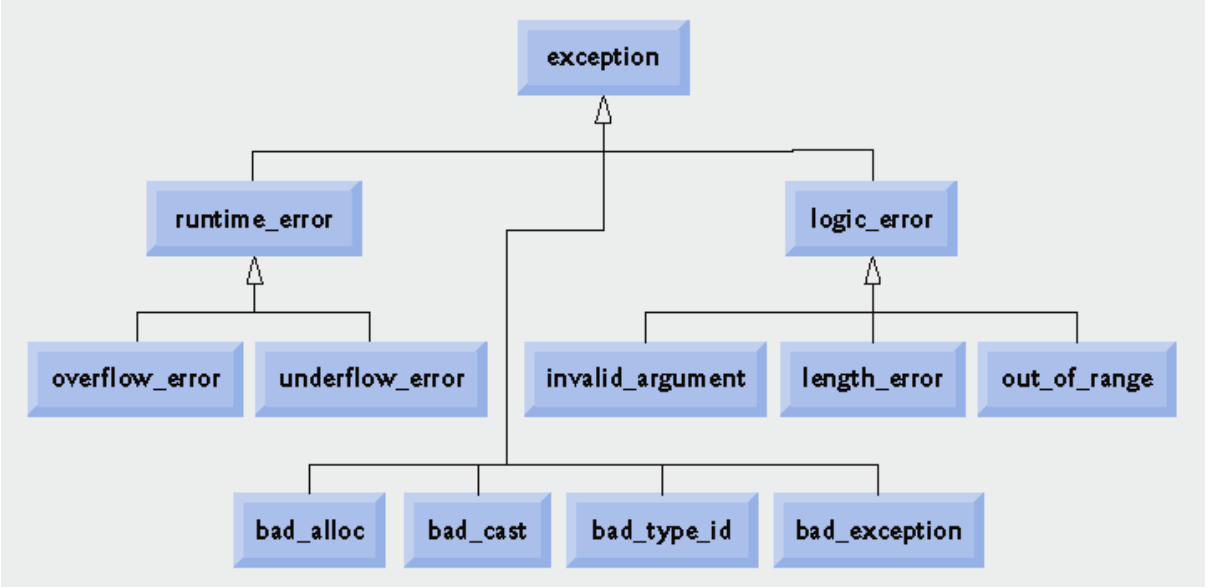
Abstract class – Exception handling - Standard libraries - Generic Programming - templates – class template - function template – STL – containers – iterators – function adaptors – allocators -Parameterizing the class - File handling concepts.

Standard Library exception



Exception	Description
std::exception	An exception and parent class of all the standard C++ exceptions.
std::bad_alloc	This can be thrown by **new**.
std::bad_cast	This can be thrown by **dynamic_cast**.
std::bad_exception	This is useful device to handle unexpected exceptions in a C++ program
std::bad_typeid	This can be thrown by **typeid**.
std::logic_error	An exception that theoretically can be detected by reading the code.
std::domain_error	This is an exception thrown when a mathematically invalid domain is used
std::invalid_argument	This is thrown due to invalid arguments.
std::length_error	This is thrown when a too big std::string is created
std::out_of_range	This can be thrown by the at method from for example a std::vector and std::bitset<>::operator[]().
std::runtime_error	An exception that theoretically can not be detected by reading the code.
std::overflow_error	This is thrown if a mathematical overflow occurs.
std::range_error	This is occured when you try to store a value which is out of range.
std::underflow_error	This is thrown if a mathematical underflow occurs.

EXCEPTION HANDLING

```
// single throw – single catch
#include<iostream.h>
#include<conio.h>
#include<exception.h>
#include<math.h>
class MyException
```

```
{
int n;
public:
MyException(int a)
{
n=a;
}
int Divide(int x)
{
try
{
if(x==0)
{
cout<<"This statement will not be executed\n";
throw x;
}
else
{
cout<<"Trying division suceed\n";
cout<<n/x;
}
}
catch(int a)
{
cout<<"Exception : Division by zero. Check the value";
}
}
};
void main()
{
MyException ex(20);
ex.Divide(0);
}

// multiple throw – multi catch statement
#include<iostream.h>
#include<conio.h>
void test(int x)
{
  try
  {
          if (x==1) throw x;
          if (x ==0) throw 'x';
         if (x==-1) throw 1.0;
  }

  catch(int x)
  {
          cout<<"Catch a integer and that integer is:"<<x;
  }
```

```
  catch(char x)
  {
        cout<<"Catch a character and that character is:"<<x;
  }
  catch(float x)
  {
        cout<<"Catch a float value and that value is:"<<x;
  }

}

void main()
{
  clrscr();
  cout<<"Testing multiple catches\n:";
  test(1);
  test(0);
  test(-1);
  getch();
}
```

Output:

```
Testing multiple catches
Catch a integer and that integer is: 1
Catch a character and that character is: x
Catch a float value and that value is: 1.0
```

// multiple throw - single catch

```
#include<iostream.h>
#include<conio.h>
void test(int x)
{
  try
  {
        if (x==1) throw x;
        if (x ==0) throw 'x';
       if (x==-1)  throw 1.0;
  }
   catch()
  {
        cout<<"\n Caught an exception";
  }
   }
 void main()
{
  clrscr();
  cout<<"\n Testing single catch for multiple throws :";
  test(1);
  test(0);
```

```
  test(-1);
  getch();
}
```

Output:

Testing single catch for multiple throws :
Caught an exception
Caught an exception
Caught an exception

Re throwing an exception

```
#include <iostream.h>
//#include <exception.h>
void division(int a, int b)
{
try
{
 if( b == 0 )  throw 0;
  else
    cout << " quotient =  "<< a/b;
}
catch(int )
  {
          cout<<"\n Division by zero condition! – function ";
throw;
  }
}
int main ()
{
try
{
division(50, 2);
division(50, 0);
}
//catch() – won't work
catch(int )
  {
          cout<<"\n Division by zero condition! – main ";
  }
return 0;
}
```

Output:

25
Division by zero condition! – function
Division by zero condition! – main

// VIRTUAL FUNCTIONS

```
#include<iostream.h>
```

```
#include<conio.h>
class Base
{
        int a;
        public:
        Base()
        {
                a = 1;
        }
        void display()
        {
                cout<<"\n Base class display()  is invoked";

        }
        virtual void show()
        {
                cout<<"\n Base class show() is invoked";
                cout << a<<"\n";
        }
};
class Derived: public Base
{
        int b;
        public:
        Derived()
        {
                b = 2;
        }
        void display()
        {
                cout<<"\n Derived class display() is invoked";

        }
        void show()
        {
                cout<<"\n Derived class show() is invoked";
                cout << b<<"\n";
        }
};
int main()
{
        clrscr();
          Base B;
          Derived D;
          Base *bp;
          B.show();
          B.display();
          D.show();
          D.display();
```

```
        bp=&B;
        bp->show();
        bp->display();
        bp = &D;
        bp->show();
        bp->display();
        getch();
        return 0;
}
```

// TRUE VIRTUAL FUNCTIONS - Abstract class

If a Base class has virtual function, then objects cannot be created for that base class. Such Base class
Are called as Abstract class

```
#include<iostream.h>
#include<conio.h>
class Base
{       protected:
          int a;
          public:
          Base()
          {
                    a = 1;
          }
          void display()
          {
                    cout<<"\n Base class display()  is invoked";

          }
          virtual void show() =0;
          };
class Derived: public Base
{
           int b;
           public:
           Derived()
           {
                    b = 2;
           }
           void display()
          {
                    cout<<"\n Derived class display() is invoked";

          }
           void show()
           {
                    cout<<"\n No base class show. only Derived class show() is invoked";
```

```
                cout << b<<"\n";
        }
};
int main()
{
        clrscr();

          Derived D;
          Base *bp;
          D.show();
          D.display();

          bp = &D;
          bp->show();
          bp->display();
          getch();
          return 0;
}
```

pointers to functions

```
// pointers to functions
#include<iostream.h>
#include<conio.h>

typedef void (*fnptr) (int,int);
void add(int x, int y)
{
cout<<"\n sum = "<<x+y;
}
void sub(int x, int y)
{
cout<<"\n difference = "<<x-y;
}

int main()
{
  fnptr p;
  clrscr();
  int a,b;
  cout<<"enter 2 integer values";
  cin>>a>>b;
  p=&add;
  p(a,b);
//   p(8,2);
  p=&sub;
  p(a,b);
 // p(9,4);
  getch();
  return 0;
}
```

Function templates

// 1. function templates with single data type

```
#include<iostream.h>
#include<conio.h>
template<class T>
T product(T value1,T value2)
{
T value;
value=value1*value2;
cout<<value;
return value;
}
void main()
{
int a,b,c;
float d,e,f;
clrscr();
cout<<"\n enter 2 int values\n";
cin>> a>>b;
cout<<"\n product of 2 int values\n";
c=product(a,b);
cout<<"\n product of int values are :"<<c;
cout<<"\n enter 2 float values\n";
cin>>d>>e;
cout<<"\n product 2 float values\n";
f=product(d,e);
cout<<"\n product of int values are :"<<f;
getch();
}
```

//2. function templates with 2 different data types

```
#include<iostream.h>
#include<conio.h>
template<class T1, class T2>
T2 product(T1 value1, T2 value2)
{
T2 value;
value=value1*value2;
cout<<value;
return value;
}
void main()
{
int a;
float d,e;
clrscr();
cout<<"\n enter a int value and a float value\n";
cin>> a>>d;
e=product(a,d);
cout<<"\n product of int and float values are :"<<e;
getch();   }
```

Templates for a class

//templates- class

```
#include<iostream.h>
#include<conio.h>
template<class T>
class sample
{
private:
T value,value1,value2;
public:
void getdata();
void product();
};
template<class T>
void sample<T>::getdata()
{
cin>>value1>>value2;
}
template<class T>
void sample<T>::product()
{
T value;
value=value1*value2;
cout<<value;
}
void main()
{
sample<int>obj1;
```

```
sample<float>obj2;
clrscr();
cout<<"\n enter 2 int values\n";
obj1.getdata();
cout<<"\n product of 2 int values\n";
obj1.product();
cout<<"\n enter 2 float values\n";
obj2.getdata();
cout<<"\n product 2 float values\n";
obj2.product();
getch();
}
```

PROGRAM USING STL - LISTS

Write a C++ program using lists from STL to input 10 numbers and store them in a list. From this list,create two more lists, one containing the even numbers, and the other containing the odd numbers. Output all the three lists.

```
int main()
{
list<int>li,oli,eli;
int i ,x,temp;
list<int>::iterator Iter;
for(i=0;i<10;i++)
{
cin>>temp;
li.push_back(temp);
}
cout<<" list of data: ";
for(Iter = li.begin(); Iter != li.end(); Iter++)
cout<<" "<<*Iter;
cout<<endl;
for(i=0;i<10;i++)
{
x=li.pop_back();
if(x%2==0)
eli.push_back(x);
else
oli.push_back(x);
}
cout<<"odd list data: ";
for(Iter = oli.begin(); Iter != oli.end(); Iter++)
cout<<" "<<*Iter;
cout<<endl;
cout<<"even list data: ";
for(Iter = eli.begin(); Iter != eli.end(); Iter++)
cout<<" "<<*Iter;
//for(i=0;i<5; i++)
//cout<<" "<<eli(i);
cout<<endl;
}
```

PROGRAM USING STL - VECTORS

Program:

```
#include<stdlib.h>
#include<vector.h>
#include<iostream.h>
#include<algorithm.h>
void main()
{
int n;
vector<int>vi;
for(int i=0;i<10;i++)
{
cout<<"Enter number["<<i<<"] ";
cin>>n;
vi.push_back(n);
}
vector<int>::iterator Index;
cout<<"Sort in asending order"<<endl;
sort(vi.begin(),vi.end());
for(Index=vi.begin();Index<vi.end();Index++)
{
cout<<*Index<<"\t";
}
cout<<"sort in descending order"<<endl;
sort(vi.rbegin(),vi.rend());
for(Index=vi.begin();Index<vi.end();Index++)
{
cout<<*Index<<"\t";
}
}
```

Output:

```
Enter number[0] 10
Enter number[1] 90
Enter number[2] 80
Enter number[3] 70
Enter number[4] 60
Enter number[5] 50
Enter number[6] 30
Enter number[7] 40
Enter number[8] 20
Enter number[9] 100

Sort in asending order
10      20      30      40      50      60      70      80      90      100

sort in descending order
```

100 90 80 70 60 50 40 30 20 10

//program to check balanced paranthesis using STL stack

```
#include<iostream>
#include<stack>
//#include<stdlib.h>
using namespace std;
int main()
{int i=0,c=0;
stack<char>s;
char expr[20];
cout<<"\nEnter the expression:";
cin>>expr;
while(expr[i]!='\0')
{
if(expr[i]=='(')
{
s.push('expr[i+1]');
c=1;
}
if(expr[i]==')')
{
if(s.top()==NULL)
c=c-1;
else
{
c=c-1;
s.pop();
}}
i++;
}
if(c==0)
cout<<"\nThe paranthesis is balanced";
else
cout<<"\nParanthesis is not balanced";
return 0;
}
```

OUTPUT:

```
elcot@elcot:~$ ./a.out
Enter the expression:(a-a)
The paranthesis is balanced  elcot@elcot:~$ ./a.out
Enter the expression:(a-s
Parathesis is not balancedelcot  @elcot:~$ ./a.out
Enter the expression:a
The paranthesis is balanced
```

FILE STREAM CLASSES

What is stream? Why they are useful? (NOV/DEC 2012)
A stream means the flow of data. There are two types of flow namely Input flow and output flow. Two common kinds of input flows are from input devices (keyboard, mouse) or Files means reading or receiving of data from a source. First case source is an input device, in second case source is a File. Similarly for output stream uses output devices and Files.

- **ofstream: Stream class to write on files**
- **ifstream: Stream class to read from files**
- **fstream: Stream class to both read and write from/to files.**

What is file mode? List any four file modes.
A file mode describes how a file is to be used, to read, to write to append etc. When you associate a stream with a file, either by initializing a file stream object with a file name or by using open() method, you can provide a second argument specifying the file mode. e.g.
List of filemodes available in C++
• ios::in
• ios::out
• ios::binary
• ios::ate

stream_object.open("filename",filemode);

Aim:

To write a c++ program to implement file stream classes.

Algorithm:

Step 1: Start.
Step 2: Declare object outfile for ofstream class.
Step 3: Using outfile write the data into the file.
Step 4: Close the file using close function.
Step 5: Delclare object infile for ifstream class.
Step 6: Use infinfile read the data from the file.
Step 7: Close the file.
Step 8: Stop.

Write a program to read a string to store it in a file and read the same string from the file to display it onthe output device (8) (NOV/DEC 2014)

Program:

```
// 1. Program to write the contents in a file
#include<fstream.h>
#include<iostream.h>
int main()
{
char data[100];
ofstream outfile;
//outfile.open("a.dat"); // write mode
outfile.open("a.dat",ios::app); //- append
//outfile.open("a.dat",ios::bin); //- binary mode
//outfile.open("a.dat",ios::ate); //- append
cout<<"writing into the file";
cout<<"enter details for first person \n";
cout<<"Enter your name";
cin.getline(data,100);
outfile<<data;
cout<<"Enter your age";
cin>>data;
cin.ignore();
outfile<<data;
cout<<"enter details for second person \n";
cout<<"Enter your name";
cin.getline(data,100);
outfile<<data;
cout<<"Enter your age";
cin>>data;
cin.ignore();
outfile<<data;
outfile.close();
return 0;
}
```

Output:

```
Writing to the file
enter details for first person
Enter your name: aaa
Enter your age: 10
enter details for first person
Enter your name: bbb
Enter your age: 20
```

// 2. Program to read the contents from a file

```
#include<fstream.h>
#include<iostream.h>
int main()
{
char data[100];
ifstream infile;
infile.open("a.dat"); //read mode
cout<<"Reading from the file";
while(!infile.eof())
{
infile>>data;
cout<<"\n"<<data;
}
infile.close();
return 0;
}
```

Output:

```
Reading from the file
aaa
10
bbb
20
```