Programming Basics

Marwadi University

Department of Diploma Engineering

UNIT 2

Object Oriented Programming with C++ – 09CE2301

Prof. Meghnesh Jayswal

Objectives

- Introduction to c++
- Structure of c++
- I/O operators
- Tokens
- Types of operators
- Library function
- Directives
- Manipulators
- Storage classes
- Scope resolution
- Type conversion
- Control structure

Introduction to C++

- C++ is **Object Oriented Programming language.**
- C++ was developed by **Bjarne stroustrup in 1983** at Bell labs.
- C++ is an extended(superset) of C programming.
- C++ adds features like Classes, Inheritance , function overloading and operator overloading.

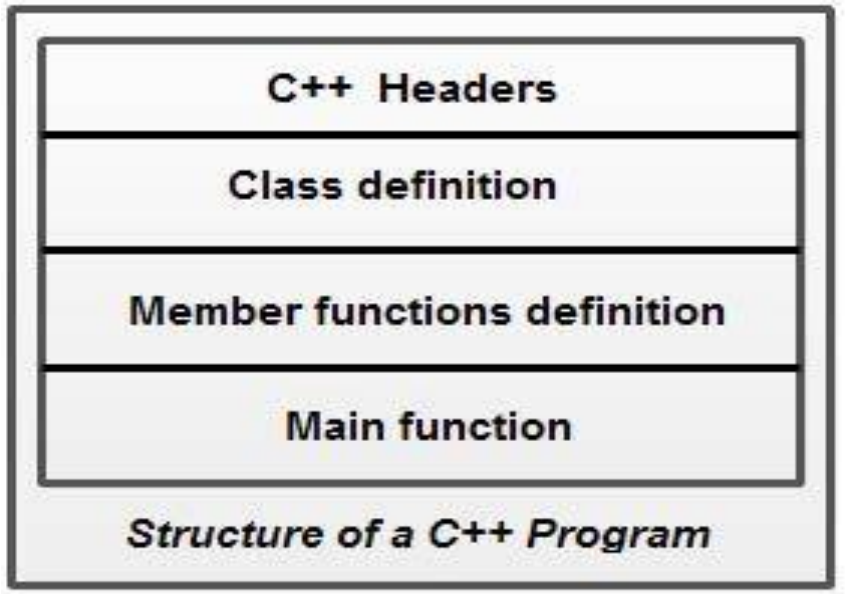
Introduction to C++

- **C++ is not purely object oriented** because
  - you can write code without creating a class in C++.
  - You can declare global variable. ➔ Partial Encapsulation
  - Use of friend function. ➔ can access private and protected members of other class
- **C++ is truly object oriented.**

Structure of C++ Program

Better to organize a program file into three separate files.

- Class declarations in a header file.
- Definition of member function in a separate file.
- The main program that uses the class is placed in a third file which includes the previous two files.



Structure of a C++ Program

A Simple C++ Program

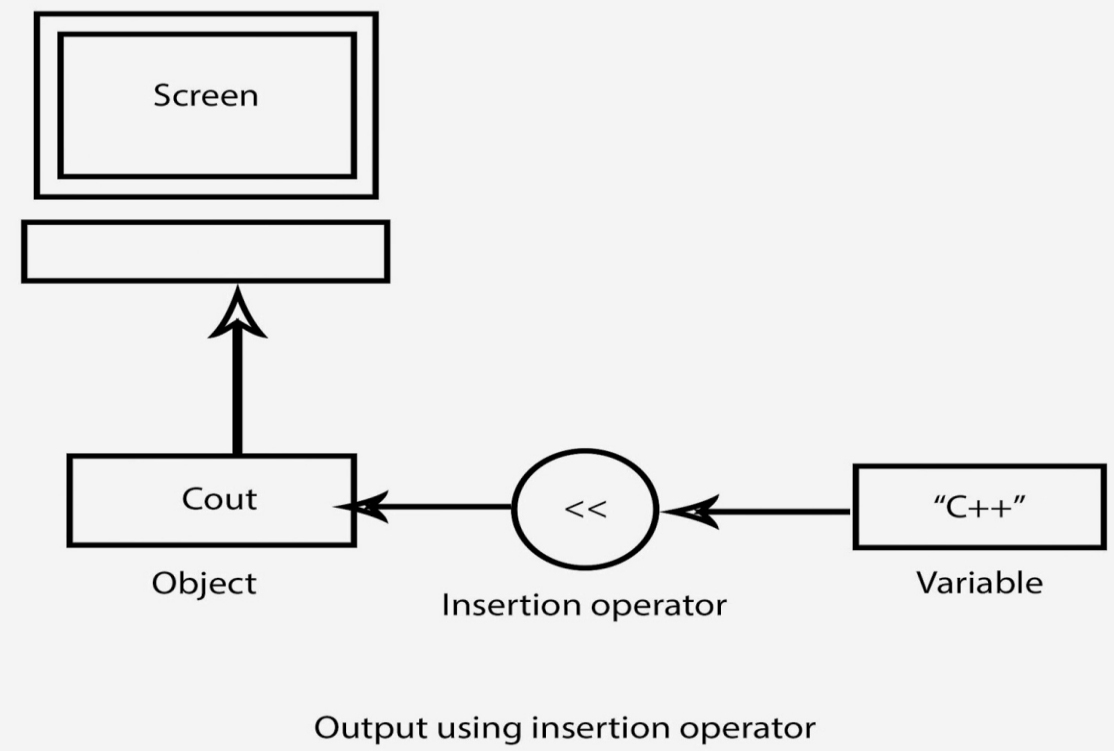
```
/*
 File: main.cpp
 About first c++ program
*/

#include <iostream.h>
#include<conio.h>
void main()
{
 cout<<"This is my first C++ Program.";
getch();
}
```

Input Output Operators in C++

- The operator used for taking the input is known as the **extraction** or **get from operator** (>>), while the operator used for displaying the output is known as the **insertion** or **put to operator** (<<).

Output Operator with cout

- To send output to the screen we use the insertion operator on the object cout
- Format: cout << *Expression*;



Output using insertion operator

Output Operator with Cout

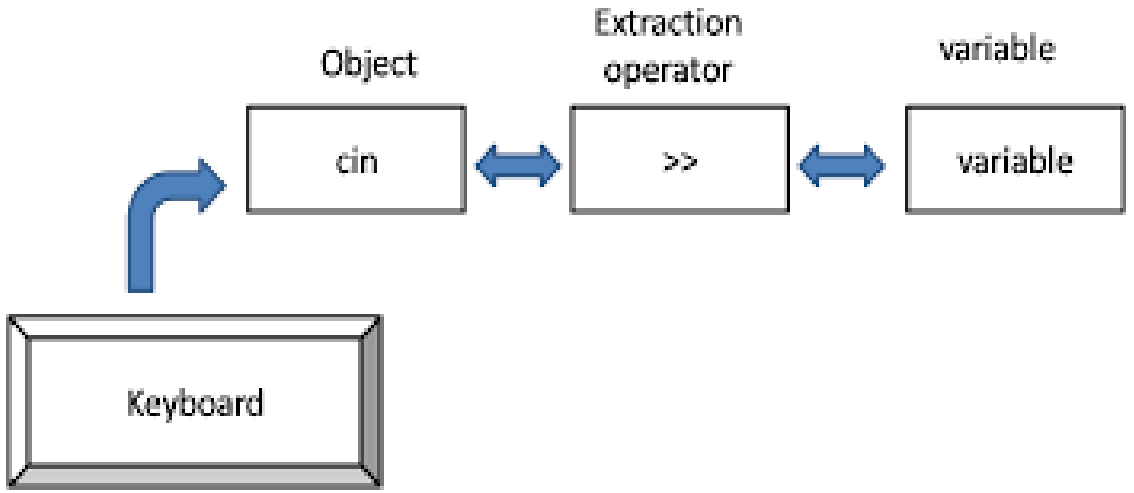
- Example:

```
cout << 5;  // Outputs 5
cout << 4.1; // Outputs 4.1
cout << “String”; // Outputs String
```

```
#include<iostream.h>
#include<conio.h>
void main()
{
        int a=10;
        cout<<a;
getch();
}
```

Input Operator with CIN

- To get input from the keyboard we use the extraction operator and the object cin
- Format: cin >> Variable;



Input Operator with CIN

Example:

```
int X;
float Y;
cin >> X; // Reads in an integer
cin >> Y; // Reads in a float
```

```
#include<iostream.h>
#include<conio.h>
void main ()
{       int a;
        cin>>a;
        a = a+1;
getch();
}
```

Cascading Input/Output Operator

- The cascading(Relation between both operator) of the input and output operators refers to the consecutive occurrence of input or output operators in a single statement.

Cascading Input / Output Operator

A program without cascading of the input/output operator.

```
#include<iostream.h>
#include<conio.h>
void main ()
{           int a, b;
        cin>>a;
        cin>>b;
        cout<<"The value of a is
        cout<<a;
        cout<<"The value of b is
        cout<<b;
getch();
}
```

Cascading Input / Output Operator

A program with cascading of the input/output operator

```
#include<iostream.h>
#include<conio.h>
void main ()
{
        int a, b;
        cin>>a >>b;
        cout<<"The value of b is : "<<b;
        //cout<<"The value of a is "<<a;

getch();
}
```

Tokens

- The smallest individual units in a program are known as tokens.
  - Keywords
  - Identifiers
  - Constants
  - Strings
  - Operators

Keywords

C++ Reserved Words				
asm	auto	bad_cast	bad_typeid	bool
break	case	catch	char	class
const	const_cast	continue	default	delete
do	double	dynamic_cast	else	enum
except	explicit	extern	false	finally
float	for	friend	goto	if
inline	int	long	mutable	namespace
new	operator	private	protected	public
register	reinterpret_cast	return	short	signed
sizeof	static	static_cast	unsigned	struct
switch	template	this	throw	true
try	type_info	typedef	typeid	typename
union	unsigned	using	virtual	void
volatile	wchar_t	while		

Identifiers

- A valid identifier is a sequence of one or more letters, digits or underscore characters (_).
- Neither spaces nor punctuation marks or symbols can be part of an identifier.
- Only letters, digits and single underscore characters are valid.
- In addition, variable identifiers always have to begin with a letter.
- They can also begin with an underline character (_).

Identifiers

- The name of a variable:
- Starts with an underscore “_” or a letter, lowercase or uppercase, such as a letter from a to z or from A to Z.

  Examples are Name, gender, _Students, pRice.
- Can include letters, underscore, or digits.

  Examples are: keyboard, Master, Junction, Player1, total_grade, _ScoreSide1.
- Cannot include special characters such as !, %,], or $.
- Cannot include an empty space.
- Cannot be any of the reserved words. (Example: PI)
- Should not be longer than 32 characters (although allowed).

Constants

- Two ways of creating symbolic constant in C++.
- Using the qualifier const
- Defining a set of integer constants using enum keyword.

- Any value declared as const can not be modified by the program in any way. In C++, we can use const in a constant expression.

```
const int size = 10;
Char name[size]; // This is illegal in C.
```

Constants

- const allows us to create typed constants.
- #define - to create constants that have no type information.
- The named constants are just like variables except that their values can not be changed. C++ requires a const to be initialized.
- A const in C++ defaults, it is local to the file where it is declared. To make it global the qualifier extern is used.

Operators

- **Unary Operators:** ++,--, sizeof
- **Binary Operators:**<<(left shift),>>(Right shift)
- **Arithmetic Operators:** +,-,*,/,%
- **Logical Operators:** && (AND), || (OR) , ! (NOT)
- **Relational Operators:** >,>=,<,<=,==
- **Bitwise Operators:** &, |, !
- **Ternary Operator:** ?:

Operators

- **New operators (All C operators are valid in C++)**
- << Insertion Operator
- >> Extraction Operator
- : : Scope Resolution Operator
- : : * Pointer-to-member declaration
- ->* Pointer-to-member Operator
- .* Pointer-to-member Operator
- delete Memory Release Operator
- endl Line Feed Operator
- new Memory Allocation Operator
- setw Field Width Operator

Memory Management Operators (New(), delete())

In C

- **malloc( )** and **calloc( )** functions are used to allocate memory dynamically at run time.
- The function **free( )** to free dynamically the allocated memory.

In C++

- The unary operators **new** and **delete** perform
- the task of allocating and freeing the memory.

Memory Management Operators (New(), delete())

- **new** to create an object
- **delete** to destroy an object

- A data object created inside a block with **new**, will remain in existence until it is explicitly destroyed by using **delete**.
-
- Thus the life time of an object is directly under our control and is unrelated to the block structure of the program.

Memory Management Operators (New(), delete())

- **new** to create an object
- Syntax: pointer-variable = **new** data-type
- pointer-variable is a pointer of type data-type

- The **new** operator allocates sufficient memory to hold a data object of type data-type and returns the address.
- The data type may any valid data type
- The pointer-variable holds the address of the memory space allocated

Memory Management Operators (New(), delete())

- pointer-variable = **new** data-type;

- p = new int; // p is a pointer of type int
- q = new float; // q is a pointer of type float
- Here p and q must have already been declared as pointers of appropriate types.

- Alternatively, we can combine the declaration of pointers and their assignments as:
- int *p = new int;
- float *q = new float;

Memory Management Operators (New(), delete())

- **new** can be used to create a memory space for any data type including user-defined types such as arrays, structures and classes.
- Syntax: **pointer-variable = new data-type [size];**

  int *p = new int [10];

- When creating multi-dimensional arrays with new, all the array sizes must be supplied.
- array_ptr = new int[3][5][4]; //legal
- array_ptr = new int[m][5][4]; //legal
- array_ptr = new int[3][5][ ]; //illegal
- array_ptr = new int[ ][5][4];//illegal

Memory Management Operators (New(), delete())

- **delete** to destroy an object
- Syntax: **delete pointer-variable;**
- When a data object is no longer needed, it is destroyed to release the memory space for reuse.
  - delete p;
  - delete q;
- **delete [size] pointer-variable;**
- The size specifies the number of elements in the array to be freed.
- delete [ ]p; // delete the entire array pointed to by **p**

Memory Management Operators (New(), delete())

Advantages of new over malloc( ):

- It automatically computes the size of the data object. No need to use sizeof operator.
- It automatically returns the correct pointer type. No need to use type cast.
- It is possible to initialize the object while creating the memory space.
- Like any operator, new and delete can be overloaded.

Library Functions

HEADER FILES	TYPES (Full Forms)
iostream.h	Include all input and output streams
conio.h	All console input and output functions
math.h	Include all Mathematical functions
stdlib.h	Include all standard library functions
string.h	All string manipulation functions
ctype.h	All character manipulating function
iomanip.h	Include all input and output manipulators

Directives

- Preprocessor Directives
  - Inclusion Directive
  - Macro Definition Directive
- using Directive

Inclusion Directive

- Pre-processor directives start with ‘#’ character.
- This category has only one directive, which is called **#include**.
- This inclusion directive is used to include files into the current file
- #include is used with two options
  - angular brackets (<>)
  - inverted commas (“”).

Inclusion Directive

- When #include is written with <> ➔ we are instructing to include the files from the include path defined in the compiler settings which is generally the include folder from the compiler. .
- When #include is written with “” ➔ we are instructing to include a file: first the current folder is checked and if it is not found there, then the include folder of the compiler is checked.

Inclusion Directive

#include <stdio.h>	includes stdio.h from include folder
#include <iostream>	includes cpp class library header iostream
#include<my.cpp>	includes my.cpp file from include folder
#include “my.h”	includes my.h file from current working folder
#include “myfolder/abc.h”	includes abc.h file from the myfolder which is available in current working folder

Macro Definition Directive

- Macros are piece of code in a program which is given some name. Whenever this name is encountered by the compiler, the compiler replaces the name with the actual piece of code.
- The '**#define**' directive is used to define a macro.

Macro Definition Directive

```
#include <iostream.h>
#include<conio.h>
#define PI 3.14
#define SQR(X) ((X)* (X))
#define CUBE(y) SQR(y)* (y)
void main()
{
int radius;
cout << endl << "Enter Radius of the Circle:";
cin >> radius;
cout << endl << "Area of the Circle is: " << PI * SQR(radius);
cout << endl << "Perimeter  of Circle is : "
      <<  2 * PI * radius;

cout << endl << "Area of the Sphere of Radius "
      << radius << " is: "
      << 4/3.0 * 3.14 *CUBE(radius);

getch();
}
```

using Directive

- A namespace is designed for providing a way to keep one set of names separate from another.
- The *using* keyword states that the program is using the names in the given namespace.

using Directive

- *using* namespace std;
  - All the program statement that follow are within the std namespace.
  - cout , cin declared within std namespace
- If we do not use ***using*** directive then we need to write each statements like
- **std :: cout << “Hello” ;**

Manipulators

- endl Manipulator
- setw Manipulator
- setfill Manipulator
- setprecision Manipulator

Endl Manipulator

- **endl** manipulator is used to Terminate a line and flushes the buffer.
- When writing output in C++,you can use either endl or “\n” to produce a newline, but each has a different effect.
- std::endl sends a newline character '\n' and flushes the output buffer.
- '\n' sends the newline character, but does not flush the output buffer.
- **NOTE:-**The buffer flush is used to transfer of computer data from one temporary storage area to computers permanent memory. If we change anything in some file, the changes we see on the screen are stored temporarily in a buffer.

Endl Manipulator

```
#include <iostream.h>

#include<conio.h>

void main()

{

        cout<<"USING '\\n' ...\n";

        cout<<"Line 1 \nLine 2 \nLine 3 \n";

        cout<<"USING end ..."<< endl;

        cout<< "Line 1" << endl << "Line 2" << endl << "Line 3" << endl;

getch();

}
```

- **Output:**

  USING '\n' ...

  Line 1

  Line 2

  Line 3

  USING end ...

  Line 1

  Line 2

  Line 3

setw and setfill Manipulator

- setw manipulator sets the width of the filed assigned for the output.
- The field width determines the minimum number of characters to be written in some output representations.
- setfill character is used in output insertion operations to fill spaces with characters when standard width of the representation is shorter than the field width.

setw and setfill Manipulator

Syntax

```
setw( [number_of_characters] );

setfill( [character] );
```

setw and setfill Manipulator

```
#include <iostream.h>
#include <iomanip.h>
    int main()
    {               cout<<"USING setw() .....\n";
                    cout<< setw(10) <<11<<"\n";
                    cout<< setw(10) <<2222<<"\n";
                    cout<< setw(10) <<33333<<"\n";
                    cout<< setw(10) <<4<<"\n";
cout<<"USING setw() & setfill() [type- I]...\n";
                    cout<< setfill('0');
                    cout<< setw(10) <<11<<"\n";
                    cout<< setw(10) <<2222<<"\n";
                    cout<< setw(10) <<33333<<"\n";
                    cout<< setw(10) <<4<<"\n";
cout<<"USING setw() & setfill() [type-II]...\n";
                    cout<< setfill('-')<< setw(10) <<11<<"\n";
                    cout<< setfill('*')<< setw(10) <<2222<<"\n";
                    cout<< setfill('@')<< setw(10) <<33333<<"\n";
                    cout<< setfill('#')<< setw(10) <<4<<"\n";
                    return 0;
    }
```

```
USING setw() ..............
        11
       2222
      33333
         4
USING setw() & setfill() [type- I]...
0000000011
0000002222
0000033333
0000000004
USING setw() & setfill() [type-II]...
--------11
******2222
@@@@@33333
ccccc#####
#########4
```

Setprecision manipulator

- setprecision manipulator sets the total number of digits to be displayed, when floating point numbers are printed.
- Syntax: setprecision([number_of_digits]);
- Example:

```
cout<<setprecision(5)<<1234.537;

// output will be : 1234.5
```

Storage Classes

- Storage class of a variable defines the **lifetime** and **visibility** of a variable. **Lifetime** means the duration till which the variable remains active and **visibility** defines in which module of the program the variable is accessible.
- There are four types of storage classes
  - Automatic
  - External
  - Static
  - Register

Automatic Storage Class

- *auto* keyword is used to declare automatic variables.
- variable is declared without any keyword inside a function, it is automatic by default.
- **Lifetime :** same as lifetime of the function. Once the execution of function is finished, the variable is destroyed.
- **Visibility :** only within the function it is declared

Automatic Storage Class

Declaration:

datatype var_name1;

or

auto datatype var_name1;

Example:

float y = 5.67;

auto int x;

External Storage Class

- *extern* keyword is used to declare external variables.
- variable is declared outside the given program of function (like : global variable)
- **Lifetime :** visible to all the functions present in the program.
- **Visibility :** throughout the program

External Storage Class

Declaration:

extern datatype var_name1;

Example:

extern float var1;

External Storage Class

File: sub.cpp

```
int test=100;

void multiply(int n)

{

    test=test*n;

}
```

File: main.cpp

```
#include<iostream>

#include "sub.cpp"  // includes the content of sub.cpp

using namespace std;

extern int test;  // declaring test


int main()

{

    cout<<test<<endl;

    multiply(5);

    cout<<test<<endl;

    return 0;

}
```

Static Storage Class

- Static storage class has the visibility of a local variable but lifetime of an external variable.
- It can be used only within the function where it is declared but destroyed only after the program execution has finished.
- When a function is called, the variable defined as static inside the function retains its previous value and operates on it.

Static Storage Class

Declaration:

static datatype var_name1;

Example:

static int x = 101;

static float sum;

Register Storage Class

- Register variable differs in the manner in which it is stored in the memory.
- **Automatic** variables are stored in **primary memory** ,but **register** variables are stored in **CPU registers**.
- The objective of storing a variable in register to increase its access speed, which makes the program run faster.
- If no registers vacant to accommodate variable, then it is stored like automatic variable.

Register Storage Class

Declaration:

register datatype var_name1;

Example:

register int id;

register char a;

All Storage Classes

```
#include<iostream>
using namespace std;
int g;    //global variable, initially holds 0
void test_function()
{
    static int s;    //static variable, initially holds 0
    register int r;    //register variable
    r=5;
    s=s+r*2;
    cout<<"Inside test_function"<<endl;
    cout<<"g = "<<g<<endl;
    cout<<"s = "<<s<<endl;
    cout<<"r = "<<r<<endl;
}
int main()
{
    int a;    //automatic variable
    g=25;
    a=17;
    test_function();
    cout<<"Inside main"<<endl;
    cout<<"a = "<<a<<endl;
    cout<<"g = "<<g<<endl;
    test_function();
    return 0;
}
```

All Storage Classes

Output:

Inside test_function

g = 25

s = 10

r = 5

Inside main

a = 17

g = 25

Inside test_function

g = 25

s = 20

r = 5

Scope Resolution Operator

- Scope resolution operator (::) in C++ programming language is used to define a function outside a class
- when we want to use a global variable but also has a local variable with the same name.

Scope Resolution Operator

```
#include <iostream>

using namespace std;

char c = 'a';     // global variable

int main() {

  char c = 'b';   //local variable

  cout << "Local  variable: " << c << "\n";

  cout << "Global variable: " << ::c << "\n";  //using scope resolution operator

  return 0;

}
```

Output:

```
Local Variable : b
Global Variable : a
```

Type Casting/Type Conversion

- C++ permit explicit type conversion of variables or expressions using the type cast operator.
- (type-name) expression // C notation
- type-name (expression) // C++ notation
  // like a function call
  // notation
- eg:- average = sum /(float) i; // C notation
  average = sum / float(i); // C++ notation

Type Casting/Type Conversion

- In c++, the concept of converting an integer type variable into a float type is known as type conversion. in example shows how the integer type variable is automatically converted to a float type:
- In example 1 the int type variable, a, is assigned a value of 10, which is automatically converted to 10.0 during the addition of the values of the variables b and a. In example 1, you notice the automatic conversion of the data type from int to float. However, if you want to convert the double type variable to an int type, you need to explicitly provide code for it. The concept of explicitly converting one data type variable to another is known as type casting. Example 2 shows how to type cast the double type variable to an int type:

Type Casting/Type Conversion

Example 1: Converting an Integer Type Variable to a Float Type

```
#include <iostream.h>
int main()
{
int a = 10;
float b = 22.7;
b = b + a;
cout << "The value of the b variable is:" <<b;
return 0;
}
```

Output:

```
The value of the b variable is:32.700001
```

Type Casting/Type Conversion

Example 2: Type Casting the Double Type Variable to an int Type

```
#include <iostream.h>

int main()

{       double a = 2134.56;
        int b;
        b = int(a);
        cout << "The value of the b variable is:" <<b;

        return 0;

}
```

Output:

The value of the b variable is:2134

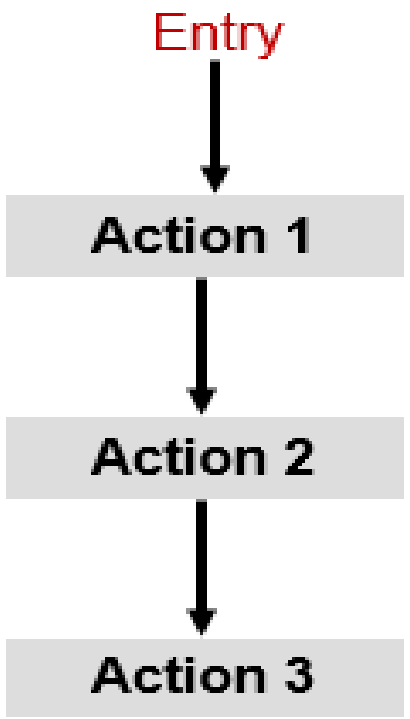
In Example 2, a double type variable is type cast to the int type variable, b.

Expressions & Control Structures

- Expressions and Their Types
- Special Assignment Expressions
- Control Structures
  - If statement
  - Switch statement
  - Do-while, While and For statement

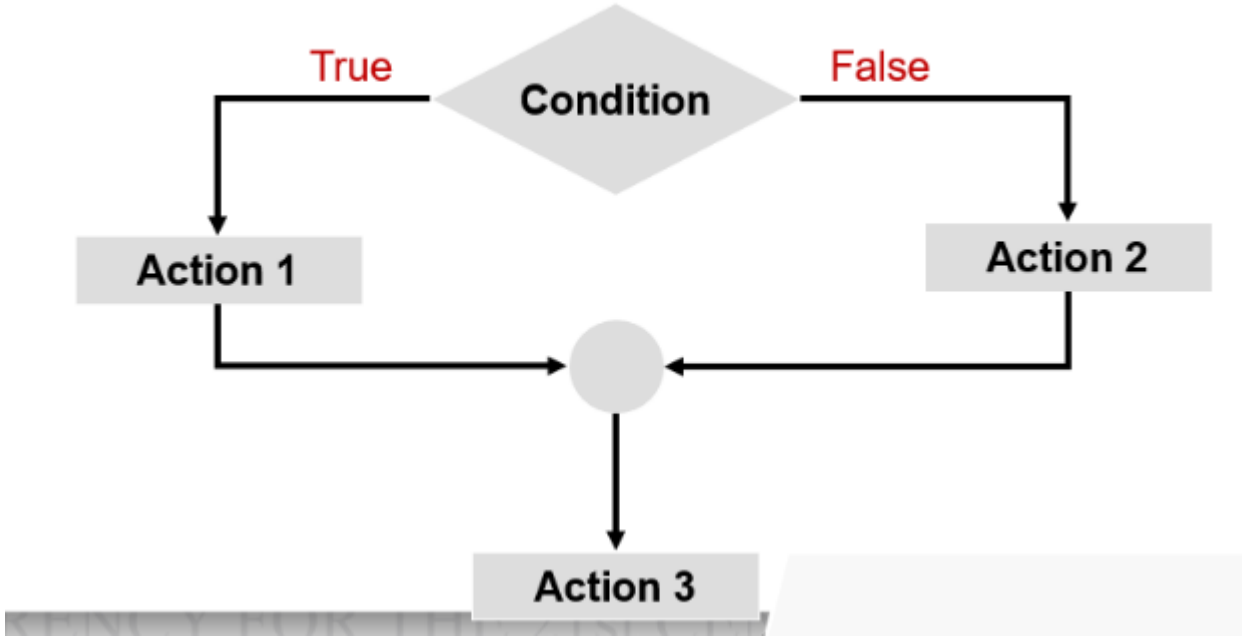
Control Structures

- **Sequence Structure (straight line)**



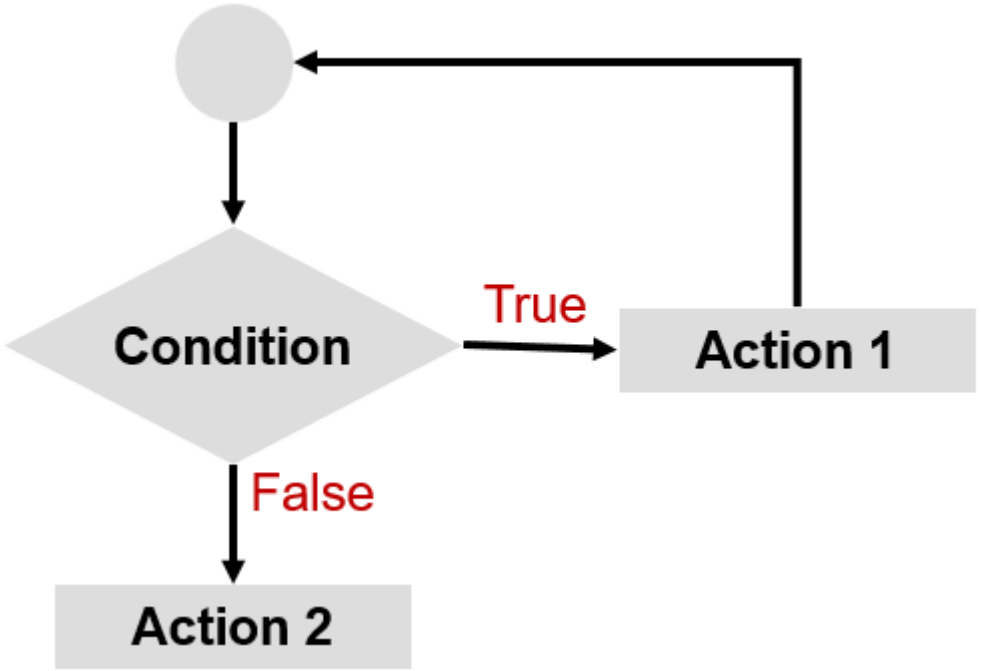
Control Structures

- **Selection Structure (branching)(if-else)**



Control Structures

- Loop Structure (iteration or repetition)



If Statement

The if statement is implemented in two forms:

- Simple if statement

```
if (expression is true)
{
        action 1;
}
action 2;
```

If Statement

- if … else statement

```
if (expression is true)
{
        action 1;
}
else
{
        action 2;
}
action 3;
```

The Switch Statement

```
switch (choice)
{
        case 1:
                action 1;
                break;
        case 2:
                action 2;
                break;
        case 3:
                action 3;
                break;
        default:
                action 4;

}
        action 4;
```

Thank You