.

Range: -2,147,483,648 to 2,147,483,647 (on 32-bit systems)

Example: `int age = 25;`

2. char (Character)

Use: Stores a single character.

Size: 1 byte.

Stored as: ASCII value internally.

Example: `char grade = 'A';`

Datatype in C++

AMBITION GURU

3. float (Floating-point)

Use: Stores decimal numbers (single precision).

Size: 4 bytes.

Precision: Up to 6-7 digits after the decimal.

Example: `float temperature = 36.6;`

4. double (Double floating-point)

Use: Stores decimal numbers with more precision than float.

Size: 8 bytes.

Precision: Up to 15 digits.

Example: `double pi = 3.14159265359;`

Datatype in C++

AMBITION GURU

5. string

Use: Stores a sequence of characters (text).

Note: It's part of the std namespace, so you need to #include <string>.

Example: `string name = "Ambition";`

6. bool (Boolean)

Use: Stores truth values: true or false.

Size: Typically 1 byte.

Used in: Conditions, loops, flags, etc.

Example: `bool isPassed = true;`

AMBITION GURU

Datatype in C++ Example:

```
#include <iostream>
#include <string> // Required for using string type
using namespace std;
int main() {
    int age = 25;                // Integer data type
    float height = 5.9;          // Float data type
    char grade = 'A';            // Char data type
    string name = "Ambition";       // String data type
    bool isStudent = true;       // Bool data type: true or false

    cout << "Name: " << name << endl;
    cout << "Age: " << age << endl;
    cout << "Height: " << height << endl;
    cout << "Grade: " << grade << endl;
    cout << "Is Student: " << isStudent << endl;
    return 0;
}
```

Set Precision & Manipulators in C++

In C++, manipulators from the <iomanip> header help control the formatting of output, especially for floating-point numbers. One of the most used manipulators is setprecision.

Set Precision & Manipulators in C++

AMBITION GURU

1. setprecision(n):

Sets the total number of significant digits (not just after the decimal point).

```
#include <iostream>
#include <iomanip> // required for setprecision
using namespace std;

int main() {
    float pi = 3.14159265;

    cout << "Default: " << pi << endl;
    cout << "setprecision(2): " << setprecision(2) << pi << endl;
    cout << "setprecision(5): " << setprecision(5) << pi << endl;

    return 0;
}
```

Set Precision & Manipulators in C++

AMBITION GURU

2. fixed: Forces the number to be displayed in fixed-point notation — it makes setprecision(n) count digits after the decimal point.

Code:

```
cout << fixed << setprecision(2) << pi;
// Output: 3.14
```

3. scientific: Displays the number in scientific (exponential) format.

Code:

```
cout << scientific << pi;
// Output: 3.141593e+00
```

Set Precision & Manipulators in C++

AMBITION GURU

4. setw(n): Sets the width (in characters) of the next output field.

Code:

```
cout << setw(10) << 123;
// Output: '       123' (padded with spaces)
```

5. left, right: Used to justify output.

Code:

```
cout << left << setw(10) << 123;
cout << right << setw(10) << 123;
```

Operators in C++,The scope resolution operator, The new & delete Operations., Implicit Conversion, Control Structure in C++.

Operators in C++

AMBITION GURU

- **Operators** in C++ are special symbols used to perform operations on variables and values. C++ supports a wide range of operators grouped into different types.

1. Arithmetic Operators: These operators perform mathematical calculations: + (Addition), - (Subtraction), * (Multiplication), / (Division), and % (Modulus - remainder of division).

2. Assignment Operators: These operators assign values to variables:

- = (Assignment)
- += (Add and assign)
- -= (Subtract and assign)
- *= (Multiply and assign)
- /= (Divide and assign)
- %= (Modulus and assign)

Operators in C++

AMBITION GURU

3. **Comparison Operators:** These operators compare two values and return a boolean result:

- == (Equal to)
- != (Not equal to)
- > (Greater than)
- < (Less than)
- >= (Greater than or equal to)
- <= (Less than or equal to)

4. Logical Operators: These operators combine or modify boolean expressions:

- && (Logical AND)
- || (Logical OR)
- ! (Logical NOT)

Operators in C++

AMBITION GURU

5. Bitwise Operators: These operators perform operations on the individual bits of integers:

& (Bitwise AND), | (Bitwise OR), ^ (Bitwise XOR), ~ (Bitwise NOT), << (Left shift), and >> (Right shift).

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

int main() {
    int a = 6, b = 4;

    // Binary AND operator
    cout << "a & b is " << (a & b) << endl;

    // Binary OR operator
    cout << "a | b is " << (a | b) << endl;

    // Binary XOR operator
    cout << "a ^ b is " << (a ^ b) << endl;

    // Left Shift operator
    cout << "a<<1 is " << (a << 1) << endl;

    // Right Shift operator
    cout << "a>>1 is " << (a >> 1) << endl;

    // One's Complement operator
    cout << "~(a) is " << ~(a);

    return 0;
}
```

Output:
a & b is 4
a | b is 6
a ^ b is 2
a<<1 is 12
a>>1 is 3
~(a) is -7

Operators in C++

AMBITION GURU

6. Unary Operators: These operators act on a single operand:

- ++ (Increment)
- -- (Decrement)
- + (Unary plus)
- - (Unary minus)
- ! (Logical NOT)
- ~ (Bitwise NOT)

7. Other Operators:

- sizeof (Returns the size of a variable or type)
- ? : (Ternary operator - conditional expression)
- :: (Scope resolution operator)
- . (Member access operator)
- -> (Pointer member access operator)

New & delete Operations

New & delete Operations

AMBITION GURU

In C++, new and delete are used for dynamic memory allocation and deallocation at runtime. They replace traditional malloc() and free() from C with safer and type-aware mechanisms.

1. new Operator:

- Allocates memory on the heap.
- Returns a pointer to the allocated memory.
- Automatically calls constructor (for objects).

New & delete Operations

AMBITION GURU

Syntax:

```
pointer = new dataType;
pointer = new dataType(value);      // with initialization
pointer = new dataType[size];       // array
```

Example:

```
int* ptr = new int;         // allocates memory for 1 int
*ptr = 10;
cout << *ptr << endl;       // output: 10
```

New & delete Operations

AMBITION GURU

2. delete Operator

- Frees memory allocated by new.
- Prevents memory leaks.
- Automatically calls destructor (for objects).

Syntax:

```
delete pointer;
delete[] pointer;
```

Example:

```
delete ptr;
delete[] arr;
```

New & delete Operations

Example:

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

int main() {

    int *num = new int(25);

    cout << "Value: " << *num << endl;

    // Freeing memory
    delete num;
    return 0;
}
```

Output:
Value: 25

Type Conversion

AMBITION GURU

Type Conversion

- Type conversion is the process that converts the predefined data type of one variable into an appropriate data type.
- The main idea behind type conversion is to convert two different data type variables into a single data type to solve mathematical and logical expressions easily without any data loss.
- For example, we are adding two numbers, where one variable is of int type and another of float type; we need to convert or typecast the int variable into a float to make them both float data types to add them.

There are two types of type conversion:

1. Implicit Type Conversion
2. Explicit Type Conversion

Implicit Type Conversion

- The type conversion that is done automatically by the compiler is known as implicit type conversion.
- This type of conversion is also known as automatic conversion.
- When two operands of different types are encountered in the same expression, the lower type variable is converted to the type of the higher type variable by the compiler automatically.
- This is also called type promotion.

- The order of types is given below:

Data Type	Order
long double	(highest)
double	
float	
long	
int	
char	(lowest)

AMBITION GURU

// An example of implicit conversion

```
#include <iostream>
using namespace std;
int main()
{
    int x = 10;
    char y = 'a';
    float z;
    // y implicitly converted to int.
    //ASCII value of 'a' is 97
    x = x + y;
    // x is implicitly converted to float
    z = x + 1.5;
```

```
cout << "x = " << x << endl
        << "y = " << y << endl
        << "z = " << z << endl;
    return 0;
}
```

Output:

x = 107

y = a

Z = 108.5

AMBITION GURU

Explicit Type Conversion

- Conversions that require ***user intervention*** to change the data type of one variable to another, is called the **explicit type conversion**.
- In other words, an explicit conversion allows the programmer to manually change or typecast the data type from one variable to another type. Hence, it is also known as typecasting.
- Generally, we force the explicit type conversion to convert data from one type to another because it does not follow the implicit conversion rule.
- C++ permits explicit type conversion of variables or expressions as follows:

(type-name) expression //C notation

type-name (expression) //C++ notation

Example:


```
#include <iostream>
using namespace std;
int main() {
    double num_double = 5.63;
    int num_int1,num_int2;
    cout << "num_double = " << num_double << endl;

    // C-style conversion from double to int
    num_int1 = (int)num_double;
    cout << "num_int1   = " << num_int1 << endl;
```

```
    // function-style conversion from double to int
    num_int2 = int(num_double);
    cout << "num_int2   = " << num_int2 << endl;
    return 0;
}
```

Output:

num_double = 5.63

num_int1 = 5

num_int2 = 5

Control Structure in C++

Control Statements

1. Sequential structure (straight line)
2. Selection structure (branching or decision)
3. Loop structure (iteration or repetition)

Sequence structure (straight line)

- It is the default mode i.e., sequential execution of code statements (one line after another from top to bottom).

```
statement 1;
statement 2;
statement 3;
……..
statement n;
```

```
#include<iostream>
using namespace std;
int main()
{
    int num1,num2,sum;
    cout<<"Enter two numbers:\n";
    cin>>num1>>num2;
    sum=num1+num2;
    cout<<"Sum is:"<<sum;
    return 0;
}
```

```
Enter two numbers:
3
4
Sum is:7
```

AMBITION GURU

Conditional Structure

- Conditional statements help us to make a decision based on certain conditions.
- These conditions are specified by a set of conditional statements having Boolean expressions which are evaluated to a Boolean value true or false.
- There are following types of conditional statements.

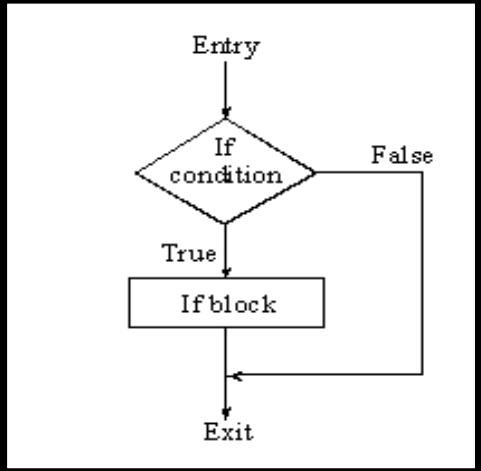
1. If statement
2. If-else statement
3. Nested if-else statement
4. If-else-if ladder
5. Switch statement

If statement

- The single if statement in C++ language is used to execute the code if a condition is true. It is also called one-way selection statement.

Syntax:

```
if(condition)
{
        statement(s);
}
```



```
#include<iostream>
using namespace std;
int main()
{
    int num;
    cout<<"Enter a number:\n";
    cin>>num;
    if(num%2==0)
    {
        cout<<num<<" is even";
    }
    return 0;
}
```

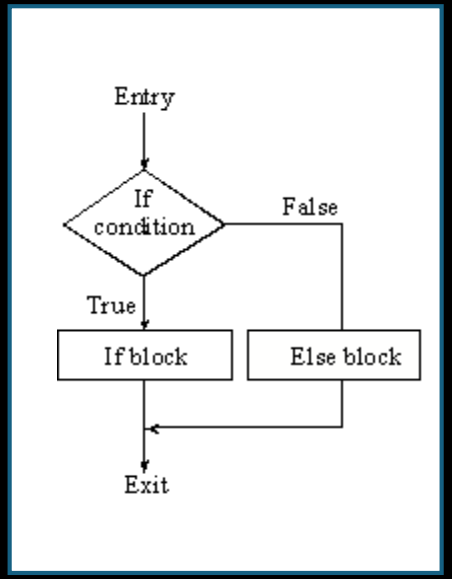
```
Enter a number:
10
10 is even
```

If...else statement

- The if...else statement in C++ language is used to execute the code if condition is true or false.
- It is also called two-way selection statement.

Syntax:

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



```
#include<iostream>
using namespace std;
int main(){
    int num;
    cout<<"Enter a number:\n";
    cin>>num;
    if(num%2==0){
        cout<<num<<" is even";
    }
    else{
        cout<<num<<" is odd";
    }
    return 0;
}
```

```
Enter a number:
5
5 is odd
```

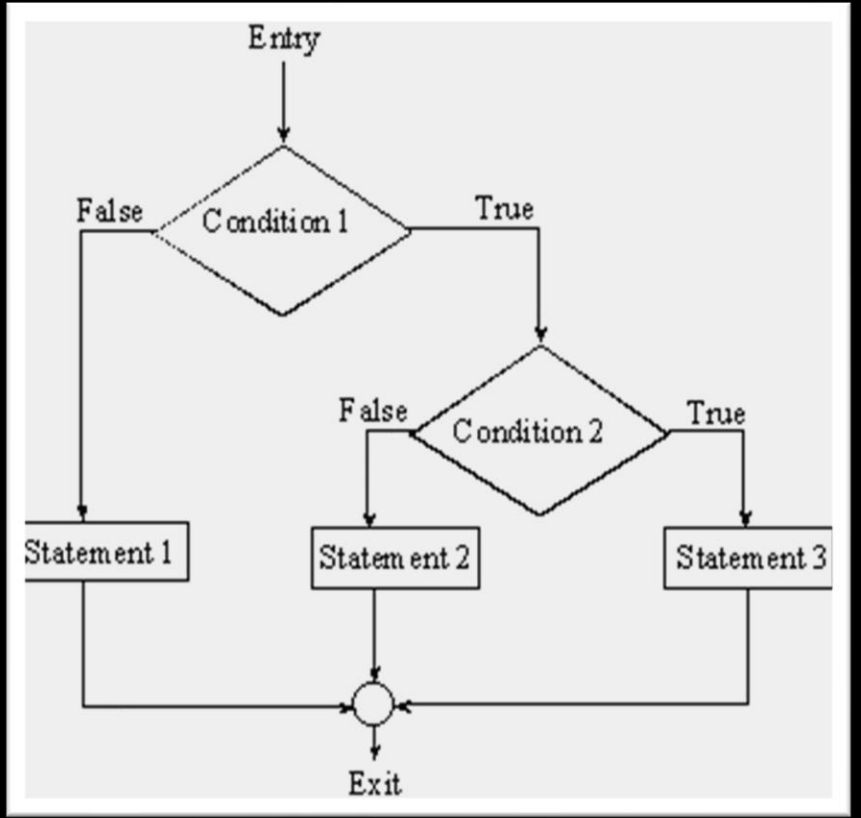
Nested if...else statement

- The nested if...else statement is used when a program requires more than one test expression.
- It is also called a multi-way selection statement. When a series of the decision are involved in a statement, we use if else statement in nested form.

AMBITION GURU

Syntax:

```
if(condition1 )
{
    if(condition2)
     {
        statement1;
     }
    else
      {
        statement 2;
      }
}
else
{
    statement 3;
}
```



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

int main() {
    int a, b, c;
    cout << "Enter 3 numbers:\n";
    cin >> a >> b >> c;

    if (a > b) {
        if (a > c) {
            cout << a << " is greatest";
        } else {
            cout << c << " is greatest";
        }
    } else {
        if (b > c) {
            cout << b << " is greatest";
        } else {
            cout << c << " is greatest";
        }
    }

    return 0;
}
```


AMBITION GURU

Enter 3 number:
3 5 2
5 is greatest

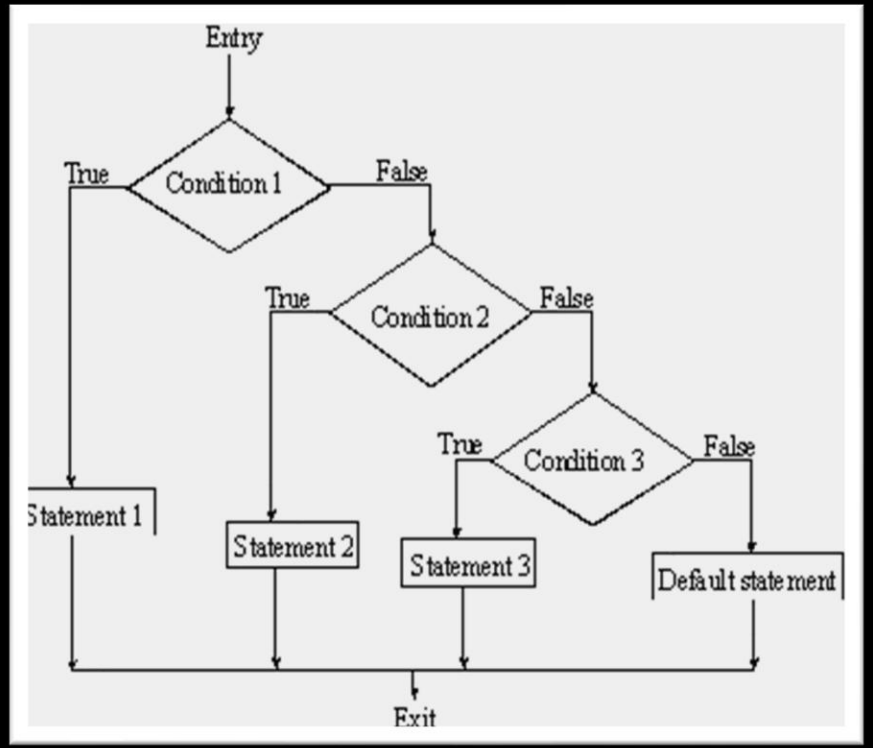
if...else if ladder

- The if...else if statement is used to execute one code from multiple conditions.
- It is also called multipath decision statement.
- It is a chain of if..else statements in which each if statement is associated with else if statement and last would be an else statement.

Syntax:


```
if(condition1)
        statement1;
    else if (condition2)
        statement2;
           else if (condition3)
               statement3;
              .
              .
              .
           else

default_statement;
```



Switch statement

- C++ has a built-in multiway decision statement known as "switch" which tests the value of a given variable (or expression) against a list of case values, and when a match is found, a block of statements associated with that case is executed.

Syntax:


```
switch (expression){
        case value-1:
                block-1;
            break;
        case value-2:
            block-2;
            break;
        .
        .
        .
        default:
            default-block;
}
```

```
#include<iostream>
using namespace std;
int main( )
{
    int n;
    cout<<"Enter any no. from 1 to 7:\n";
    cin>>n;
    switch (n)
    {
        case 1:
        cout<<"Sunday";
        break;
       case 2:
        cout<<"Monday";
        break;
        case 3:
        cout<<"Tuesday";
        break;
        case 4:
        cout<<"Wednesday";
        break;
         case 5:
        cout<<"Thursday";
        break;
```

```
    case 6:
          cout<<"Friday";
          break;
    case 7:
               cout<<"Saturday";
        break;
    default:
               cout<<"Enter valid number from 1 to 7";
}
return 0;
}
```

```
Enter any no. from 1 to 7:
5
Thursday
```

AMBITION GURU

Repetitive Structure

- *Repetitive structures*, or *loops* are used when a program needs to repeatedly process one or more instructions until some condition is met, at which time the loop ends.
- The process of performing the same task over and over again is called *iteration*, and C++ provides built-in iteration functionality.
- A loop executes the same section of program code over and over again, as long as a loop condition of some sort is met with each iteration.
- This section of code can be a single statement or a block of statements (a compound statement).

AMBITION GURU

- There are three looping structures in C++:

1. for loop
2. do.... While
3. while Loop

for loop

- It is a count controlled loop which is used when some statement(s) is to be repeated certain number of times.

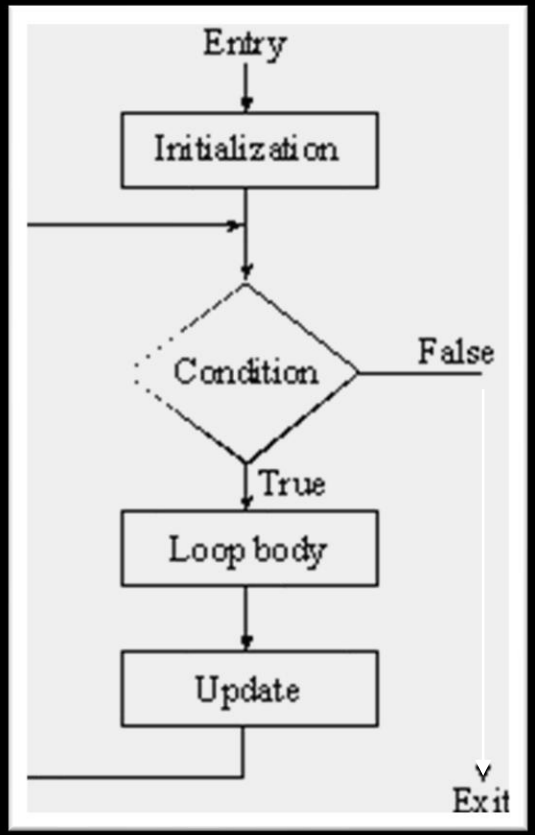
Syntax:

```
for (initialization ; condition ; update)
{
    body of the loop; //statement(s) to be executed repeatedly
}
```

AMBITION GURU

How for loop works?

- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of for loop are executed, and the update expression is updated.
- Again the test expression is evaluated.
- This process goes on until the test expression is false. When the test expression is false, the loop terminates.



```
#include<iostream>
using namespace std;
int main()
{
    int i;
    for (i = 1; i <=10; i++)
    {
        cout<<i<<"\t";
    }
    return 0;
}
```

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

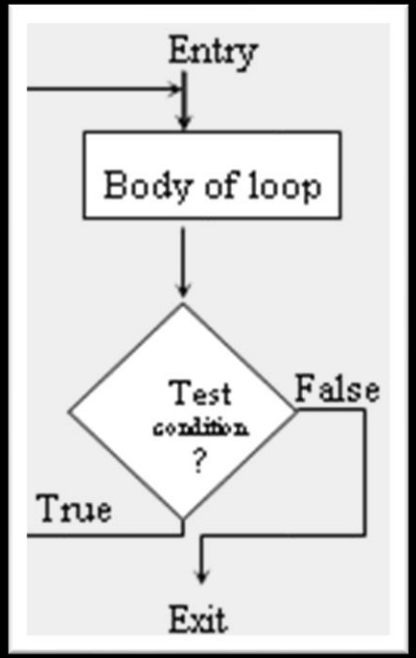
do ..... while loop

- “do-while loop” is exit-controlled loop(condition controlled) in which test condition is evaluated at the end of loop.
- The body of the loop executes at least once without depending on the test condition and continues until the condition is true.

Syntax:

```
do
{
   body of the loop;
} while (test condition);
```

Flowchart



```
#include<iostream>
using namespace std;
int main()
{
    int i;
    i = 1;
    do
    {
        cout<<i<<"\t";
        i++;
   } while (i <=10);
    return 0;
}
```

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

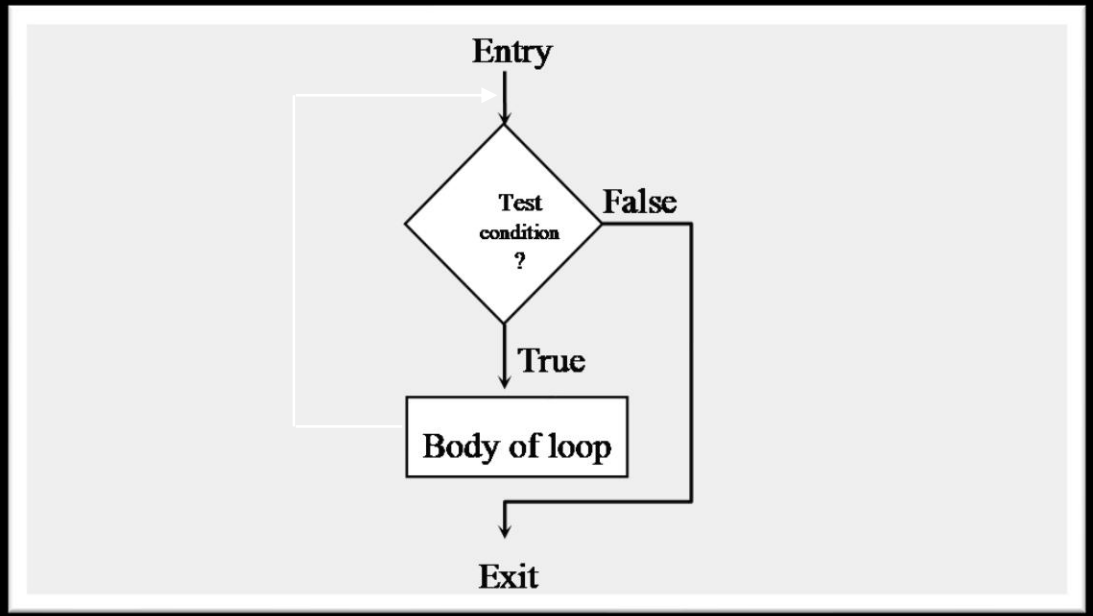
while loop

- “while loop” is entry-controlled loop (condition controlled) in which test condition is evaluated at the beginning of the loop execution.
- If the test condition is true, only then the body of the loop executes, or else it does not.

Syntax:

```
while (test condition)
{
    body of the loop;
}
```

Flowchart:



```
#include<iostream>
using namespace std;
int main()
{
    int i;
    i = 1;
    while(i<=10)
    {
        cout<<i<<"\t";
        i++;
    }
    return 0;
}
```

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

Functions: Simple Functions, Function declaration, calling the function, function definition, passing argument to, returning value from function, passing constants, Variables, pass by value, passing structure variables, pass by reference, Default arguments, return statements, return by reference, overloaded functions; Different number of arguments, Different Kinds of argument, Static Data Member and Static Function, inline function

Function

- A **function** is a group of statements that together perform a task.
- Functions are made for code **reusability** and for saving **time** and **space**.

Library Function	User defined function
Predefined	Created by user
Declarations inside header files	Reduced complexity of program
Body inside .dll files	

Difference between library and user defined function.

Categories of User-defined Functions

AMBITION GURU

- Function with no argument and no return value.
- Function with no argument but return value.
- Function with argument but no return value.
- Function with argument and return value.

The main function

AMBITION GURU

- main() function is the **entry point** at which execution of program is started.
- C allows main() function type to be void. **C++ does not all main() function to be void.**
- In C++, main() returns an integer type value to the operating system. Therefore, every main() in C++ should end with a **return(0)** statement.
- The explicit use of a return(O) statement will indicate that the **program was successfully executed.**

```
int main()
{
    .........
    .........
    return 0;
}
```

Function Prototyping / Declaration

AMBITION GURU

- Describes number and type of arguments and the type of return values.
- When a function is called, the compiler ensure that proper arguments are passed, and the return value is treated correctly.
- Any violation in matching the arguments or the return types will be caught by the compiler at the time of compilation itself.
- These checks and controls did not exist in the conventional C functions.

```
void f();

int main()
{
    f(5);
}

void f()
{

}
```

Simple Function

AMBITION GURU

Function Prototyping / declaration

Syntax:
```
return-type function-name (argument-list);
```

Example:
```
float volume(int,float,float);  OR  float volume(int x,float y,float z);
```

Function Calling:
```
float volume(int x,float y,z); //illegal
```

Function Definition:
```
float cube=volume(b1,w1,h1); //Actual Parameter
```

```
float volume(int a,float b,float c)
{
    float v=a*b*c;
    .....
    .....
}
```

Simple Function

AMBITION GURU

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

int add(int, int);  //Function prototype(declaration)

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

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

```
Enters two numbers to add:
8
-4
Sum = 4
```

Function Overloading

- More than one functions having same name but differs either in number of arguments or type of arguments or both is said to be function overloading.
- Function overloading shows the compile time polymorphism because the particular function call and its associated function definition is known by the compiler before the running of the program i.e.at the compile time only.

Advantages of function overloading:

- Remembering names is easier
- Lesser complexity
- Increases readability

WAP to find the volume of a cube, cuboid and cylinder using the concept of functionoverloading.

AMBITION GURU

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

void volume(float l)
{
   cout<<"The volume of the cube is:"<<l*l*l<<endl;
}

void volume(float l,float b,float h)
{
   cout<<"The volume of the cuboid is:"<<l*b*h<<endl;
}

void volume(float r, float h)
{
   cout<<"The volume of the cylinder is:"<<3.14*r*r*h;
}
```

```
int main( )
{
    volume(3.5);
    volume(3.5,6.5,9.5);
    volume(3.5,5.5);
    return 0;
}
```

Output:

```
The volume of the cube is:42.875
The volume of the cuboid is:216.125
The volume of the cylinder is:211.558
```

WAP to find the cube of a integer , float and double number using the concept of function overloading(passing single argument to the function)

WAP to find the cube of a integer , float and double number using the concept of function overloading(passing single argument to the function)

AMBITION GURU

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

void findcube(int a){
   cout<<"The cube of the integer number is:"<<a*a*a<<endl;
}

void findcube(float b)
{
   cout<<"The cube of the float number is:"<<b*b*b<<endl;
}

void findcube(double c)
{
   cout<<"The cube of the double number is:"<<c*c*c;
}
```

```
int main( )
{
   int x;
   float y;
   double z;
   cout<<"Enter values for int, float double type variables respectively:";
   cin>>x>>y>>z;
   findcube(x);
   findcube(y);
   findcube(z);
   return 0;
}
```

Output:

```
Enter values for int, float double type variables respectively:2
3.5
4.5e+7
The cube of the integer number is:8
The cube of the float number is:42.875
The cube of the double number is:9.1125e+22
```

Inline functions

AMBITION GURU

- To eliminate the cost of calls to small functions, C++ proposes a new feature called inline function.
- An **inline function** is a function that is expanded in line when it is invoked. That is, the compiler replaces the function call with the corresponding function code (something similar to macros expansion).
- **Syntax:**

```
inline return-type function-name(parameters)

{
    //function code
}
```

Inline functions

AMBITION GURU

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

inline double cube(double s)
{
    return s*s*s;
}

int main()
{
    cout << "The cube of 3 is: " << cube(3.0) << "\n";
    cout << "The cube of 4 is: " << cube(2.5+1.5) << "\n";
    return 0;
}
```

```
The cube of 3 is: 27
The cube of 4 is: 64
```

Inline functions

AMBITION GURU

NOTE:

- All inline functions must be **defined** before they are called.
- To make a function inline, prefix the keyword **inline** to the function definition.
- Usually, functions are made inline when they are **small** enough to be defined in one or two lines.
- If the function grows in size, the speed benefit of inline function diminish.

Inline functions

AMBITION GURU

- Remember, inline is only a **request** to the compiler, not a command.
- Compiler can **ignore the request** for inline if function definition is too long or too complicated and compile the function as a normal function.

Some situations where inline expansion may not work:

- If a function contains a loop. (for, while, do-while)
- If a function contains static variables.
- If a function is recursive.
- If a function return type is other than void, and the return statement doesn't exist in function body.
- If a function contains switch or goto statement.

Inline functions

AMBITION GURU

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

inline float mul(float x, float y)
{
    return (x*y);
}

inline double div(double p, double q)
{
    return (p/q);
}

int main()
{
    float a=12.345;
    float b=9.82;

    cout<<mul(a,b)<<"\n";
    cout<<div(a,b)<<"\n";

    return 0;
}
```

```
121.228
1.25713
```

AMBITION GURU

THANK YOU
Any Queries ?