CONTENTS

Module 1 Lecture

Introduction to C++ : Object Oriented Technology, Advantage of OOP, Input-output in C++, Tokens, Keyword, Identifiers, Data Types C++, Derives data types. The void data type, Type Modifiers, Typecasting, Constant, Operator, Precedence of Operators, String.

Module 2 Lecture

Control Structures and Functions : Decision making statements like if-else, Nested if-else, goto, break, continue, switch case, Loop statement like for loop, nested for loop, while loop, do-while loop. Partsof function, User- defined Functions, Value-Returning Functions void Functions, Value Parameters, Function overloading, Virtual Functions.

Module 3 Lecture

Classes and Data Abstraction : Structure in C++, Class, Build-in Operations on Classes, Assignment Operator and Classes, Class Scope, Reference parameters and Class Objects (Variables), Member functions, Accessor and Mutator Functions, Constructors, default Constructor, Destructors.

Module 4 Lecture

Overloading, Templates and Inheritance : Operator Overloading, Function Overloading, Function Templates, Class Templates. Single and Multiple Inheritance, virtual Base class, Abstract Class, Pointer and ineritance, Overloading Member Function.

Module 5 Lecture

Pointers, Arrays and Exception Handling : Void Pointers, Pointer to Class, Pointer to Object, Void Pointer, Arrays. The keywords try, throw and catch. Creating own Exception Classes, Exception Handling Techniques (Terminate the Program, Fix the Error and Continue, Log the Error and Continue), Stack Unwinding.

Unit - 1

Introduction to c++

Object oriented technology,advantage of oop,input output in c++, keywords, identifiers, data types inc++, derived datat types in c++, the void data type, typemodifiers, typecasting, constant, operator, precedence of operators, strings.

Q.1. Explain About C++ Programming Language.In Which Programming Paradigm It Is Based?

(AKU.2013)

Ans. C++ is a **multi-paradigm programming language** that **supports object-oriented programming (OOP)**, created by **B Jarne Stroustrup** in **1983** at **Bell Labs**. C++ is an extension (superset) of C programming and the programs written in C language can run in C++ compilers.

C++ is an Object Oriented Programming language but is not purely Object Oriented. Its features like Friend and Virtual, violate some of the very important OOPS features, rendering this language unworthy of being called completely Object Oriented. Its a middle level language.

C++ is used by programmers to create computer software. It is used to create general systems software, drivers for various computer devices, software for servers and software for specific applications and also widely used in the creation of video games.

.2. Write The Advantages Or Benefits Of Using C++ Programming Language Over C Programming Language.(procedural programming language)-

(AKU.2012,2013)

ns. Benefits of C++ over C Language

The major difference being OOPS concept, C++ is an object oriented language whereas C language is a procedural language. Apart form this there are many other features of C++ which gives this language an upper hand on C laguage.

Following features of C++ makes it a stronger language than C,

1. There is Stronger Type Checking in C++.
2. All the OOPS features in C++ like Abstraction, Encapsulation, Inheritance etc makes it more worthy and useful for programmers.
3. C++ supports and allows user defined operators (i.e Operator Overloading) and function overloading is also supported in it.
4. Exception Handling is there in C++.
5. The Concept of Virtual functions and also Constructors and Destructors for Objects.
6. Inline Functions in C++ instead of Macros in C language. Inline functions make complete function body act like Macro, safely.
7. Variables can be declared anywhere in the program in C++, but must be declared before they are used.

Q.3. List Out The Different Compilers Available For C++ Programming Languages.

Ans. List of Available C++ Compilers for Different OS

For Windows :

- Code::Blocks
- Borland C++
- Microsoft Visual C++
- Turbo C++

For Linux:

- g++ is a C++ compiler that comes with most *nix distributions.

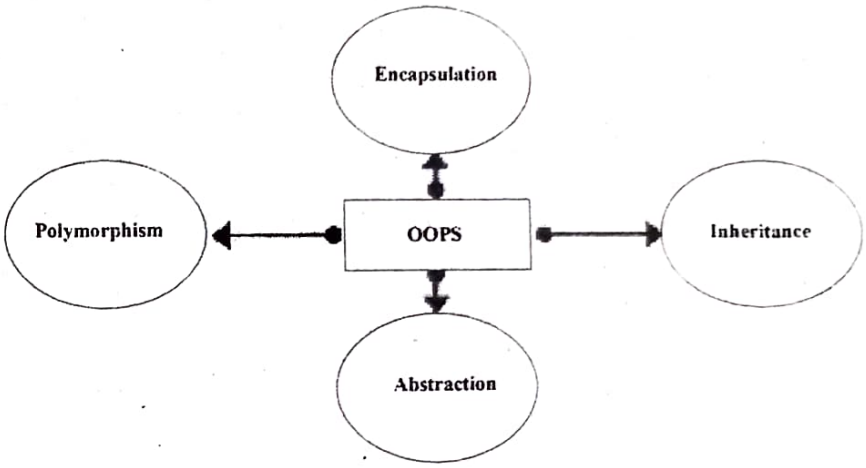
For Mac OS :

- Apple XCode
- C++ Code::Block

Q.4. Explain How C++ Fillow The Oops Concept? Write All The Features Of Oo Which C++ Follows. **(AKU.201**

Ans. C++ and Object Oriented Programming

Object Oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc.



1. Objects

Objects are the basic unit of OOP. They are instances of class, which have data members and uses various member functions to perform tasks.

2. Class

It is similar to structures in C language. Class can also be defined as user defined data type but it also contains functions in it. So, class is basically a blueprint for object. It declare & defines what data variables the object will have and what operations can be performed on the class's object.

Classes name must start with capital letter, and they contain data variables and member functions.

```
class Abc
{
inti;              //data variable
void display()  //Member Function
{
cout<<"Inside Member Function";
}
}; // Class ends here
int main()
{
```

```
Abcobj;  // CreatigAbc class's object
obj.display();  //Calling member function using class object
}
```

3. Abstraction

Abstraction refers to showing only the essential features of the application and hiding the details. In C++, classes provide methods to the outside world to access & use the data variables, but the variables are hidden from direct access. This can be done access specifiers.

4. Encapsulation

It can also be said data binding. Encapsulation is all about binding the data variables and functions together in class.

5. Inheritance

Inheritance is a way to reuse once written code again and again. The class which is inherited is called base calls & the class which inherits is called derived class. So when a derived class inherits a base class, the derived class can use all the functions which are defined in base class, hence making code reusable.

6. Polymorphism

It is a feature, which lets us create functions with same name but different arguments which will perform differently. That is function with same name, functioning in different way. Or, it also allows us to redefine a function to provide its new definition. You will learn how to do this in details soon in coming lessons.

7. Exception Handling

Exception handling is a feature of OOP, to handle unresolved exceptions or errors produced at runtime.

Q.5. Write The First Basic Program In C++ And Explain It's Each Parts.

Ans. First C++ program

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

```
cout<< "Hello this is C++";
}
```

- Header files are included at the beginning just like in C program. Here iostream is a header file which provides us with input & output streams. Header files contained predeclared function libraries, which can be used by users for their ease.
- Using namespace std, tells the compiler to use standard namespace. Namespace collects identifiers used for class, object and variables. NameSpace can be used by two ways in a program, either by the use of using statement at the beginning, like we did in above mentioned program or by using name of namespace as prefix before the identifier with scope resolution (::) operator.

  Example : std::cout<< "A";
- main(),.is the function which holds the executing part of program its return type is int.
- cout<<, is used to print anything on screen, same as printf in C language. cin and cout are same as scanf and printf, only difference is that you do not need to mention format specifiers like, %d for intetc, in cout&cin.

Q.6. Write About Basic Input/Output Used In C++.

Ans. The input/output in C++

C do not have built-in input/output facilities. Instead, it left the I/O to the compiler as external library functions (such as printf and scanf) in stdio (standard input output) library. The ANSI C standard formalized these IO functions into Standard IO package (stdio.h). Likewise, C++ continues this approach and formalizes IO in iostream and fstream libraries.

Features of I/O in C++

- C++ IO is type safe.
- C++ IO operations are based on streams of bytes and are device independent.

Streams in C++

C++ IO is based on streams, which are sequence of bytes flowing in and out of the programs (just like water and oil flowing through a pipe). I/O systems in C++ are designed to work with a wide variety of devices including terminals, disks and tape drives. The input output system supplies an interface to the programmer that is independent of the actual device being accessed. This interface is known as stream. A stream is a sequence

of bytes which acts either as a source from which input data can be obtained or as a destination to which output data can be sent. The source stream which provides data to the program is called the input stream and the destination stream which receives output from the program is called the output stream.

Example:

```
#include<iostream>
usingnamespacestd;

void main()
{
int g;
cin>>g;
cout<<"Output is: "<< g;
}
```

Q.7. Write about put() and get() function and getline() and write() function IN C++

(AKU.201

Ans. put() and get() functions

The classes istream and ostream defines two member functions get() and put() respectiv to handle single character input/output operations.

getline() and write()

You can read and display lines of text more efficiently using the lie oriented input/out functions. They are:

- getline()
- write()

The getline() function reads the entire line of texts that ends with newline charac The general form of getline() is:

cin.getline (line, size);

The write() function displays the entire line of text and the general form of writing function is:

cout.write (line, size);

Q.8. What Do You Mean By Data Type In C++ .Write The Typesof Data Types Derived And Built In Data Type. **(AKU.2012)**

Ans. Data Types in C++

They are used to define type of variables and contents used. Data types define the way you use storage in the programs you write. Data types can be built in or abstract.

1. Built in Data Types

These are the data types which are predefined and are wired directly into the compiler. eg: int, char etc.Any programming language has built in data types. The data types are used to create variables or your new data types. A variable is a amount of memory that has its own name and value. That's why it is important to know the built in data types, their properties, diapason and size. The official Microsoft site is offering the following information about C++ built in data types:

Type	Keyword
Boolean	Bool
Character	Char
Integer	Int
Floating point	Float
Double floating point	Double
Valueless	Void
Wide character	wchar_t

Q.9. Write A Program To Find Out The Correct Size Of Various Data Types In C++.

Ans. Following is the example, which will produce correct size of various data types on your computer.

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

int main() {
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 float : " <<sizeof(float) <<endl;
cout<< "Size of double : " <<sizeof(double) <<endl;
cout<< "Size of wchar_t : " <<sizeof(wchar_t) <<endl;

return 0;
}
```

This example uses endl, which inserts a new-line character after every line and << operator is being used to pass multiple values out to the screen. We are also using sizeof() operator to get size of various data types.

When the above code is compiled and executed, it produces the following result which can vary from machine to machine "

```
Size of char : 1
Size of int : 4
Size of short int : 2
Size of long int : 4
Size of float : 4
Size of double : 8
Size of wchar_t : 4
```

Q.10. What Is User Defined Or Abstract Data Type ?Explain With Example.

Ans.

- User defined or Abstract data types

  These are the type, that user creates as a class. In C++ these are classes where as in C it was implemented by structures.

  Enum as Data type

  Enumerated type declares a new type-name and a sequence of value containing identifiers which has values starting from 0 and incrementing by 1 every time.

For Example :

enum day(mon, tues, wed, thurs, fri) d;

Here an enumeration of days is defined with variable d. mon will hold value 0, tue will have 1 and so on. We can also explicitly assign values, like, enumday(mon, tue=7, wed);. Here, mon will be 0, tueis assigned 7, so wed will have value 8.

Q.11. Why to use enum ? Benefits of enum.

Ans.

1. Easier to change the values in future
2. Reduces errors due to mistyping numbers
3. Easy to read code so less chances of errors
4. Code will have forward compatibility when we want to change enumeration values in future.we use enum when we want a readable code that uses constants. Enum makes our program easier and clearer to read. By default enum values will be of type int. The integer values can be positive or negative and can be non-unique. Any non-defined enumerators will be assigned one greater value than the previous enumerator automatically.

Q.12. Write the uses of void data type in C++ ?

Ans.

- Void_data_type:

  It is used for following purposes;

  - It specifies the return type of a function when the function is not returning any value.
  - It indicates an empty parameter list on a function when no arguments are passed.
  - A void pointer can be assigned a pointer value of any basic data type.

Q.13. Define modifiers used in C++? Write the names of modifiers.

Ans.

- Modifiers:->Specifiers modify the meanings of the predefined built-in data types and expand them to a much larger set. There are four data type modifiers in C++, they are :

1. long
2. short

3. signed

4. unsigned

Below mentioned are some important points you must know about the mo

- long and short modify the maximum and minimum values that a data type
- A plain int must have a minimum size of short.
- Size hierarchy : short int<int< long int
- Size hierarchy for floating point numbers is : float < double < long double
- long float is not a legal type and there are no short floating point numbers.
- Signed types includes both positive and negative numbers and is the defau
- Unsigned, numbers are always without any sign, that is always positive.

Q.14. Explain All The Named Of Type Quantifiers Used In C++.

Ans. Type Qualifiers in C++

The type qualifiers provide additional information about the variables they

Sr. No.	Qualifier & Meaning
1.	Const. Objects of type const cannot be changed by your program during execution.
2.	VolatileThe modifier volatile tells the compiler that a variable's value may be changed in ways not explicitly specified by the program.
3.	RestrictA pointer qualified by restrict is initially the only means by which the object it points to can be accessed. Only C99 adds a new type qualifier called restrict.

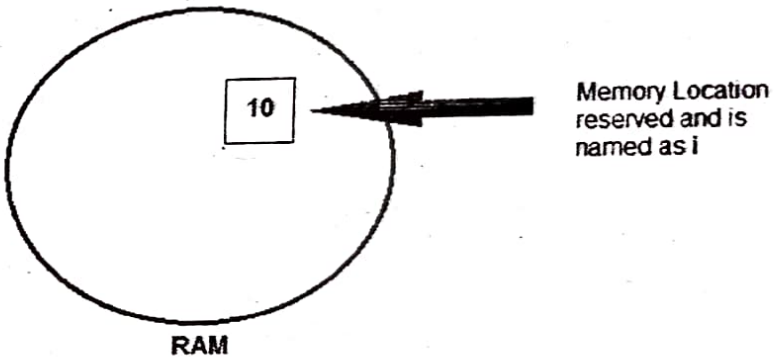
Q.15. WhatAre Variables? Explain it with proper example.

Ans. A variable provides us with named storage that our programs can manipul variable in C++ has a specific type, which determines the size and layout of the memory; the range of values that can be stored within that memory; and t operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C++ is case-sensitive .

Variable are used in C++, where we need storage for any value, which will change in program. Variable can be declared in multiple ways each with different memory requirements and functioning. Variable is the name of memory location allocated by the compiler depending upon the datatype of the variable.

Example : `int i=10; // declared and initialised`



Basic types of Variables :

Each variable while declaration must be.given a datatype, on which the memory assigned to the variable depends. Following are the basic types of variables,

bool	For variable to store boolean values (True or False)
char	For variables to store character types.
int	for variable with integral values

float and double are also types for variables with large and floating point values

Q.16. How You Can Declare And Initialize The Variable?

Ans. Declaration and Initialization :

Variable must be declared before they are used. Usually it is preferred to declare them at the starting of the program, but in C++ they can be declared in the middle of program too, but must be done before using them.

Example :

inti; // declared but not initialised

char c;

inti, j, k; // Multiple declaration

Initialization means assigning value to an already declared variable,

inti; // declaration

i = 10; // initialization

Initialization and declaration can be done in one single step also,

inti=10; //initialization and declaration in same step

inti=10, j=11;

If a variable is declared and not initialized by default it will hold a garbage value. Also if a variable is once declared and if try to declare it again, we will get a compile time error.

Q.17. Write About Scope Of Variables Of C++.

Ans. Scope of Variables :

All the variables have their area of functioning, and out of that boundary they don't hold their value, this boundary is called scope of the variable. For most of the cases, between the curly braces,in which variable is declared that a variable exists, not outside it. We will study the storage classes later, but as of now, we can broadly divide variables into two main types,

- Global Variables
- Local variables

1. Global variables :

Global variables are those, which ar once declared and can be used throughout the lifetime of the program by any class or any function. They must be declared outside the main() function. If only declared, they can be assigned different values at different time in program lifetime. But even if they are declared and initialized at the same time outside the main() function, then also they can be assigned any value at any point in the program.

Example : Only declared, not initialized.

include<iostream>

using namespace std;

```
int x;                  // Global variable declared
int main()
{
 x=10;                  // Initialized once
cout<<"first value of x = "<< x;
 x=20;                  // Initialized again
cout<<"Initialized again with value = "<< x;
}
```

2. Local Variables :

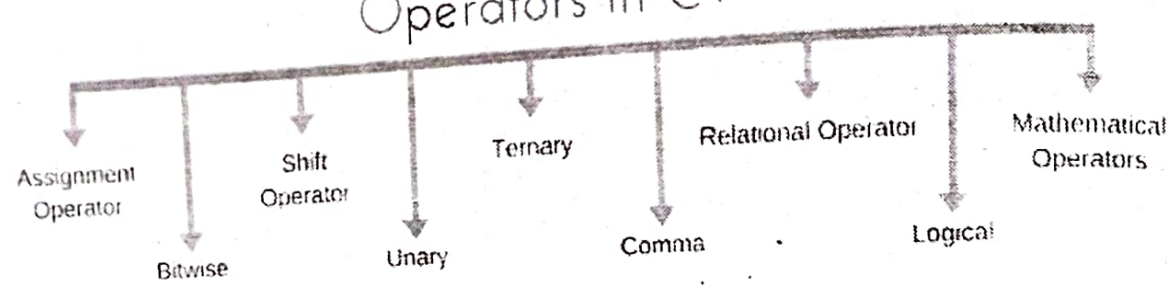
Local variables are the variables which exist only between the curly braces, in which its declared. Outside that they are unavailable and leads to compile time error.

Example :

```
include<iostream>
using namespace std;
int main()
{
inti=10;
if(i<20)        // if condition scope starts
  {
int n=100;  // Local variable declared and initialized
  }             // if condition scope ends
cout<< n;       // Compile time error, n not available here
}
```

Q.18. Define the Operators used in C++. Explain it's all types.

Ans. Operators are special type of functions, that takes one or more arguments and produces a new value. For example : addition (+), substraction (-), multiplication (*) etc, are all operators. Operators are used to perform various operations on variables and constants.



Types of operators

1. Assignment Operator
2. Mathematical Operators
3. Relational Operators
4. Logical Operators
5. Bitwise Operators
6. Shift Operators
7. Unary Operators
8. Ternary Operator
9. Comma Operator

- Assignment Operator (=)

  Operates '=' is used for assignment, it takes the right-hand side (called rvalue) and it into the left-hand side (called lvalue). Assignment operator is the only operator w can be overloaded but cannot be inherited.

- Mathematical Operators

  There are operators used to perform basic mathematical operations. Addition (subtraction (–) , diversion (/) multiplication (*) and modulus (%) are the b mathematical operators. Modulus operator cannot be used with floating-point numb

  C++ and C also use a shorthand notation to perform an operation and assignmen same type.

  Example :

  int x=10;

  x += 4 // will add 4 to 10, and hence assign 14 to X.

  x –= 5 // will subtract 5 from 10 and assign 5 to x.

Relational Operators

These operators establish a relationship between operands. The relational operators are : less than (<) , grater thatn (>) , less than or equal to (<=), greater than equal to (>=), equivalent (==) and not equivalent (!=).

You must notice that assignment operator is (=) and there is a relational operator, for equivalent (==). These two are different from each other, the assignment operator assigns the value to any variable, whereas equivalent operator is used to compare values, like in if-else conditions, Example

```
int x = 10;  //assignment operator
x=5;         // again assignment operator
if(x == 5)   // here we have used equivalent relational operator, for comparison
{
cout<<"Successfully compared";
}
```

- Logical Operators

The logical operators are AND (&&) and OR (||). They are used to combine two different expressions together.

If two statement are connected using AND operator, the validity of both statements will be considered, but if they are connected using OR operator, then either one of them must be valid. These operators are mostly used in loops (especially while loop) and in Decision making.

- Bitwise Operators

There are used to change individual bits into a number. They work with only integral data types like char, int and long and not with floating point values.

- Bitwise AND operators &
- Bitwise OR operator |
- And bitwise XOR operator ^
- And, bitwise NOT operator ~

They can be used as shorthand notation too, &= , |= , ^= , ~= etc.

- Shift Operators

Shift Operators are used to shift Bits of any variable. It is of three types,

1. Left Shift Operator <<
2. Right Shift Operator >>
3. Unsigned Right Shift Operator >>>

- Unary Operators

These are the operators which work on only one operand. There are many una operators, but increment ++ and decrement — operators are most used.

Other Unary Operators : address of &, dereference *, new and delete, bitwise not logical not !, unary minus - and unary plus +.

- Ternary Operator

The ternary if-else ? : is an operator which has three operands.

int a = 10;

a> 5 ? cout<< "true" : cout<< "false"

- Comma Operator

This is used to separate variable names and to separate expressions. In case expressions, the value of last expression is produced and used.

Example :

inta,b,c; // variables declaration using comma operator

a=b++, c++; // a = c++ will be done.

Q.19. What Is Operator Precedence In C++?

Ans. Precedence of Operators :

An Operator Precedence is the hierarchy in which operators are evaluated. For examp in the expression 2 + 2 * 3, the result would be 8 since the "*" operator has high precedence over the "+" operator.

Now the same expression is used with a paranthesis like (2 + 2) * 3, the result wou be 12. The "()" is used to force precedence by making the expression inside the brac executed first. If the precedence are equal then the associativity is used to evaluate a expression.

This is the Operator Precedence in C++.

Name	Operator	Direction	Precedence
Parentheses	()	Left to Right	1
Post-increment	++	Left to Right	2
Post-decrement	—	Left to Right	2
Address	&	Right to Left	2
Bitwise NOT	~	Right to Left	2
Typecast	(type)	Right to Left	2
Logical NOT	!	Right to Left	2
Negation	-	Right to Left	2
Plus Sign	+	Right to Left	2
Pre-increment	++	Right to Left	2
Pre-decrement	—	Right to Left	2
Size of data	sizeof	Right to Left	2
Modulus	%	Left to Right	3
Multiplication	*	Left to Right	3
Division	/	Left to Right	3
Addition	+	Left to Right	4
Subtraction	-	Left to Right	4
Bitwise Shift Left	<<	Left to Right	5
Bitwise Shift Right	>>	Left to Right	5
Less Than	<	Left to Right	6
Less Than or Equal	<=	Left to Right	6
Greater Than	>	Left to Right	6

Name	Operator	Direction	Precedence
Greater Than or Equal	>=	Left to Right	6
Equal	==	Left to Right	7
Not Equal	!=	Left to Right	7
Bitwise AND	&	Left to Right	8
Bitwise XOR	^	Left to Right	9
Bitwise OR	|	Left to Right	10
Logical AND	&&	Left to Right	11
Logical OR	||	Left to Right	12
Condition Expression	?:	Right to Left	13
Assignment	=	Right to Left	14
Additive Assignment	+=	Right to Left	14
Subtractive Assignment	-=	Right to Left	14
Multiplicative Assignment	*=	Right to Left	14
Divisional Assignment	/=	Right to Left	14
Modulating Assignment	%=	Right to Left	14
Left Shift Assignment	>>=	Right to Left	14

What Is The Use Of Storage Classes In C++? Explain All The Types Of Sto Classes.

Storage Classes in C++

Storage classes are used to specify the lifetime and scope of variables. How sto is allocated for variables and How variable is treated by complier depends on t storage classes.

These are basically divided into 5 different types :

Global variables

Local variables

3. Register variables
4. Static variables
5. Extern variables

Global Variables :

These are defined at the starting , before all function bodies and are available throughout the program.

```
using namespace std;
int globe;      // Global variable
voidfunc();
int main()
{
 .....
}
```

Local variables :

They are defined and are available within a particular scope. They are also called Automatic variablebecause they come into being when scope is entered and automatically go away when the scope ends.

The keyword auto is used, but by default all local variables are auto, so we don't have to explicitly add keyword auto before variable dedaration. Default value of such variable is garbage.

Register variables :

This is also a type of local variable. This keyword is used to tell the compiler to make access to this variable as fast as possible. Variables are stored in registers to increase the access speed.

But you can never use or compute address of register variable and also , a register variable can be declared only within a block, that means, you cannot have global or static register variables.

Static Variables :

Static variables are the variables which are initialized & allocated storage only once at the beginning of program execution, no matter how many times they are used and called

in the program. A static variable retains its value until the end of program.

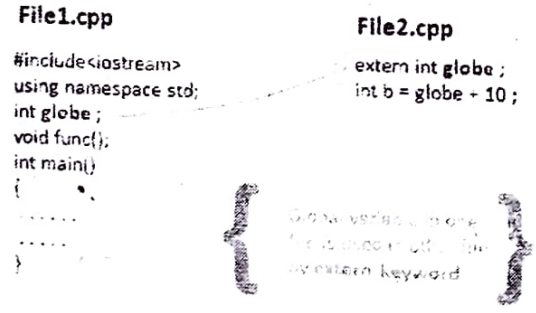
```
void fun()
{
staticinti = 10;
i++;
cout<<i;
}
int main()
{
fun();      // Output = 11
fun();      // Output = 12
fun();      // Output = 13
}
```

As, i is static, hence it will retain its value through function calls, and is initialized once at the beginning.

Static specifiers are also used in classes, but that we will learn later.

Extern Variables :

This keyword is used to access variable in a file which is declared & defined in s other file, that is the existence of a global variable in one file is declared using ex keyword in another file.



Q.21. Explain strings in C++. What is the two types of string representation.

Ans. C++ provides following two types of string representations—

- The C-style character string.
- The string class type introduced with Standard C++.

The C-Style Character String :

The C-style character string originated within the C language and continues to be supported within C++. This string is actually a one-dimensional array of characters which is terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null.

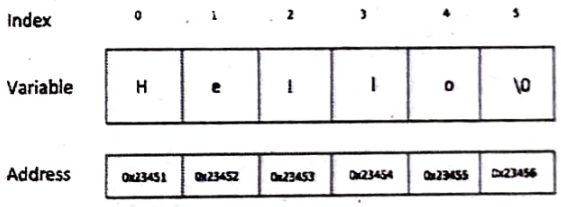
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

If you follow the rule of array initialization, then you can write the above statement as follows—

char greeting[] = "Hello";

Following is the memory presentation of above defined string in C/C++ —



Following example makes use of few of the above-mentioned functions—

```
#include<iostream>
#include<cstring>

usingnamespacestd;

int main (){
char str1[10]="Hello";
char str2[10]="World";
char str3[10];
intlen;
// copy str1 into str3
strcpy( str3, str1);
```

```
cout<<"strcpy( str3, str1) : "<< str3 <<endl;
// concatenates str1 and str2
strcat( str1, str2);
cout<<"strcat( str1, str2): "<< str1 <<endl;
// total lenghth of str1 after concatenation
len=strlen(str1);
cout<<"strlen(str1) : "<<len<<endl;
return0;
}
```

When the above code is compiled and executed, it produces result something as follows.

```
strcpy( str3, str1) : Hello
strcat( str1, str2): HelloWorld
strlen(str1) : 10
```

Q.22. Define The String Class in C++.

Ans. The standard C++ library provides a string class type that supports all the operation mentioned above, additionally much more functionality. Let us check the followin example –

```
#include<iostream>
#include<string>
usingnamespacestd;
int main (){
string str1 ="Hello";
string str2 ="World";
string str3;
intlen;
// copy str1 into str3
   str3 = str1;
cout<<"str3 : "<< str3 <<endl;
```

```
// concatenates str1 and str2
  str3 = str1 + str2;
cout<<"str1 + str2 : "<< str3 <<endl;// total length of str3 after concatenation
len= str3.size();
cout<<"str3.size() : "<<len<<endl;
return0;
}
```

Output:

str3 : Hello

str1 + str2 : HelloWorld

str3.size() : 10

Q.23. Define how to copy one string to another string.

Ans.

strcpy_s(destination, source)

Below is the demo of using strcpy_s

```
charmsg[10]={'M','E','S','S','A','G','E','\0'};
chardest[20];
strcpy_s(dest,msg);
cout<<"String copied from msg = "<<dest<<endl;
```

The output of above example is shown below

String copied from msg = MESSAGE

Q.24. Explain concatenation of one string to another.

Ans. Concatenate one string fo another one

strcat_s(string1, string2)

string2 will be appended to the end of the string1

```
//declare 2 messages
char string1[20]="Good";
char string2[]="Morning";
```

```
//concatenate strings
strcat_s(string1, string2);
cout<< string1 <<endl;
```

The output os above program will be:

Good Morning

Q.25. Explain how to Compare two strings ?

Ans.

strcmp(string1, string2)

This function will return **0**, if the strings are equal; **negative** value, if string1 is **less** than string2 and **positive** value if string1 is **greater** than string2.

```
//declare two strings
char str1[100];
char str2[100];
//get user input of strings:
cout<<"Please enter the first string\n";
cin>> str1;
cout<<"Please enter the second string\n";
cin>> str2;
//compare strings
int result =strcmp(str1, str2);
if(result ==0)
{
//strings are equals
cout<< str1 <<" is equal to "<< str2 <<endl;
}
else
{
if(result >0)//str1 is greater
```

```
cout<< str1 <<" is greater than "<< str2 <<endl;
else//str2 is greater
cout<< str1 <<" is less than "<< str2 <<endl;
}
```

Below is the output of above program

```
Please enter the first string
abc
Please enter the second string
abd
abc is less than abd
```

strcmp compares strings in lexicographical (alphabetical) order. "less than" for strings means that "cat" is less than "dog" because "cat" comes alphabetically before "dog".

❐❐❐

Unit - 2

Control Structures

Decision making statements like if-else, Nested if-else, goto, break, continue, switch case, Loop statement like for loop, nested for loop, while loop, do-while loop.

Q.1. Define Decision Making Structure In C++? Why It Is Used ? And Also Explain The Types Of Decision Making Structure. (AKU. 20

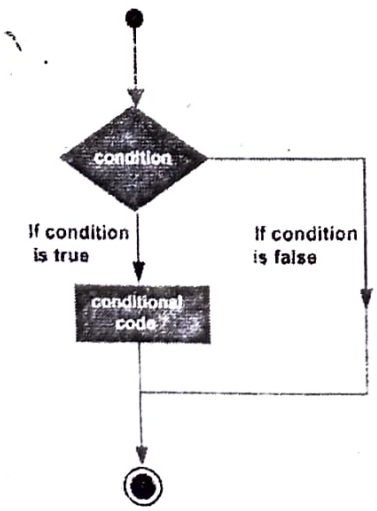
Ans. Decision making is about deciding the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met.

C++ handles decision-making by supporting the following statements :

- if statement
- switch statement
- conditional operator statement
- goto statement

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

FLOW CHART :



Q.2. Explain If Statement Used For Decision Making In C++.

Ans. The if statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are,

1. Simple if statement
2. If....else statement
3. Nested if....else statement
4. else if statement

An if statement consists of a boolean expression followed by one or more statements.

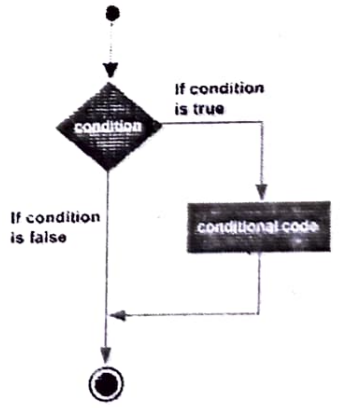
Syntax :

The syntax of an if statement in C++ is "

```
if(boolean_expression) {
   // statement(s) will execute if the boolean expression is true
}
```

If the boolean expression evaluates to true, then the block of code inside the if statement will be executed. If boolean expression evaluates to false, then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Flow Diagram :



Example

```
#include<iostream>
usingnamespace std;

int main (){
// local variable declaration:
```

```
int a =10;

// check the boolean condition
if( a <20){
// if condition is true then print the following
      cout <<"a is less than 20;"<< endl;
}
   cout <<"value of a is : "<< a << endl;

return0;
}
```

When the above code is compiled and executed, it produces the following result-

a is less than 20;

value of a is : 10

Q.3. Explain if.....else statement used in C++.

Ans. An if statement can be followed by an optional else statement, which executes wh the boolean expression is false.

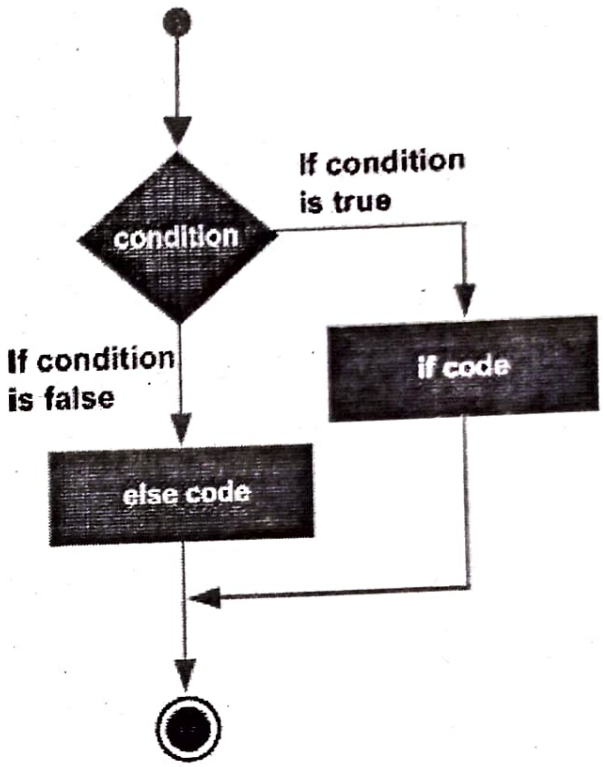
Syntax :

The syntax of an if...else statement in C++ is —

```
if(boolean_expression) {
   // statement(s) will execute if the boolean expression is true
} else {
   // statement(s) will execute if the boolean expression is false
}
```

If the boolean expression evaluates to true, then the if block of code will be execute otherwise else block of code will be executed.

Flow Diagram :



Example

```
#include<iostream>
usingnamespace std;

int main (){
// local variable declaration:
int a =100;

// check the boolean condition
if( a <20){
// if condition is true then print the following
cout <<"a is less than 20;"<< endl;
}else{
// if condition is false then print the following
cout <<"a is not less than 20;"<< endl;
}
cout <<"value of a is : "<< a << endl;
return0;
}
```

When the above code is compiled and executed, it produces the following result—

a is not less than 20;

value of a is : 100

Q.4. Explain if.else. If...else statement with a proper example. **(AKU, 201**

Ans. An if statement can be followed by an optional else if...else statement, which is ver usefull to test various conditions using single if...else if statement.

When using if, else if, else statements there are few points to keep in mind.

- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of he remaining else if's or else's will be tested.

Q.5. Write The Syntax of If...Else..If Statement With Proper Programming Example

Ans. **The syntax of an if...else if...else statement in C++ is—**

```
if(boolean_expression 1) {
   // Executes when the boolean expression 1 is true
} else if( boolean_expression 2) {
   // Executes when the boolean expression 2 is true
} else if( boolean_expression 3) {
   // Executes when the boolean expression 3 is true
} else {
   // executes when the none of the above condition is true.
}
```

EXAMPLE :

```
main( )
{
 int x,y;
 x=15;
 y=18;
 if (x > y )
```

```
{
 cout << "x is greater than y";
}
else
{
 cout << "y is greater than x";
}
}
```

Output :

y is greater than x

Q.6. Explain Switch Statement With Syntax And Flow Chart.

Ans. A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax :

The syntax for a switch statement in C++ is as follows—

```
switch(expression) {
    case constant-expression  :
 statement(s);
break; //optional
    case constant-expression  :
statement(s);
break; //optional

    // you can have any number of case statements.
    default : //Optional
 statement(s);
}
```

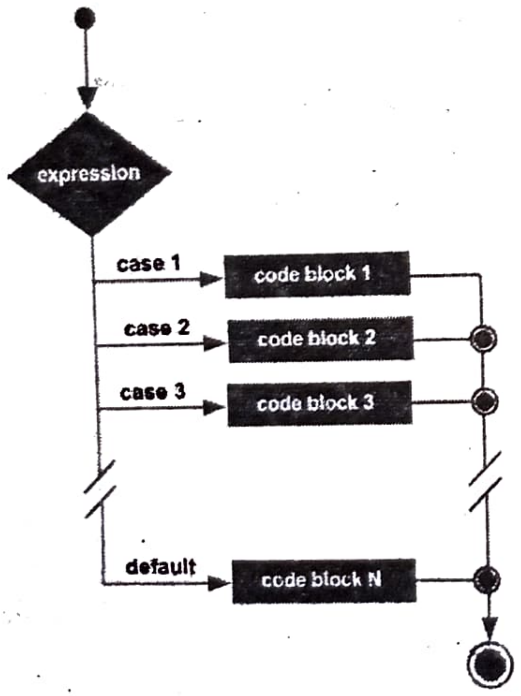
The following rules apply to a switch statement —

The expression used in a switch statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.

You can have any number of case statements within a switch. Each case is follow by the value to be compared to and a colon.

- The constant-expression for a case must be the same data type as the variable in t switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following th case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of cont jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of cont will fall through to subsequent cases until a break is reached.
- A switch statement can have an optional default case, which must appear at the end the switch. The default case can be used for performing a task when none of the cas is true. No break is needed in the default case.

Diagram :



Q.7. Write A Program By Using Switch Statement.

Ans.

```
#include<iostream>
usingnamespace std;

int main (){
```

```
// local variable declaration:
char grade ='D';

switch(grade){
case'A':
        cout <<"Excellent!"<< endl;
break;
case'B':
case'C':
        cout <<"Well done"<< endl;
break;
case'D':
        cout <<"You passed"<< endl;
break;
case'F':
        cout <<"Better try again"<< endl;
break;
default:
        cout <<"Invalid grade"<< endl;
}
   cout <<"Your grade is "<< grade << endl;

return0;
}
```

This would produce the following result—

```
You passed
Your grade is D
```

Q.8. Explain Nested If Statements With Syntax And Example.

Ans. It is always legal to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).

Syntax : The syntax for a nested if statement is as follows—

[illegible] is true

```
    }
}
```

You can nest else if...else in the similar way as you have nested if statement.

Example

```
#include <iostream>
using namespace std;
int main (){
// local variable declaration
int a =100;
int b =200;
// check the boolean condition
if( a ==100){
// if condition is true then check the following
if( b ==200){
// if condition is true then print the following
        cout <<"Value of a is 100 and b is 200"<< endl;
}
}
    cout <<"Exact value of a is :"<< a << endl;
    cout <<"Exact value of b is :"<< b << endl;
return0;
}
```

When the above code is compiled and executed, it produces the following result:

Value of a is 100 and b is 200

Exact value of a is : 1(n)

Exact value of b is : 2(n)

Q.9. Explain Nested Switch Statements with syntax.

Ans. It is possible to have a switch as part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.C++ specifies that at least 256 levels of nesting be allowed for switch statements.

Syntax : The syntax for a nested switch statement is as follows—

```
switch(ch1) {
  case 'A':
    cout << "This A is part of outer switch";
    switch(ch2) {
      case 'A':
        cout << "This A is part of inner switch";
        break;
      case 'B': // ...
    }
    break;
  case 'B': // ...
}
```

.10. Write An Example Of Nested Switch Statement. (AKU.2015)

ns.

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

```
switch(a){
case100:
        cout <<"This is part of outer switch"<< endl;
switch(b){
case200:
      cout <<"This is part of inner switch"<< endl;
}
}
   cout <<"Exact value of a is : "<< a << endl;
   cout <<"Exact value of b is : "<< b << endl;

return0:
}
```

This would produce the following result—

This is part of outer switch

This is part of innet switch

Exact value of a is : 100

Exact value of b is : 200

Note for the students :

- In if statement, a single statement can be included without enclosing it into curly brac

```
}
int a = 5;
if(a > 4)
  cout << "success";
```

No curly braces are required in the above case, but if we have more than one state inside ifcondition, then we must enclose them inside curly braces.

- == must be used for comparison in the expression of if condition, if you use = expression will always return true, because it performs assignment not compari

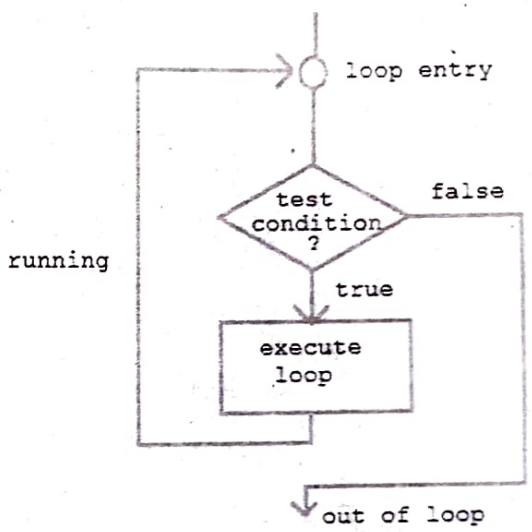
Q.11. What is the use of Looping IN C++ ?

Ans. In any programming language, loops are used to execute a set of statements repeat until a particular condition is satisfied.

There may be a situation, when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.Programming languages provide various control structures that allow for more complicated execution paths.

Q.12. Explain how loop works in C++.

Ans.



A sequence of statement is executed until a specified condition is true. This sequence of statement to be executed is kept inside the curly braces { } known as loop body. After every execution of loop body, condition is checked, and if it is found to be true the loop body is executed again. When condition check comes out to be false, the loop body will not be executed.

Q.13. How Many Types Of Looping In C++ ? Explain While Loop With Example.

Ans. There are 3 type of loops in C++ language :

1. while loop
2. for loop
3. do-while loop

While loop :

while loop can be address as an entry control loop. A while loop statement repeatedly executes a target statement as long as a given condition is true.

Syntax :

The syntax of a while loop in C++ is—

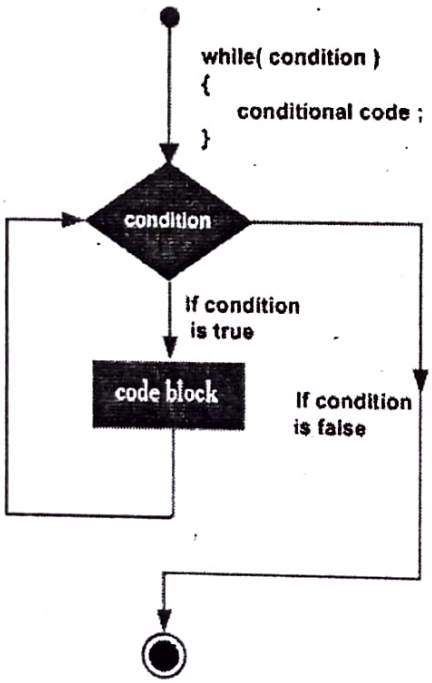
```
while(condition) {
    statement(s);
}
```

Here, statement(s) may be a single statement or a block of statements. The condi be any expression, and true is any non-zero value. The loop iterates while the c is true.

It is completed in 3 steps.

- Variable initialization.(e.g int x=0;)
- condition(e.g while(x<=10))
- Variable increment or decrement (x++ or x— or x=x+2)

Flow Diagram :



Here, key point of the while loop is that the loop might not ever run. When the c is tested and the result is false, the loop body will be skipped and the first st after the while loop will be executed.

Q.14. Write A Program By Using While Loop.

Ans.

```
#include<iostream>
usingnamespace std;

int main (){
```

```
// Local variable declaration:
int a =10;

// while loop execution
while( a <20){
      cout <<"value of a: "<< a << endl;
      a++;
}

return0;
}
```

When the above code is compiled and executed, it produces the following result –

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

Q.15. Explain For Loop With Syntax And Proper Flow Chart.

Ans. for loop is used to execute a set of statement repeatedly until a particular condition is satisfied. we can say it an open ended loop. A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax :

The syntax of a for loop in C++ is "

he flow of control in a for loop—

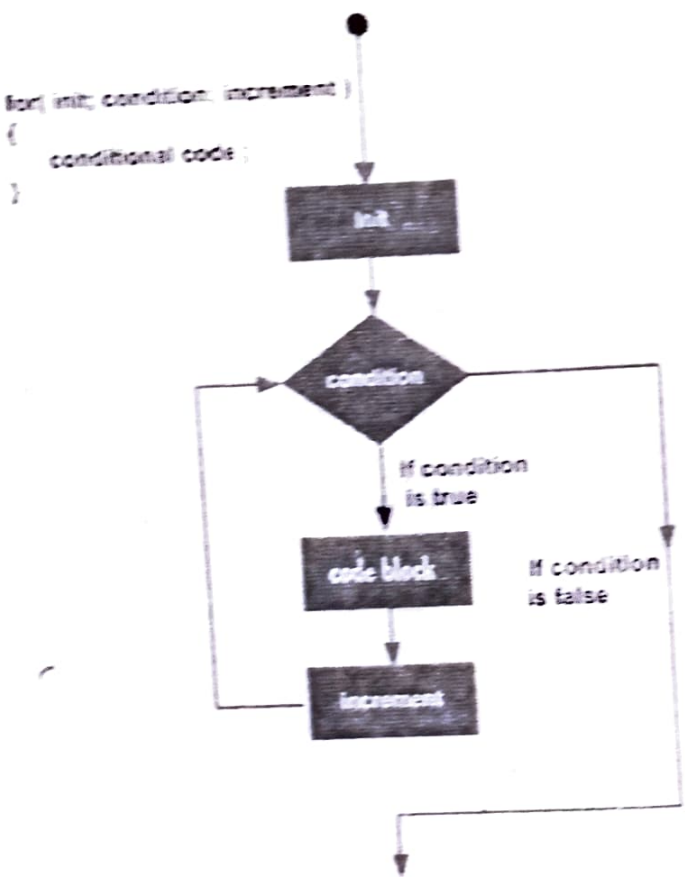
step is executed first, and only once. This step allows you to declare and initia
control variables. You are not required to put a statement here, as long a
on appears.

he condition is evaluated. If it is true, the body of the loop is executed. If i
he body of the loop does not execute and flow of control jumps to the n
ent just after the for loop.

the body of the for loop executes, the flow of control jumps back up
rement statement. This statement can be left blank, as long as a semicolon appe
he condition.

ondition is now evaluated again. If it is true, the loop executes and the proc
s itself (body of loop, then increment step, and then again condition). After
tion becomes false, the for loop terminates.

am :



Q.16. Write A

Ans.

```
#include
usingna

int mai
// for lo
for(int
    co
}

return0
}
```

When
value
value
value
value
value
value
value
value
value
value
value

Q.17. Expla

Ans. In so
Such
the b
Gene

A Program By Using For Loop.

```
<iostream>
mespace std;

n (){
op execution
a =10; a <20; a = a +1){
out <<"value of a: "<< a << endl;

;
```

the above code is compiled and executed. it produces the following result—

```
of a: 10
of a: 11
of a: 12
of a: 13
of a: 14
of a: 15
of a: 16
of a: 17
of a: 18
of a: 19
```

ain do while loop with syntax.

me situations it is necessary to execute body of the loop before testing the condition. situations can be handled with the help of do-while loop. do statement evaluates ody of the loop first and at the end, the condition is checked using while statement. eral format of do-while loop is,

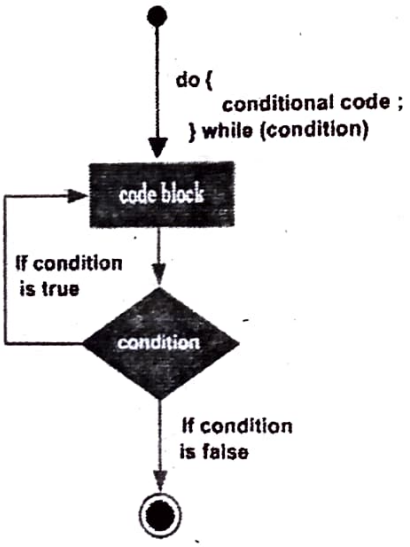
```
do
{
 ....
 .....
}
while(condition);
```

Unlike for and while loops, which test the loop condition at the top of the lo the do...while loop checks its condition at the bottom of the loop.

A do...while loop is similar to a while loop, except that a do...while loop is guarant to execute at least one time.

Q.18. Draw a flow Diagram of do While Loop.

Ans.



Q.19. Write A Program By Using Do While Loop.

Ans.

```
#include<iostream>
usingnamespace std;

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

// do loop execution
```

```
do{
    cout <<"value of a: "<< a << endl;
    a = a +1;
}while( a <20);

return0;
}
```

When the above code is compiled and executed, it produces the following result—

value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19

Q.20. Explain Nested Loops With Syntax.

Ans. A loop can be nested inside of another loop. C++ allows at least 256 levels of nesting.

Syntax :

The syntax for a nested for loop statement in C++ is as follows—

```
for ( init; condition; increment ) {
   for ( init; condition; increment ) {
      statement(s);
   }
   statement(s); // you can put more statements.
}
```

The syntax for a nested while loop statement in C++ is as follows—

```
while(condition) {
   while(condition) {
      statement(s);
   }
   statement(s); // you can put more statements.
}
```

The syntax for a nested do...while loop statement in C++ is as follows—

```
do {
   statement(s); // you can put more statements.
   do {
      statement(s);
   } while( condition );

} while( condition );
```

Q.21. What You Use For Jumping Out From Loop ?

Or, Explain Break Statement With Syntax And An Example.

Ans.

Sometimes, while executing a loop, it becomes necessary to skip a part of the loop to leave the loop as soon as certain condition becocmes true, that is jump out of lo C language allows jumping from one statement to another within a loop as well jumping out of the loop.

(1) Break statement :

When break statement is encountered inside a loop, the loop is immediately exited the program continues with the statement immediately following the loop.

The break statement has the following two usages in C++ —

- When the break statement is encountered inside a loop, the loop is immediately termin and program control resumes at the next statement following the loop.
- It can be used to terminate a case in the switch statement (covered in the next chap

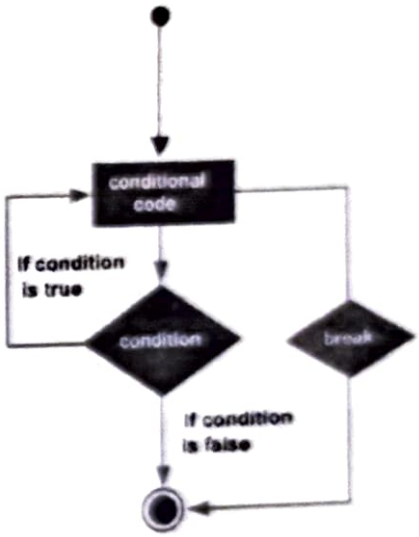
If you are using nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax

The syntax of a break statement in C++ is—

break;

Flow Diagram :



Example :

```
#include<iostream>
usingnamespace std;

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

// do loop execution
do{
      cout <<"value of a: "<< a << endl;
      a = a +1;
if( a >15){
// terminate the loop
break;
```

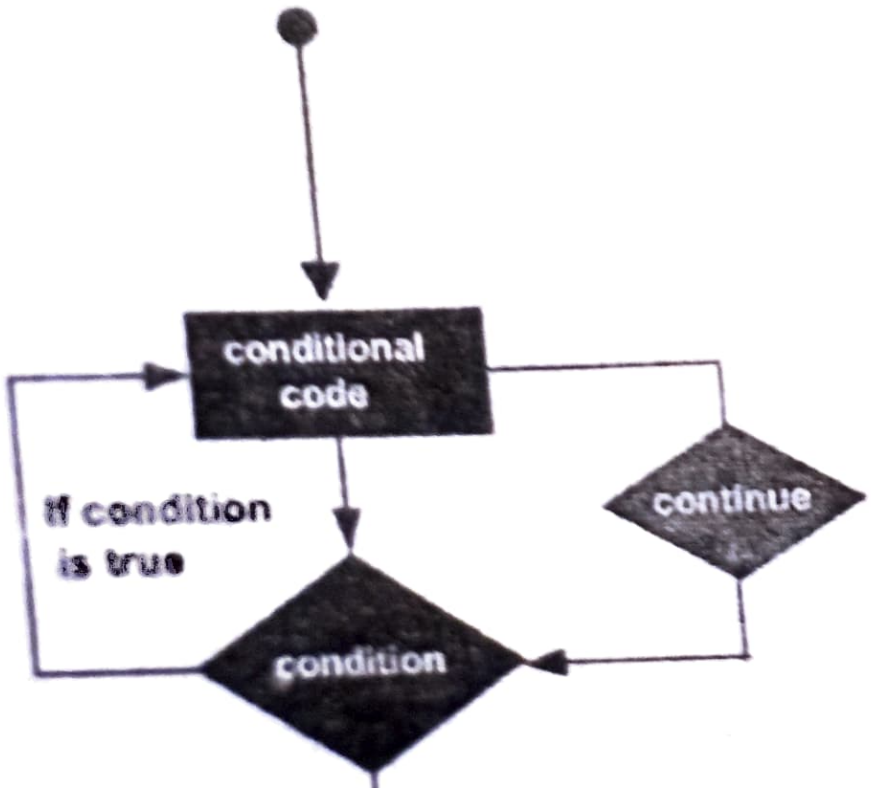
t :

l to go directly to the test-condition and then continue the l
tering continue, cursor leave the current cycle of loop, and s

nent works somewhat like the break statement. Instead of forc
er, continue forces the next iteration of the loop to take place, skipp
een.

ontinue causes the conditional test and increment portions of the lo
e while and do...while loops, program control passes to the conditio

continue statement in C++ is—



```
ocal variable declaration:
 a =10;

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

  return0;
  }
```

When the above code is compiled and executed, it produces the following resu

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

□□□

```
}
}while( a <20);

return0;
}
```

Q.22. Write About Continue Statement With Syntax And Example.

Ans. Continue statement :

It causes the control to go directly to the test-condition and then continue the lo process. On encountering continue, cursor leave the current cycle of loop, and st with the next cycle.

The continue statement works somewhat like the break statement. Instead of forci termination, however, continue forces the next iteration of the loop to take place, skippi any code in between.

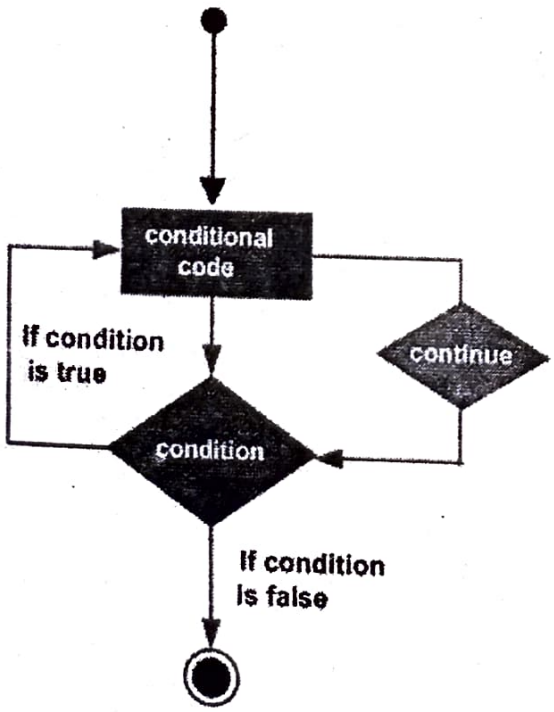
For the for loop, continue causes the conditional test and increment portions of the lo to execute. For the while and do...while loops, program control passes to the condition tests.

Syntax :

The syntax of a continue statement in C++ is—

```
continue;
```

Flow Diagram :



Example :

```
#include<iostream>
usingnamespace std;

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

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

return0;
}
```

When the above code is compiled and executed, it produces the following result—

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

□□□

Q.1. What is functions in C++ ? Why it is mostly used ?

Ans. Functions are used to provide modularity to a program. Creating an application usi function makes it easier to understand, edit, check errors etc.You can divide up yo code into separate functions. How you divide up your code among different functio is up to you, but logically the division usually is such that each function performs specific task.

A function **declaration** tells the compiler about a function's name, return type, a parameters. A function **definition** provides the actual body of the function.The C+ standard library provides numerous built-in functions that your program can call. F example, function **strcat()** to concatenate two strings, function **memcpy()** to copy o memory location to another location and many more functions.A function is known wi various names like a method or a sub-routine or a procedure etc.

Q.2. How can we define a function ? Explain each and every part of the definitio

Ans. The general form of a C++ function definition is as follows—

```
return_type function_name( parameter list ) {
body of the function
}
```

A C++ function definition consists of a function header and a function body. Here all the parts of a function "

- **Return Type** " A function may return a value. The **return_type** is the data type of value the function returns. Some functions perform the desired operations with returning a value. In this case, the return_type is the keyword **void**.
- **Function Name** " This is the actual name of the function. The function name and parameter list together constitute the function signature.

placeholder. When a function is invoked, you pass parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

Function Body : The function body contains a collection of statements that define what the function does.

Q. 3. Write A Program By Using Function To Find Out The Maximum Value Between Two Numbers.Example.

Ans.

```
// function returning the max between two numbers

int max(int num1, int num2) {
   // local variable declaration
   int result;

   if (num1 > num2)
result = num1;
   else
result = num2;

   return result;
}
```

Q.4. How can you call a function ? Also explain one method of calling a function.

(AKU. 2014)

Ans. While creating a C++ function, you give a definition of what the function has to do. To use a function, you will have to call or invoke that function.When a program calls a function, program control is transferred to the called function. A called function performs defined task and when it's return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.

Calling a Function :

Functions are called by their names. If the function is without argument, it can be called directly using its name. But for functions with arguments, we have two ways to call them,

1. Call by Value
2. Call by Reference
3. Call by Value

In this calling technique we pass the values of arguments which are stored or co into the formal parameters of functions. Hence, the original values are unchanged the parameters inside function changes.

The **call by value** method of passing arguments to a function copies the actual of an argument into the formal parameter of the function. In this case, changes to the parameter inside the function have no effect on the argument. By default, uses **call by value** to pass arguments. In general, this means that code within a fun cannot alter the arguments used to call the function and above mentioned example calling max() function used the same method.By default, C++ uses call by value to arguments. In general, this means that code within a function cannot alter the argu used to call the function.

Q.5. Write A Program of Swapping of Two Numbers By Using Call By Value Me

Ans.

```
#include <iostream>
using namespace std;
// function declaration
void swap(int x, int y);
int main () {
  // local variable declaration:
  int a = 100;
  int b = 200;
  cout << "Before swap, value of a :" << a << endl;
  cout << "Before swap, value of b :" << b << endl;
  // calling a function to swap the values.
  swap(a, b);
 cout << "After swap, value of a :" << a << endl;
  cout << "After swap, value of b :" << b << endl;
 return 0;
```

```
}
```

OUTPUT:

Before swap, value of a :100

Before swap, value of b :200

After swap, value of a :100

After swap, value of b :200

Q.6. Explain Call By Reference Method Of Function Calling. **(AKU. 2014)**

Ans. In this we pass the address of the variable as arguments. In this case the formal parameter can be taken as a reference or a pointer. in both the case they will change the values of the original variable.

```
void calc(int *p);
int main()
{
 int x = 10;
 calc(&x);       // passing address of x as argument
 printf("%d", x);
}void calc(int *p)
{
 *p = *p + 10;
}
```

Output : 20

Q.7. How in C++ function call by pointer ?

Ans. The **call by pointer** method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.

Q.8. What Do You Mean By Arguments Used In Function? Explain Formal Arguments.

Ans. If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters**of the function.

The formal parameters behave like other local variables inside the function and created upon entry into the function and destroyed upon exit.

Q.9. What Do You Mean By Member Functions In Classes? Also Write It's Ty

Ans. Member functions are the functions, which have their declaration inside the class defini and works on the data members of the class. The definition of member functions c be inside or outside the definition of class.

If the member function is defined inside the class definition it can be defined direc but if its defined outside the class, then we have to use the scope resolution :: opera along with class name alng with function name.

Types of Member Functions :

We already know what member functions are and what they do. Now lets study so special member functins present in the class. Following are different types of Mem functions :

1. Simple functions
2. Static functions
3. Const functions
4. Inline functions
5. Friend functions

Q.10. Explain Simple Member Function With Example.

Ans. Simple Member functions :

These are the basic member function, which dont have any special keyword like st etc as prefix. All the general member functions, which are of below given form, termed as simple and basic member functions.

```
return_type functionName(parameter_list)
{
function body;
}
```

Q.11. Explain Static Member Function With Example.

Ans. Static Member functions

Static is something that holds its position. Static is a keyword which can be used with data members as well as the member functions. We will discuss this in details later. As of now we will discuss its usage with member functions only.

A function is made static by using static keyword with function name. These functions work for the class as whole rather than for a particular object of a class.

It can be called using the object and the direct member access . operator. But, its more typical to call a static member function by itself, using class name and scope resolution :: operator.

Example :

```
class X
{
public:
static void f(){};
};

int main()
{
X::f();  // calling member function directly with class name
}
```

These functions cannot access ordinary data members and member functions, but only static data members and static member functions.

It doesn't have any "this" keyword which is the reason it cannot access ordinary members. We will study about "this" keyword later.

Q.12. Explain Const Member Function With Example.

Ans. Const Member functions :

We will study Const keyword in detail later, but as an introduction, Const keyword makes variables constant, that means once defined, there values can't be changed.

When used with member function, such member functions can never modify the object or its related data members.

//Basic Syntax of const Member Function

```
void fun() const {}
```

Q.13. Explain Inline Function With Proper Definition And Declaration. **(AKU. 201**

Ans. Inline Functions :

Inline functions are actual functions, which are copied everywhere during compilatio like preprocessor macro, so the overhead of function calling is reduced. All the function defined inside class definition are by default inline, but you can also make any non-clas function inline by using keyword **inline** with them.

For an inline function, declaration and definition must be done together. For example

```
inline void fun(int a)
{
  return a++;
}
```

about Inline Functions:

1. We must keep inline functions small, small inline functions have better efficiency.
2. Inline functions do increase efficiency, but we should not make all the functions inline. Because if we make large functions inline, it may lead to **code bloat**, and might affect the speed too.
3. Hence, it is adviced to define large functions outside the class definition using scope resolution ::operator, because if we define such functions inside class definition, then they become inline automatically.
4. Inline functions are kept in the Symbol Table by the compiler, and all the call for such functions is taken care at compile time.

Q.14. Write The Limitations Of Inline Functions. **(AKU. 2013)**

Ans. Limitations of Inline Functions :

1. Large Inline functions cause Cache misses and affect performance negatively.
2. Compilation overhead of copying the function body everywhere in the code on compilation which is negligible for small programs, but it makes a difference in large code bases.
3. Also, if we require address of the function in program, compiler cannot perform inlining on such functions. Because for providing address to a function, compiler will have to

allocate storage to it. But inline functions doesn't get storage, they are kept in Symbol table.

Q.15. What Is Friend Function In C++?Explain It With Example. **(AKU. 2013)**

Ans. Friend functions :

Friend functions are actually not class member function. Friend functions are made to give **private**access to non-class functions. You can declare a global function as friend, or a member function of other class as friend.

Example :

```
class WithFriend
{
 int i;
 public:
 friend void fun(); // Global function as friend
};
void fun()
{
 WithFriend wf;
 wf.i=10;  // Access to private data member
 cout << wf.i;
}
int main()
{
fun(); //Can be called directly
}
```

Hence, friend functions can access private data members by creating object of the class. Similarly we can also make function of other class as friend, or we can also make an entire class as friend class.

Q.16. Define Function Overloading ? And Also Explain Ways To Overload A Function. **(AKU 2015)**

Ans. Function Overloading :

If any class have multiple functions with same names but different parameters then they are said to be overloaded. Function overloading allows you to use the same name for different functions, to perform, either same or different functions in the same class.

Function overloading is usually used to enhance the readability of the program. If you have to perform one single operation but with different number or types of arguments then you can simply overload the function.

Ways to overload a function :

1. By changing number of Arguments.
2. By having different types of argument.

Number of Arguments different :

In this type of function overloading we define two functions with same names but different number of parameters of the same type. For example, in the below mentioned program we have made two sum() functions to return sum of two and three integers.

```
int sum (int x, int y)
{
 cout << x+y;
}

int sum(int x, int y, int z)
{
 cout << x+y+z;
}
```

Here sum() function is overloaded, to have two and three arguments. Which sum function will be called, depends on the number of arguments.

```
int main()
{
sum (10,20);  // sum() with 2 parameter will be called

sum(10,20,30);  //sum() with 3 parameter will be called
}
```

Q.17. Explain Virtual Function And Late Binding. **(AKU. 2013, 20**

Ans. Virtual Functions :

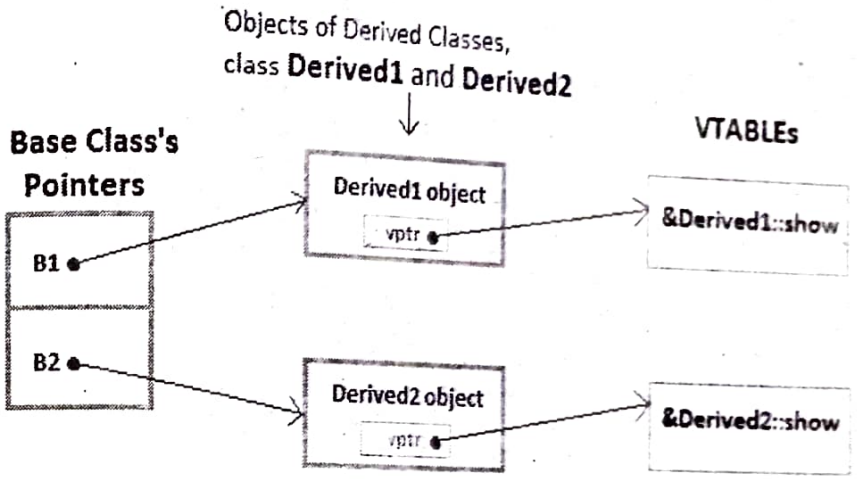
Virtual Function is a function in base class, which is overrided in the derived class, which tells the compiler to perform **Late Binding** on this function.

Virtual Keyword is used to make a member function of the base class Virtual.

Late Binding :

In Late Binding function call is resolved at runtime. Hence, now compiler determines the type of object at runtime, and then binds the function call. Late Binding is also called **Dynamic** Binding or **Runtime**Binding.

Mechanism of Late Binding :



vptr, is the vpointer, which points to the Virtual Function for that object.

VTABLE, is the table containing address of Virtual Functions of each class.

Important Points :

1. Only the Base class Method's declaration needs the **Virtual** Keyword, not the definition.
2. If a function is declared as **virtual** in the base class, it will be virtual in all its derived classes.
3. The address of the virtual Function is placed in the **VTABLE** and the copiler uses **VPTR**(vpointer) to point to the Virtual Function.

Q.18. Write A Program By Using Virtual Function.

Ans.

```
#include
using namespace std;
class A
{
   public:
virtual void show()
```

```
    {
        cout << "Base class\n";
    }
};

class B: public A
{
private:
virtual void show()
    {
        cout << "Derived class\n";
    }
};

int main()
{
    A *a;
    B b;
    a = &b;
    a -> show();
}
```

Output : Derived class

❑❑❑

Unit - 4

Classes And Data Abstraction

Structure In C++, Class, Build- In Operations On Classes, Assignment Operator And Classes, Class Scope, Reference Parameters And Class Objects (Variables), Member Functions, Accessor And Mutator Functions, Constructors, Default Constructor, Destructors. Lecture : 15

Q.1. Define Data Abstraction.

Ans. Data abstraction refers to providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

Data abstraction is a programming (and design) technique that relies on the separation of interface and implementation.

In C++, classes provides great level of data abstraction. They provide sufficient public methods to the outside world to play with the functionality of the object and to manipulate object data, i.e., state without actually knowing how class has been implemented internally. For example, your program can make a call to the sort() function without knowing what algorithm the function actually uses to sort the given values. In fact, the underlying implementation of the sorting functionality could change between releases of the library, and as long as the interface stays the same, your function call will still work.

Benefits of Data Abstraction :

Data abstraction provides two important advantages—

- Class internals are protected from inadvertent user-level errors, which might corrupt the state of the object.
- The class implementation may evolve over time in response to changing requirements or bug reports without requiring change in user-level code.

Q.2. Explain accessor member function. With an example.

Ans. An accessor is a member function that allows someone to retrieve the contents of a protected data member. For an accessor to perform its function, the following conditions must be met :

(1) The accessor must have the same type as the returned variable.

(2) The accessor does not need not have arguments.

(3) A naming convention must exist, and the name of the accessor must begin with the "Get" prefix.

The syntax of an accessor reduced to its simplest expression looks like this :

```
class MaClasse{
 private :
TypeDeMaVariableMaVariable;

 public :
TypeDeMaVariableGetMaVariable();

};

TypeDeMaVariableMaClasse::GetMaVariable(){
 return MaVariable;

}
```

In the above example, the accessor of the data member could be the following:

```
class Toto{
 private :
int age;

 public :
intGetAge();
};
```

```
int Toto::GetAge(){
return age;
}
```

Q.3. What is A Mutator Member Function ?

Ans. A mutator is a member function that allows for editing of the contents of a protected data member. For a mutator to accomplish its function, the following conditions must be present :

(1) As a parameter, it must have the value to be assigned to the data member. The parameter must be of the same type as the data member.

(2) The mutator does not need to return a value.

(3) A naming convention must exist, with the name of the accessor beginning with the "Set" prefix.

The syntax of a mutator reduced to its simplest expression looks like this :

```
class MaClasse{
 private :
TypeDeMaVariableMaVariable;

 public :
  void SetMaVariable(TypeDeMaVariable);
};
MaClasse::SetMaVariable(TypeDeMaVariableMaValeur){
MaVariable = MaValeur;
}
```

Q.4. What Is Constructor ? Explain The Types Of Constructor. **(AKU. 2013)**

Ans. Constructors are special class functions which performs initialization of every object. The Compiler calls the Constructor whenever an object is created. Constructorsiitialize values to object members after storage is allocated to the object.

```
class A
{
```

```
int x;
public:
A();  //Constructor
};
```

While defining a contructor you must remeber that the name of constructor will b same as the name of the class, and contructors never have return type.

Constructors can be defined either inside the class definition or outside class definitio using class name and scope resolution :: operator.

```
class A
{
inti;
public:
A(); //Constructor declared
};

A::A()  // Constructor definition
{
i=1;
}
```

Types of Constructors :

Constructors are of three types :

(1) Default Constructor

(2) Parametrized Constructor

(3) Copy COnstructor

(1) Default Constructor :

Default constructor is the constructor which doesn't take any argument. It ha parameter.

Syntax :

{ Constructor Definition }

Example :

```
class Cube
{
int side;
public:
Cube()
 {
side=10;
 }
};

int main()
{
Cube c;
cout<<c.side;
}
```

Output : 10

In this case, as soon as the object is created the constructor is called which initializes its data members.

A default constructor is so important for initialization of object members, that even if we do not define a constructor explicitly, the compiler will provide a default constructor implicitly.

```
class Cube
{
public:
int side;
};
```

```
int main()
{
 Cube c;
cout<<c.side;
}
```

Output : 0

In this case, default constructor provided by the compiler will be called which will initialize the object data members to default value, that will be 0 in this case.

(2) Parameterized Constructor :

These are the constructors with parameter. Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.

Example :

```
class Cube
{
public:
int side;
Cube(int x)
  {
side=x;
  }
};

int main()
{
 Cube c1(10);
 Cube c2(20);
 Cube c3(30);
```

```
cout<< c1.side;
cout<< c2.side;
cout<< c3.side;
}
```

OUTPUT : 10 20 30

By using parameterized construcor in above case, we have initialized 3 objects with user defined values. We can have any number of parameters in a constructor.

(3) **Copy Constructor :**

These are special type of Constructors which takes an object as argument, and is used to copy values of data members of one object into other object. We will study copy constructors in detail later.

Q.5. Explain Copy Constructor With An Example. **(AKU. 2012)**

Ans. A copy constructor is a member function which initializes an object using another object of the same class. A copy constructor has the following general function prototype:

ClassName (constClassName&old_obj);

Following is a simple example of copy constructor.

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

class Point
{
private:  int x, y;
public:
Point(int x1, int y1) { x = x1; y = y1; }
// Copy constructor
Point(const Point &p2) {x = p2.x; y = p2.y; }
 intgetX()        { return x; }
 intgetY()        { return y; }}; int main()
{
```

```
Point p1(10, 15); // Normal constructor is called here
Point p2 = p1; // Copy constructor is called here
// Let us access values assigned by constructors
cout<< "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
cout<< "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();
return 0;}
```

Run on IDE

Output:

p1.x = 10, p1.y = 15

p2.x = 10, p2.y = 15

The copy constructor is used to—

- Initialize one object from another of the same type.
- Copy an object to pass it as an argument to a function.
- Copy an object to return it from a function.

Q.6. Explain the concept of destructor in class. What is the role in terms of cleanu of unwanted objects ? (AKU. 201

Ans. Destructors :

Destructor is a special class function which destroys the object as soon as the scop of object ends. The destructor is called automatically by the compiler when the objec goes out of scope.

The syntax for destructor is same as that for the constructor, the class name is use for the name of destructor, with a tilde ~ sign as prefix to it.

```
class A
{
public:
 ~A();
};
```

Destructors will never have any arguments.

.7. Create a class complex and implements the following. (AKU 20

ns.

Constructor And Destructor

```
class A
{
A()
 {
cout<< "Constructor called";
 }

~A()
 {
cout<< "Destructor called";
 }
};

int main()
{
 A obj1;   // Constructor Called
int x=1
if(x)
  {
   A obj2;  // Constructor Called
  }   // Destructor Called for obj2
} //  Destructor called for obj1
```

8. Can we overload constructor and destructor ? Justify with the help of Example. **(AKU. 2012)**

s. // Program to define in different way about the overload constructors

```
#include<iostream.h>
#include<conio.h>
class integer
{
    private:
    int a;
    public:
```

```
        integer()
    {
        a=0;
        cout<<endl<<"Constructor Called. : ";
    }
        integer(inti)
    {
        a=i;
        cout<<endl<<"Constructor with argument called. : ";
    }
        ~integer()
    {
      cout<<endl<<"Destructor Called.";
    }
      void get()
    {
       cout<<endl<<"Enter the value of integer :" <<endl;
       cin>>a;
    }
        void show()
    {
       cout<<endl<<"The value of integer is :"<<"\t"<<a;
    }
};
    void main()
  {
    clrscr();
    integer x, y(10);
    x.get();
    x.show();
    y.show();
    getch();
  }
```

Q.9. Explain default constructor and also it's advantages. **(AKU. 2012)**

Ans. A Default constructor is that will either have no parameters, or all the parameters have default values. If no constructors are available for a class, the compiler implicitly creates a default parameterless constructor without a constructor initializer and a null body.

Example :

```
#include <iostream.h>
class Defal
{
public:
int x;
int y;
Defal(){x=y=0;}
}
;int main()
{
Defal A;
cout<< "Default constructs x,y value::"<<
A.x<<", "<<A.y<< "\n";
return 0;
}
```

Result:Default constructs x,y value:: 0,0

Q.10. Explain Constructor And Destructor With Multilevel Inheritance Using C++. **(AKU. 2014)**

Ans. If we create Constructors in All Levels in Multilevel Inheritance then the order of execution of Constructors is Same As the level

That is First The Base then Base 1 and Then Derived Class Constructor is Called

Destructors Follows the reverse order of inheritace

For multiple inheritance order of constructor call is, the base class's constructors are called in the order of inheritance and then the derived class's constructor.

```
// C++ program to show the order of constructor calls
// in Multiple Inheritance
#include <iostream>
usingnamespacestd;
// first base classclass
Parent1
{
public:
// first base class's Constructor
Parent1()
{
cout<< "Inside first base class"<<endl;
}
};
// second base class
classParent2
{
public:
// second base class's Constructor
Parent2()
{
    cout<< "Inside second base class"<<endl;
}
};
// child class inherits Parent1 and Parent2
classChild : publicParent1, publicParent2
{
public:
```

```
// child class's Constructor
Child()
{
cout<< "Inside child class"<<endl;
}
};
// main functionintmain()
{
// creating object of class Child
Child obj1;
return0;
}
```

Run on IDE

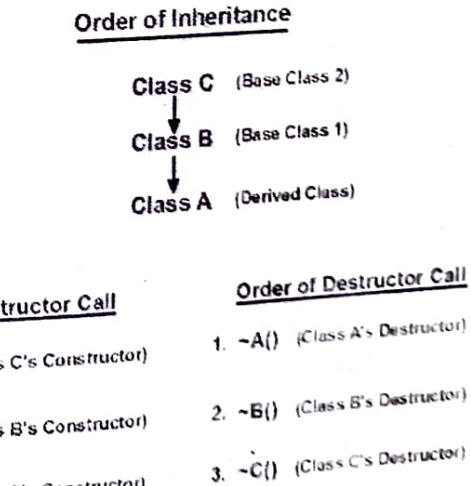
Output:

Inside first base class

Inside second base class

Inside child class

Order of constructor and Destructor call for a given order of Inheritance



Q.11. What Is Dynamic Constructor ?Explain With Example. **(AKU. 2014)**

Ans.

Dynamic Constructor

Dynamic constructor is used to allocate the memory to the objects at the run time.

- Memory is allocated at run time with the help of 'new' operator.
- By using this constructor, we can dynamically initialize the objects.

By using this constructor, we can dynamically initialize the objects.

Example:

```
class DynamicCon
{
        int * ptr;
        public:
        DynamicCon()
        {
          ptr=new int;
         *ptr=100;
        }
        DynamicCon(int v)
        {
          ptr=new int;
         *ptr=v;
        }

         int display()
        {
          return(*ptr);
        }

};

void main()
{

          DynamicCon obj1, obj2(90);
          cout<<"\nThe value of obj1's ptr  is:";
          cout<<obj1.display();
          cout<<"\nThe value of obj2's ptr is:"<<endl;
          cout<<obj2.display();
}
```

Output :

The value of obj1's ptr is:100

The value of obj2's ptr is: 90

Q.12. Explain Overloaded Constructor With An Example.

Ans. Constructor Overloading : (AKU : 2014)

Constructor can be overloaded in a similar way as function overloading. Overloaded constructors have the same name (name of the class) but different number of arguments.

Depending upon the number and type of arguments passed, specific constructor is called. Since, there are multiple constructors present, argument to the constructor should also be passed while creating an object.

Constructors Overloading are used to increase the flexibility of a class by having more number of constructor for a single class. By have more than one way of initializing objects can be done using overloading constructors.

Example :

```
#include <iostream.h>
class Overclass
{
public:
int x;
int y;
Overclass() { x = y = 0; }
Overclass(int a) { x = y = a; }
Overclass(int a, int b) { x = a; y = b; }
};
int main()
{
Overclass A;
Overclass A1(4);
Overclass A2(8, 12);
cout<< "Overclass A's x,y value:: "<<
A.x<< " , "<<A.y<< "\n";
cout<< "Overclass A1's x,y value:: "<<
A1.x << " ,"<< A1.y << "\n";
```

```
cout<< "Overclass A2's x,y value:; "<<
A2.x << " , "<< A2.y << "\n";
return 0;
}
```

Result :

Overclass A's x,y value:: 0 ; 0

Overclass A1's x,y value:: 4 ,4

Overclass A2's x,y value:; 8 , 12

Q.13. ExplainVirtul Destructor. (AKU. 2015)

Ans. Virtual Destructors :

Destructors in the Base class can be Virtual. Whenever Upcasting is done, Destructors of the Base class must be made virtual for proper destrucstion of the object when the program exits.

NOTE : Constructors are never Virtual, only Destructors can be Virtual.

Pure Virtual Destructors :

- Pure Virtual Destructors are legal in C++. Also, pure virtual Destructors must be defined which is against the pure virtual behaviour.
- The only difference between Virtual and Pure Virtual Destructor is, that pure virtual destructor will make its Base class Abstract, hence you cannot create object of that class.
- There is no requirement of implementing pure virtual destructors in the derived classes

```
class Base
{
public:
virtual ~Base() = 0;      //Pure Virtual Destructor
};

Base::~Base() { cout<< "Base Destructor"; } //Definition of Pure Virtual Destruct

classDerived:public Base
{
public:
 ~Derived() { cout<< "Derived Destructor"; }
};
```

❑❑❑

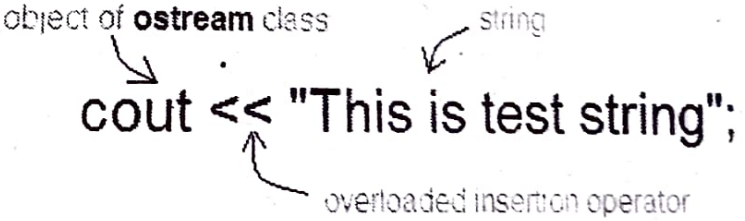
Unit - 5

Overloading And Templates

Operator Overloading, Function Overloading, Function Templates, Class Templates. Lecture : 5

Q.1. Explain operator overloading with syntax. **(AKU. 2012)**

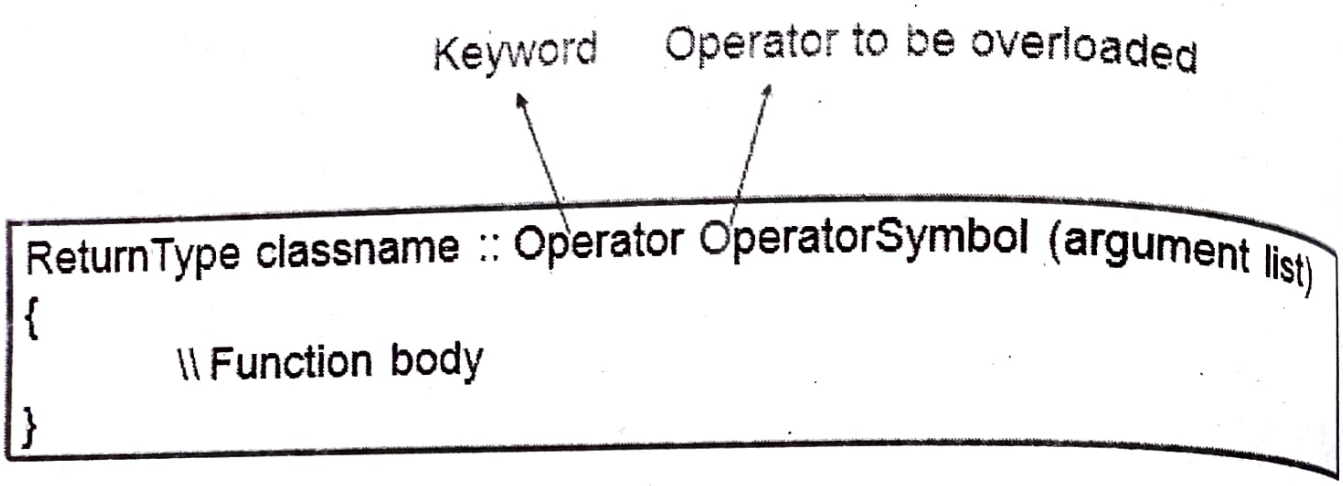
Ans. Operator overloading is an important concept in C++. It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it. Overloaded operator is used to perform operation on user-defined data type. For example '+' operator can be overloaded to perform addition on various data types, like for Integer, String(concatenation) etc.



Almost any operator can be overloaded in C++. However there are few operator which can not be overloaded. Operator that are not overloaded are follows

- scope operator - ::
- sizeof
- member selector - .
- member pointer selector - *
- ternary operator - ?:

Operator Overloading Syntax



Q.2. How Can You Implement Operator Overloading ? **(AKU. 2012)**

Ans. Operator overloading can be done by implementing a function which can be :

1. Member Function
2. Non-Member Function
3. Friend Function

Operator overloading function can be a member function if the Left operand is an Object of that class, but if the Left operand is different, then Operator overloading function must be a non-member function.

Operator overloading function can be made friend function, if it needs access to the private and protected members of class.

Q.3. What is the restrictions on operator overloading ?

Ans. Following are some restrictions to be kept in mind while implementing operator overloading.

1. Precedence and Associativity of an operator cannot be changed.
2. Arity (numbers of Operands) cannot be changed. Unary operator remains unary, binary remains binary etc.
3. No new operators can be created, only existing operators can be overloaded.
4. Cannot redefine the meaning of a procedure. You cannot change how integers are added.

Q.4. How can overloading on arithmetic operator is performed?

Ans. Arithmetic operator are most commonly used operator in C++. Almost all arithmetic operator can be overloaded to perform arithmetic operation on user-defined data type.

In the below example we have overridden the + operator, to add to Time(hh:mm:ss) objects.

Example:

overloading '+' Operator to add two time object

```
#include<iostream.h>
#include<conio.h>
class time
{
inth,m,s;
public:
time()
 {
  h=0, m=0; s=0;
 }
voidgetTime();
void show()
 {
cout<< h<< ":"<< m<< ":"<< s;
 }
time operator+(time); //overloading '+' operator
};
time time::operator+(time t1)
//operator function
{
time t;
inta,b;
a=s+t1.s;
t.s=a%60;
```

```
b=(a/60)+m+t1.m;
t.m=b%60;
t.h=(b/60)+h+t1.h;
t.h=t.h%12;
return t;
}
void time::getTime()
{
cout<<"\n Enter the hour(0-11) ";
cin>>h;
cout<<"\n Enter the minute(0-59) ";
cin>>m;
cout<<"\n Enter the second(0-59) ";
cin>>s;
}
void main()
{
clrscr();
time t1,t2,t3;
cout<<"\n Enter the first time ";
t1.getTime();
cout<<"\n Enter the second time ";
t2.getTime();
 t3=t1+t2;
//adding of two time object using '+' operator
 cout<<"\n First time ";
 t1.show();
 cout<<"\n Second time ";
```

```
t2.show();
cout<<"\n Sum of times ";
t3.show();
getch();
}
```

Q.5. How overloading on I/O operator is performed ?

- Overloaded to perform input/output for user defined datatypes.
- Left Operand will be of types ostream& and istream&
- Function overloading this operator must be a Non-Member function because left operand is not an Object of the class.
- It must be a friend function to access private data members.

You have seen above that << operator is overloaded with ostream class object cout to print primitive type value output to the screen. Similarly you can overload << operator in your class to print user-defined type to screen. For example we will overload << in time class to display time object using cout.

```
time t1(3,15,48);
cout<< t1;
```

Q.6. Differentiate Between Copy Constructor Vs. Assignment Operator. (AKU 2012)

Assignment operator is used to copy the values from one object to another already existing object. For example

```
time tm(3,15,45);
//tm object created and initialized
time t1;
//t1 object created
t1 = tm;
//initializing t1 using tm
```

Copy constructor is a special constructor that initializes a new object from an existing object.

```
time tm(3,15,45);
//tm object created and initialized
```

time t1(tm);

//t1 object created and initialized using tm object

Q.7. Explain Function Overloadingwith Example.

Ans. If any class have multiple functions with same names but different parameters then they are said to be overloaded. Function overloading allows you to use the same name for different functions, to perform, either same or different functions in the same class.

Function overloading is usually used to enhance the readability of the program. If you have to perform one single operation but with different number or types of arguments then you can simply overload the function.

Q.8. Write The Ways To Overload A Function.

Ans. Ways to overload a function :

1. By changing number of Arguments.
2. By having different types of argument.

Number of Arguments different :

In this type of function overloading we define two functions with same names but differe number of parameters of the same type. For example, in the below mentioned progra we have made two sum() functions to return sum of two and three integers.

```
int sum (int x, int y)
{
cout<<x+y;
}

int sum(int x, int y, int z)
{
cout<<x+y+z;
}
```

Here sum() function is overloaded, to have two and three arguments. Which su function will be called, depends on the number of arguments.

```
int main()
```

```
{
sum (10,20); // sum() with 2 parameter will be called
sum(10,20,30); //sum() with 3 parameter will be called
```

Q.10. Explain C++ templates with proper syntax.

Ans. Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.

A template is a blueprint or formula for creating a generic class or a function. The library containers like iterators and algorithms are examples of generic programming and have been developed using template concept.

There is a single definition of each container, such as vector, but we can define many different kinds of vectors for example, vector <int> or vector <string>.

You can use templates to define functions as well as classes, let us see how they work.

Q.11. Explain function template with syntax and example. **(AKU. 2012)**

Ans. The general form of a template function definition is shown here—

```
template<class type> ret-type func-name(parameter list)
{
  // body of function
}
```

Here, type is a placeholder name for a data type used by the function. This name can be used within the function definition.

For example, to create a template function that returns the greater one of two objects we could use:

```
1 template <class myType>
2 myTypeGetMax (myType a, myType b) {
3 return (a>b?a:b);
4 }
```

Q.12. Write a program of a function template that returns the maximum of two values.

Ans.

```
#include<iostream>
```

```
#include<string>
usingnamespacestd;
template<typename T>
inline T const&Max(T const& a, T const& b){
return a < b ?b:a;
}
int main (){
inti=39;
int j =20;
cout<<"Max(i, j): "<<Max(i, j)<<endl;
double f1 =13.5;
double f2 =20.7;
cout<<"Max(f1, f2): "<<Max(f1, f2)<<endl;
string s1 ="Hello";
string s2 ="World";
cout<<"Max(s1, s2): "<<Max(s1, s2)<<endl;
return0;
}
```

If we compile and run above code, this would produce the following result—

```
Max(i, j): 39
Max(f1, f2): 20.7
Max(s1, s2): World
```

Q.13. Explain class templates with.

Ans. We also have the possibility to write class templates, so that a class can ha members that use template parameters as types. For example:

```
1 template<class T>
2 classmypair {
3 T values [2];
```

```
4 public:
5 mypair (T first, T second)
6 {
7 values[0]=first; values[1]=second;
8 }
9 };
```

The class that we have just defined serves to store two elements of any valid type. For example, if we wanted to declare an object of this class to store two integer values of type int with the values 115 and 36 we would write :

```
mypair<int>myobject (115, 36);
```

□□□

Inheritance

Single and Multiple Inheritance, virtual Base class, Abstract Class, Pointer and Inheritance, Pointers and Arrays : Void Pointers, Pointer to Class, Pointer to Object, The this Pointer, Void Pointer, Arrays. (Lecture : 6)

Q.1. Define inheritance. With an example. **(AKU. 2013, 2014)**

Ans. Inheritance is the capability of one class to acquire properties and characteristics from another class. The class whose properties are inherited by other class is called the Parent or Base or Super class. And, the class which inherits properties of other class is called Child or Derived or Sub class.

Inheritance makes the code reusable. When we inherit an existing class, all its methods and fields become available in the new class, hence code is reused.

Example: "She had inherited the beauty of her mother"

Inheritance in Object Oriented Programming can be described as a process of creating new classes from existing classes.

Purpose of Inheritance :

Code Reusability, Method Overriding (Hence, Runtime Polymorphism.), Use of Virtual Keyword

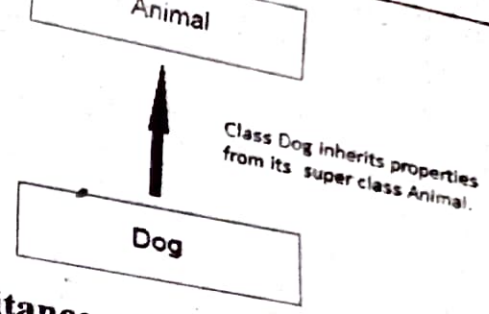
Basic Syntax of Inheritance

class Subclass_name : access_mode Superclass_name

While defining a subclass like this, the super class must be already defined or atleas declared before the subclass declaration.

Access Mode is used to specify, the mode in which the properties of superclass wi be inherited into subclass, public, privtate or protected.

Example of Inheritance



Q.2. Write the types of inheritance and explain each. **(AKU. 2013, 2014)**

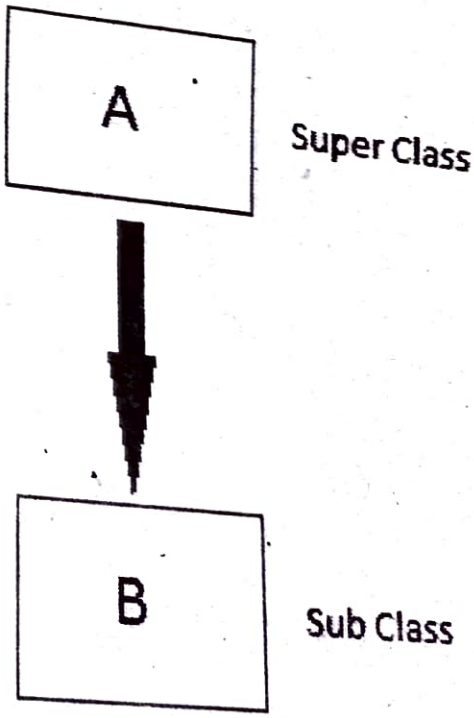
Ans. Types of Inheritance :

In C++, we have 5 different types of Inheritance. Namely,

1. Single Inheritance
2. Multiple Inheritance
3. Hierarchical Inheritance
4. Multilevel Inheritance
5. Hybrid Inheritance (also known as Virtual Inheritance)

Q.3. Explain single inheritance with example. Single Inheritance. (2013)

In this type of inheritance one derived class inherits from only one base class. It is the most simplest form of Inheritance.



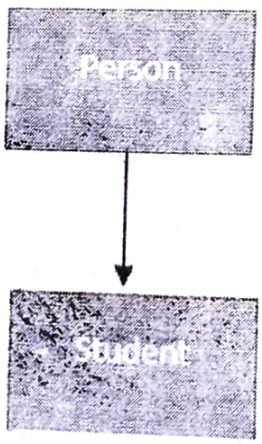
Example of Single Inheritance

```
//base class
classPerson
{
public:
```

```
    Person(string szName,int iYear)
    {
        m_szLastName = szName;
        m_iYearOfBirth = iYear;
    }
    string m_szLastName;
    int m_iYearOfBirth;
    voidprint()
    {
        cout <<"Last name: "<< szLastName << endl;
        cout <<"Year of birth: "<< iYearOfBirth << endl;
    }
    protected:
    string m_szPhoneNumber;
    };
```

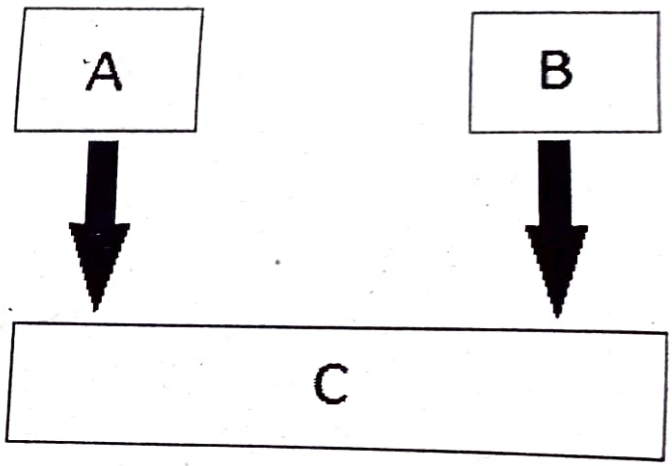
We want to create new class Student which should have the same information as Person class plus one new information about university. In this case, we can create a derived class Student:

```
//derived class
class Student:public Person
{
public:
    string m_szUniversity;
};
```



Q.3. Explain Multiple Inheritance with example. **(AKU. 2013)**

Ans. In this type of inheritance a single derived class may inherit from two or more than two base classes.



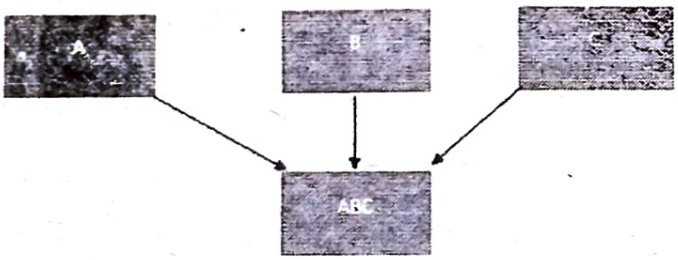
Example of Multiple Inheritance

```
class A
    {
    int m_iA;
    A(int iA):m_iA(iA)
    {
    }
};

class B
{
    int m_iB;
    B(int iB):m_iB(iB)
    {
    }
};

class C
```

```
{
    int m_iC;
    C(int iC):m_iC(iC)
    {
    }
};
```

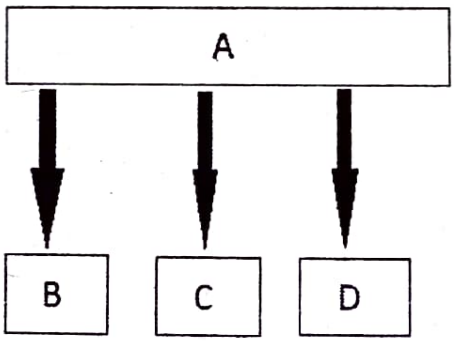
You can create a new class that will inherit all the properties of all these classes:

```
class ABC :public A,public B,public C
{
    int m_iABC;
    //here you can access m_iA, m_iB, m_iC
};
```



Q.4. Explain Hierarchical Inheritance with a program.

Ans. In this type of inheritance, multiple derived classes inherits from a single base class.

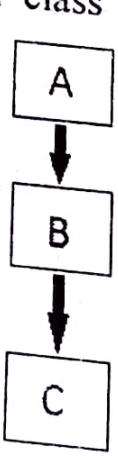



```
class base
{
    //content of base class
};

class derived1 :public base
{
```

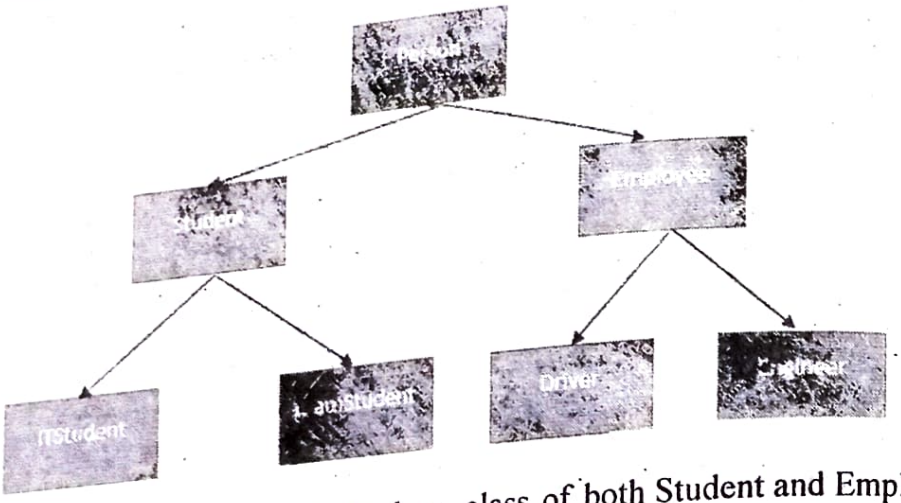
```
        //content of derived1
    };

    class derived2 :public base
    {
        //content of derived
    };

    class derived3 :public base
    {
        //content of derived3
    };

    class derived4 :public base
    {
        //content of derived4
    };
```

Q.5. Explain Multilevel Inheritance with a program.

Ans. In this type of inheritance the derived class inherits from a class, which in turn inherit from some other class. The Super class for one, is sub class for the other.



Below Image shows the example of multilevel inheritance



As you can see, Class Person is the base class of both Student and Employee classes. At the same time, Class Student is the base class for IT Student and Math Student classes. Employee is the base class for Driver and Engineer classes.

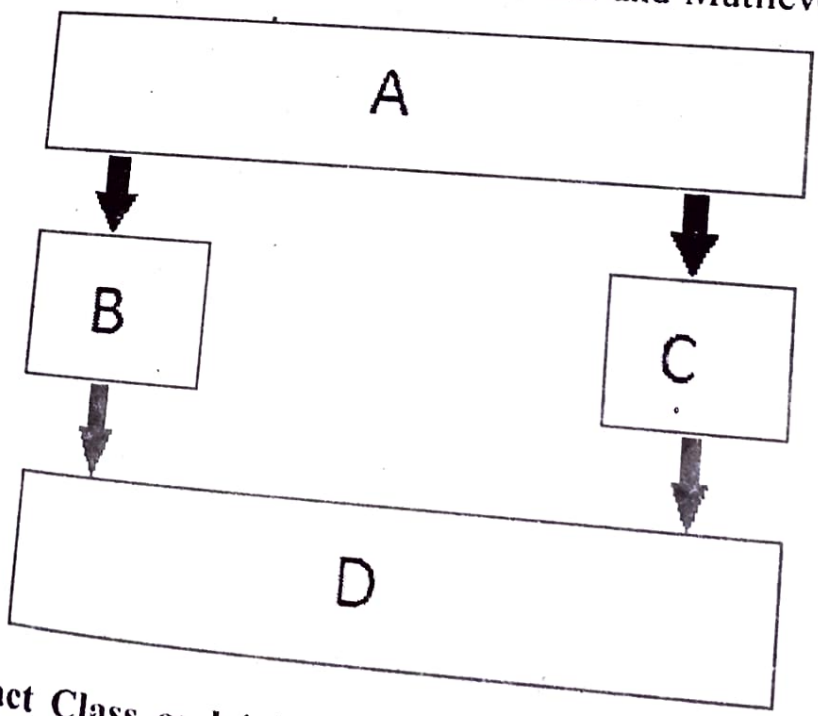
The code for above example of multilevel inheritance will be as shown below

```
classPerson

{
    //content of class person
};

classStudent:publicPerson
{
    //content of Student class
};

classEmployee:publicPerson
{
    //content of Employee class
};

classITStundet:publicStudent
{
```

```
        //content of ITStudent class
    };

    classMathStundet:publicStudent
    {
        //content of MathStudent class
    };

    classDriver:publicEmployee
    {
        //content of class Driver
    };

    classEngineer:publicEmployee
    {
        //content of class Engineer
};
```

Explain Hybrid (Virtual) Inheritance with an example.

Hybrid Inheritance is combination of Hierarchical and Mutilevel Inheritance.



ain Abstract Class and it's characteristics.

act Class is a class which contains atleast one Pure Virtual function in it. Abstract

classes are used to provide an Interface for its sub classes. Classes inheriting an Abstract Class must provide definition to the pure virtual function, otherwise they will also become abstract class.

Characteristics of Abstract Class :

1. Abstract class cannot be instantiated, but pointers and refrences of Abstract class type can be created.
2. Abstract class can have normal functions and variables along with a pure virtual function.
3. Abstract classes are mainly used for Upcasting, so that its derived classes can use its interface.
4. Classes inheriting an Abstract Class must implement all pure virtual functions, or else they will become Abstract too.

Example of Abstract Class

```
class Base              //Abstract base class
{
public:
 virtual void show() = 0;              //Pure Virtual Function
};

class Derived:public Base
{
public:
 void show()
 { cout << "Implementation of Virtual Function in Derived class"; }
};

int main()
{
 Base obj;        //Compile Time Error
 Base *b;
 Derived d;
 b = &d;
```

```
    b->show();
}
```

Output :

Q.8. What is Pure Virtual Functions ? **(AKU. 2014, 2017)**

Ans. Pure virtual Functions are virtual functions with no definition. They start with virtual keyword and ends with = 0. Here is the syntax for a pure virtual function,

```
virtual void f() = 0;
```

Q.9. What is a virtual base class ?

Ans. An ambiguity can arise when several paths exist to a class from the same base class. This means that a child class could have duplicate sets of members inherited from a single base class.

C++ solves this issue by introducing a virtual base class. When a class is made virtual, necessary care is taken so that the duplication is avoided regardless of the number of paths that exist to the child class.

Q.10 What is Virtual base class? Explain its uses.

Ans. When two or more objects are derived from a common base class, we can prevent multiple copies of the base class being present in an object derived from those objects by declaring the base class as virtual when it is being inherited. Such a base class is known as virtual base class. This can be achieved by preceding the base class' name with the word virtual.

Q.11. What is Pointers ?

Ans. A pointer is a variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is –

```
type *var-name;
```

Here, type is the pointer's base type; it must be a valid C++ type and var-name is the name of the pointer variable. The asterisk you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration "

```
int    *ip;    // pointer to an integer
double *dp;    // pointer to a double
float  *fp;    // pointer to a float
```

```
char   *ch      // pointer to character
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

```
#include<iostream>
usingnamespace std;
int main (){
intvar=20;// actual variable declaration.
int*ip;// pointer variable
  ip =&var;// store address of var in pointer variable
 cout <<"Value of var variable:";
   cout <<var<< endl;
// print the address stored in ip pointer variable
   cout <<"Address stored in ip variable:";
   cout << ip << endl;
// access the value at the address available in pointer
   cout <<"Value of *ip variable:";
   cout <<*ip << endl;
return0;
}
```

When the above code is compiled and executed, it produces result something as follows—

```
Value of var variable: 20
Address stored in ip variable: 0xbfc601ac
Value of *ip variable: 20
```

Q.12. Explain Null Pointers used in c++.

Ans. It is always a good practice to assign the pointer NULL to a pointer variable in case you do not have exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer. The NULL pointer is a constant with a value of zero defined in several standard libraries, including iostream.

Consider the following program—

```
#include <iostream>
using namespace std;
int main () {
  int  *ptr = NULL;
  cout << "The value of ptr is " << ptr ;
  return 0;
}
```

When the above code is compiled and executed, it produces the following result—

The value of ptr is 0

Q.13. **Differentiate between References vs Pointers.**

Ans. References are often confused with pointers but three major differences between references and pointers are—

You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.

Once a reference is initialized to an object, it cannot be changed to refer to another object. Pointers can be pointed to another object at any time.

A reference must be initialized when it is created. Pointers can be initialized at any time.

. **Explain this Pointer with an example.** **(AKU. 2013)**

Every object in C++ has access to its own address through an important pointer called this pointer. The this pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object. Friend functions do not have a this pointer, because friends are not members of a class. Only member functions have a this pointer.

. **What is array ? Explain it with an example array.** **(AKU. 2017)**

C++ provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last el

```
char   *ch      // pointer to character
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

```
#include<iostream>
usingnamespace std;
int main (){
intvar=20;// actual variable declaration.
int*ip;// pointer variable
  ip =&var;// store address of var in pointer variable
 cout <<"Value of var variable:";
   cout <<var<< endl;
// print the address stored in ip pointer variable
   cout <<"Address stored in ip variable:";
   cout << ip << endl;
// access the value at the address available in pointer
   cout <<"Value of *ip variable:";
   cout <<*ip << endl;
return0;
}
```

When the above code is compiled and executed, it produces result something as follows–

```
Value of var variable: 20
Address stored in ip variable: 0xbfc601ac
Value of *ip variable: 20
```

Q.12. Explain Null Pointers used in c++.

Ans. It is always a good practice to assign the pointer NULL to a pointer variable in case you do not have exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer. The NULL pointer is a constant with a value of zero defined in several standard libraries, including iostream. Consider the following program—

```
#include <iostream>
using namespace std;
int main () {
  int  *ptr = NULL;
  cout << "The value of ptr is " << ptr ;
 return 0;
}
```

When the above code is compiled and executed, it produces the following result—

The value of ptr is 0

Q.13. Differentiate between References vs Pointers.

Ans. References are often confused with pointers but three major differences between references and pointers are—

- You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.
- Once a reference is initialized to an object, it cannot be changed to refer to another object. Pointers can be pointed to another object at any time.
- A reference must be initialized when it is created. Pointers can be initialized at any time.

Q.14. Explain this Pointer with an example. (AKU. 2013)

Ans. Every object in C++ has access to its own address through an important pointer called this pointer. The this pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object. Friend functions do not have a this pointer, because friends are not members of a class. Only member functions have a this pointer.

Q.15. What is array ? Explain it with an example array. (AKU. 2017)

Ans. C++ provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.Arrays are

fixed in sizeAfter declaring an array, we cannot change the size of the array. That means we can't reduce the size nor increase the size of an array.

Elements of an array will be allocated contiguously in memory.When we create an array then each element of an array will be allocated in contiguous memory locations.Contiguous memory locations means that just after the first element of an array, second element will be present in the memory. And just after the second element, third element will be present, and so on.All the elements will be allocated in the memory locations back to back.The first element of an array will have the lowest address and the last element will have the highest address.

Declaring Arrays :

To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array as follows "

type arrayName [arraySize];

This is called a single-dimension array. The arraySize must be an integer constant greater than zero and type can be any valid C++ data type.

For example, to declare a 10-element array called balance of type double, use this statement—

double balance[10];

Q.16. Explain Array initialization with example.

Ans. There are several ways to initialize an array:

1. By using one statement with the square brackets :

   int anotherIntArray[3]={1,2,5,7};

2. If you initialize an array in this way, you will be able to omit the size of array in the declaration:

3. The size of array will be set automatically according to the number of elements. This array will be of size 4.

Q.17. Explain Multi-dimensional Arrays, how it is different from simple array ?

Ans. Two-Dimensional Arrays :

The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size x,y, you would write something as follows—

type arrayName [x][y];

Where type can be any valid C++ data type and arrayName will be a valid C++ identifier.

A two-dimensional array can be think as a table, which will have x number of rows and y number of columns. A 2-dimensional array a, which contains three rows and four columns can be shown as below—

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in array a is identified by an element name of the form a[i] [j], where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.

itializing Two-Dimensional Arrays :

Multidimensioned arrays may be initialized by specifying bracketed values for each row.

Following is an array with 3 rows and each row have 4 columns.

```
nt a[3][4] = {
  {0, 1, 2, 3} ,   /* initializers for row indexed by 0 */
  {4, 5, 6, 7} ,   /* initializers for row indexed by 1 */
  {8, 9, 10, 11}   /* initializers for row indexed by 2 */
};
```

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to previous example—

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements :

An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example—

```
int val = a[2][3];
```

The above statement will take 4th element from the 3rd row of the array. You can verify it in the above digram.

```
#include<iostream>
usingnamespace std;
```

```
int main (){
// an array with 5 rows and 2 columns.
int a[5][2]={{0,0},{1,2},{2,4},{3,6},{4,8}};

// output each array element's value
for(int i =0; i <5; i++)
for(int j =0; j <2; j++){

cout <<"a["<< i <<"]["<< j <<"]: ";
cout << a[i][j]<< endl;
}

return0;
}
```

When the above code is compiled and executed, it produces the following result–

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

As explained above, you can have arrays with any number of dimensions, although it is likely that most of the arrays you create will be of one or two dimensions.

□□□

Unit - 7

Exception Handling

The keywords try, throw and catch. Creating own Exception Classes, Exception Handling Techniques (Terminate the Program, Fix the Error and Continue, Log the Error and Continue), Stack

.1. What Is Exception Handling ? How are exceptions handled in C++? (AKU. 2013)

ns. An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch,and throw.

1) **throw** – A program throws an exception when a problem shows up. This is done using a throw keyword.

2) **catch** – A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

3) **try** – A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

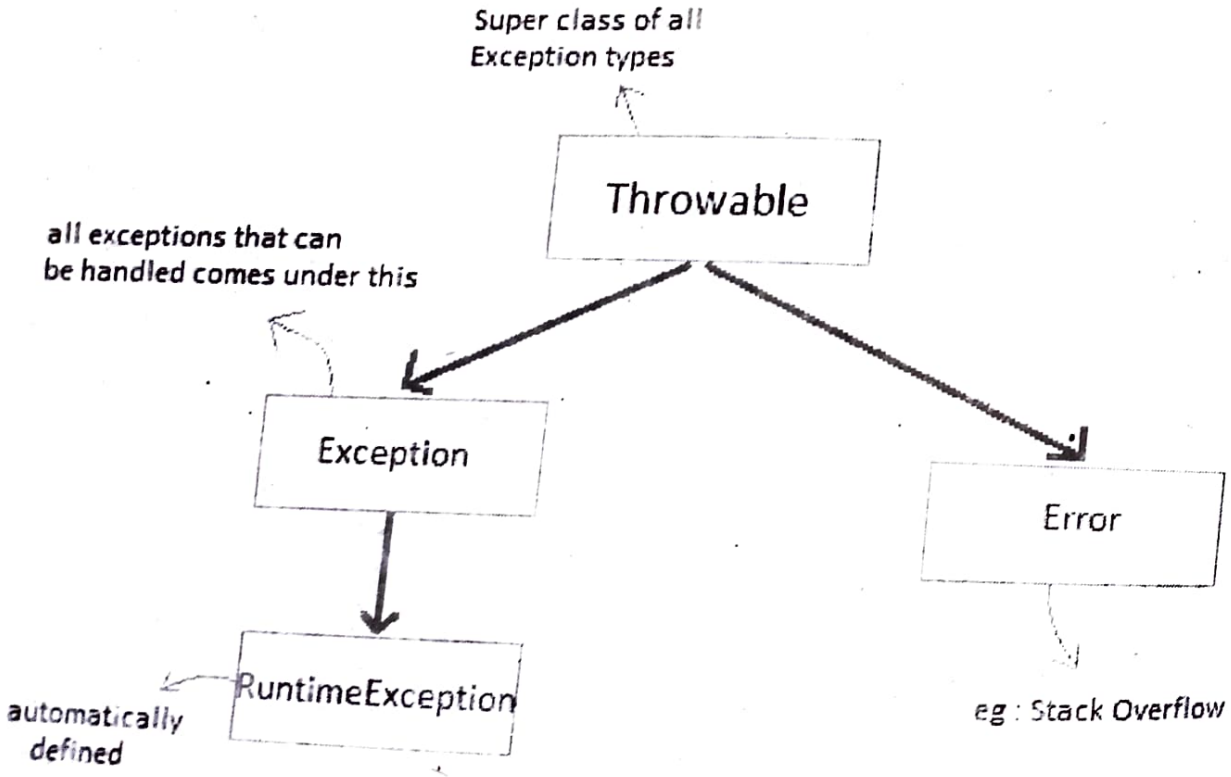
Assuming a block will raise an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch as follows –

```
try {
```

```
        // protected code
    } catch( ExceptionName e1 ) {
        // catch block
    } catch( ExceptionName e2 ) {
        // catch block
    } catch( ExceptionNameeN ) {
        // catch block
    }
```

Exception class Hierarchy :

All exception types are subclasses of class Throwable, which is at the top of exception class hierarchy.



Exception class is for exceptional conditions that program should catch. This class is extended to create user specific exception classes.

RuntimeException is a subclass of Exception. Exceptions under this class are automatically defined for programs.

Exceptions of type Error are used by the Java run-time system to indicate errors having to do with the run-time environment, itself. Stack overflow is an example of such an error.

Exception are categorized into 3 category :

- Checked Exception

The exception that can be predicted by the programmer at the compile time.

Example : File that need to be opened is not found. These type of exceptions must be checked at compile time.

- Unchecked Exception

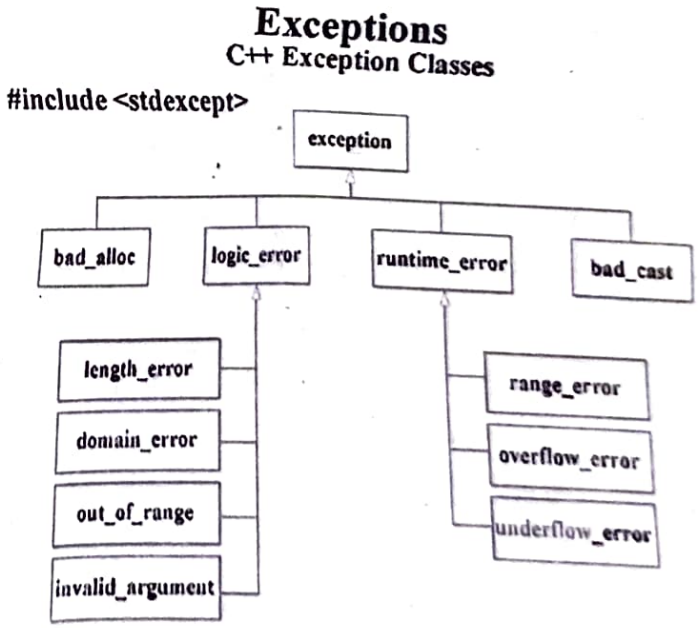
Unchecked exceptions are the class that extends RuntimeException. Unchecked exception are ignored at compile time.

Example : Arithmetic Exception, Null Pointer Exception, Array Index out of Bound exception. Unchecked exceptions are checked at runtime.

- Error

Errors are typically ignored in code because you can rarely do anything about an error.

Example :if stack overflow occurs, an error will arise. This type of error cannot be handled in the code.



Q.2. Explain the uses of Try, Throw and catch keywords, used for exception handling. (AKU. 2013)

Or,

Q. Write The Mechanism Of Exception Handling.

Ans. Exception Handling Mechanism :

Try, catch, throw

Using try and catch :

Try is used to guard a block of code in which exception may occur. This block of code is called guarded region. A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in guarded code, the catch block that follows the try is checked, if the type of exception that occured is listed in the catch block then the exception is handed over to the catch block which then handles it.

Example using Try and catch

```
classExcp
{
public static void main(String args[])
 {
inta,b,c;
try
  {
   a=0;
   b=10;
   c=b/a;
System.out.println("This line will not be executed");
  }
catch(ArithmeticException e)
  {
System.out.println("Divided by zero");
  }
System.out println("After exception is handled");
 }
}
```

Divided by zero

After exception is handled :

An exception will thrown by this program as we are trying to divide a number by zero inside try block. The program control is transferred outside try block. Thus the line "This line will not be executed" is never parsed by the compiler. The exception thrown is handled in catch block. Once the exception is handled, the program control is continue with the next line in the program i.e after catch block. Thus the line "After exception is handled" is printed.

- Multiple catch blocks :

A try block can be followed by multiple catch blocks. You can have any number of catch blocks after a single try block.If an exception occurs in the guarded code the exception is passed to the first catch block in the list. If the exception type of exception, matches with the first catch block it gets caught, if not the exception is passed down to the next catch block. This continue until the exception is caught or falls through all catches.

try statement can be nested inside another block of try. Nested try block is used when a part of a block may cause one error while entire block may cause another error. In case if inner try block does not have a catch handler for a particular exception then the outer try catch block is checked for match.throw Keyword

- ->throw keyword is used to throw an exception explicitly. Only object of Throwable class or its sub classes can be thrown. Program execution stops on encountering throw statement, and the closest catch statement is checked for matching type of exception.

Syntax :

throwThrowableInstance

Creating Instance of Throwable class

There are two possible ways to create an instance of class Throwable,

1. Using a parameter in catch block.
2. Creating instance with new operator.

newNullPointerException("test");

throws Keyword

Any method that is capable of causing exception[illegible]all the exceptions possibl

during its execution, so that anyone calling that method gets a prior knowledge about which exceptions are to be handled. A method can do so by using the throws keyword

Syntax :

```
typemethod_name(parameter_list)throwsexception_list
{
//definition of method
}
```

Q.3. Write The Difference Between Throw And Throws.

Ans. Difference between throw and throws

Throw	Throws
Throw keyword is used to throw an exception explicitly.	Throws keyword is used to declare an exception possible during its execution.
Throw keyword is followed by an instance of Throwable class or one of its sub-classes.	Throws keyword is followed by one or more Exception class names separated by commas.
Throw keyword is declared inside a method body.	Throws keyword is used with method signature (method declaration).
We cannot throw multiple exceptions using throw keyword.	We can declare multiple exceptions (separated by commas) using throws keyword.

Q.4. What Is Unexpected Exceptions?How It Can Be Handled ? **(AKU. 2014)**

Ans. Exception are categorized into 3 category.

Checked Exception

The exception that can be predicted by the programmer at the compile time.

Example : File that need to be opened is not found. These type of exceptions must be checked at compile time.

Unchecked Exception

...that extends RuntimeException. Unchecked exception