Module 5 [Advanced Topics]

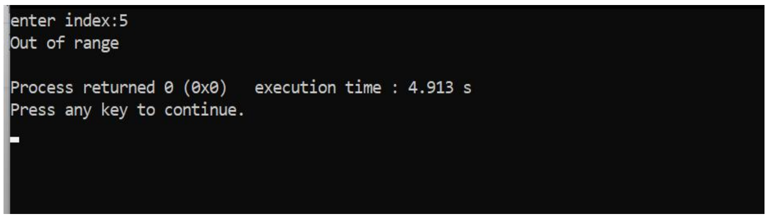
Topic 1[Exception Handling]
Problem Statement: In the following input i is the index of array ax[]. The program prints ax[i]. Then write catch block if i is out of range of ax[]. Write three catch blocks to fullfill the purpose
i) a catch block receives the value of i
ii) a catch block receives string “Out of Range Error”
iii) a default catch() if above two catch block doesn’t match

Theory:
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 .

Code i) :

```
#include <iostream>
using namespace std;
int main(){
int i;
int ax[5]={10,20,60,40,30};
cout<<"enter index:";
cin>>i;

try{

   if(i>4){
      throw i;
   }
   cout<<"ax["<<i<<"]="<<ax[i]<<endl;
}

catch(int t){
   cout<<"Out of range"<<endl;
   }

}
```

Output :



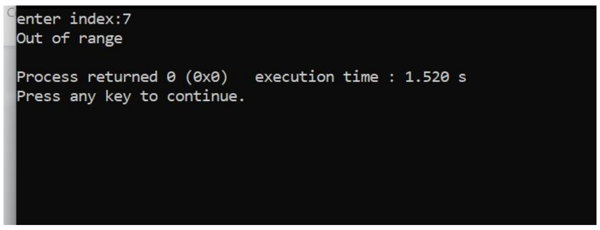
```
enter index:5
Out of range

Process returned 0 (0x0)   execution time : 4.913 s
Press any key to continue.
```

Discussion :
1. We can save our code from run time errors by using exceptional handling . 2. We need to throw a value or string by throw function to catch function.

Code ii) :

```
#include<bits/stdc++.h>
using namespace std;
int main(){
int i;
int ax[5]={10,20,60,40,30};
cout<<"enter index:";
cin>>i;

try{

   if(i>4){
      throw "Out of range";
   }
   cout<<"ax["<<i<<"]="<<ax[i]<<endl;
}

   catch(char const *ex){
   cout<<ex<<endl;
   }

}
```

Output :



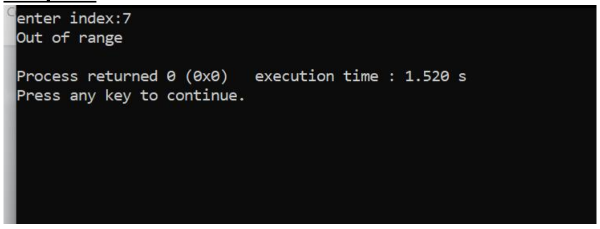
```
enter index:7
Out of range

Process returned 0 (0x0)   execution time : 1.520 s
Press any key to continue.
```

Code iii) :

```
#include <iostream>
using namespace std;
int main(){
int i;
int ax[5]={10,20,60,40,30};
cout<<"enter index:";
cin>>i;

try{

   if(i>4){
      throw "Out of range";
      throw i;
   }
   cout<<"ax["<<i<<"]="<<ax[i]<<endl;
}

   catch(char const *ex){
   cout<<ex<<endl;
   }
   catch(int t){
```

```
  cout<<"Out of range"<<endl;
  }
  catch(...){
  cout<<"Error"<<endl;
  }

}
```

Output :



```
enter index:7
Out of range

Process returned 0 (0x0)   execution time : 1.520 s
Press any key to continue.
```

Topic 2 [class Template]:

Problem Statement: In the following class data members x and y are integers and the method Sum() adds x and y. However, we need to perform the sum of int+int, int+double, douuble+int and double+double . To achieve it, change the definition of x,y,setData() and Sum() accordingly.

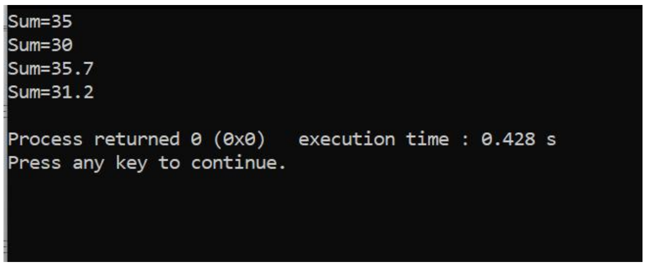
Theory :

The relationship between a class template and an individual class is like the relationship between a class and an individual object. An individual class defines how a group of objects can be constructed, while a class template defines how a group of classes can be generated .

Code :

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

template<typename T,typename R> class A{

T x;
R y;
public:

void setData(T x,R y){
this->x=x;
this->y=y;

}

T getSum(){

 return x+y;

}

};
int main(){

A <int,int>a;
a.setData(15,20);
cout<<"Sum="<<a.getSum()<<endl;

A <int,double>b;
b.setData(10,20.99);
cout<<"Sum="<<b.getSum()<<endl;

A <double,int>c;
c.setData(15.7,20);
cout<<"Sum="<<c.getSum()<<endl;

A <double,double>d;
d.setData(10.5,20.7);
cout<<"Sum="<<d.getSum()<<endl;

}
```

Output :



```
Sum=35
Sum=30
Sum=35.7
Sum=31.2

Process returned 0 (0x0)   execution time : 0.428 s
Press any key to continue.
```

Topic 3 [STL:Array class]

Problem statement: Declare a STL array object ax with 6 elements and do the following:

i) Assign 10,60,30,70,20 to ax using single statement
ii) Print third element of ax using at() function
iii) Print first element of ax using front() function
iv) Print last element of ax using back() function
v) Fill the elements of ax using fill() function
vi) Test whether ax is empty or not using empty() function
vii) Print size of ax
viii) Print maximum size of ax using max_size() function
ix) Print address of first element of ax using begin() function
x) Print address of last element of ax using end() function

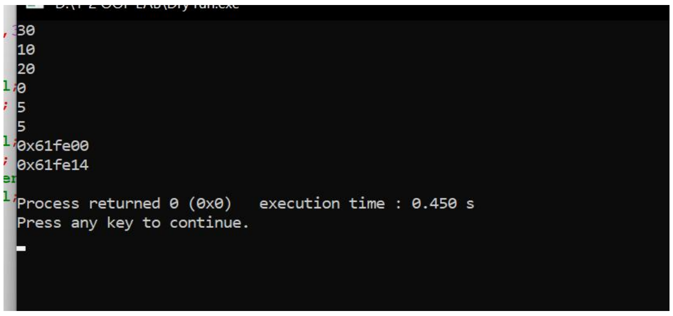
Theory :

C++ STL (standard template library) is a software library for the C++ language that provides a collection of templates representing containers, iterators, algorithms, and function objects.

Code :

```
#include <bits/stdc++.h>
using namespace std;

int main(){
array<int,5>ax={10,60,30,70,20};

cout<<ax.at(2)<<endl;
cout<<ax.front()<<endl;
cout<<ax.back()<<endl;
ax.fill(1);
cout<<ax.empty()<<endl;
cout<<ax.size()<<endl;
cout<<ax.max_size()<<endl;
cout<<ax.begin()<<endl;
cout<<ax.end()<<endl;

}
```

<u>Output :</u>



```
30
10
20
0
5
5
0x61fe00
0x61fe14

Process returned 0 (0x0)   execution time : 0.450 s
Press any key to continue.
```

<u>Discussion :</u>
1. STL is highly used in c++ . 2. It saves our time .We can do many works by stl functions without writing a large code.

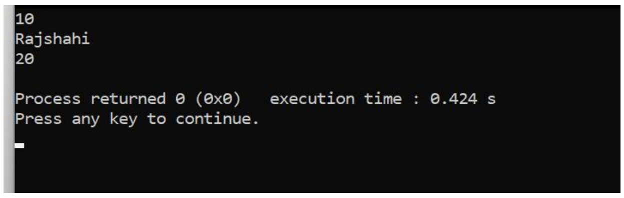
<u>Topic 4[STL: pair class]</u>
<u>Problem statement</u>: Define a pair class object px with int and string elements. Write statements to do the following
i) Assign 10 to int and "Rajshahi" to px using make_pair() function
ii) Print int data member by first
iii) Print string data member by second
iv) Modify first data member to 20 using get<>() function
v) Declare another pair bx and assign values to bx and swap it with ax

<u>Theory :</u> A pair in C++ is described as a container that combines two elements of
the same or difference data types .Pair in C++ consists of two elements, first and second (must be in this
order), and they are accessed using the dot (.) operator and the keyword first or second

<u>Code :</u>

```
#include <bits/stdc++.h>
using namespace std;
int main(){

pair<int,string>px={10,"Rajshahi"};
cout<<px.first<<endl;
cout<<px.second<<endl;

get<0>(px)=20;

cout<<px.first<<endl;
pair<int,string>bx,ax;
bx=px;
swap(ax,bx);

}
```

<u>Output :</u>


```
10
Rajshahi
20

Process returned 0 (0x0)   execution time : 0.424 s
Press any key to continue.
```

Topic 5 [STL: tuple class]

Problem statement: Define a tuple class object tx with int.string and double elements. Write statements to do the following

i) Assign <100,”Kamal”,3.5> to tx using make_tuple() function
ii) Print int data member by get() function

iii) Print string data member by get() function
iv) Print double data member by get() function
v) Modify third data member to 3.7 using get<>() function
vi) Declare another tuple bx and assign values to bx and swap it with ax

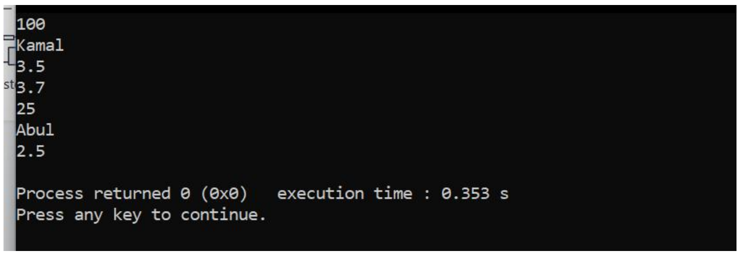
Theory :

A tuple is an object that can hold a number of elements .The elements can be of different data types . The elements of tuple is initialized as arguments in order which they will be accessed . Oprerations on tuple are get( ) and make tuple ( ) . get() : It is used to access the tuple values and modify them , it accepts the index and tuple names as arguments to access a particular tuple. make tuple() : It is used assign tuple with values .The values passed should be inorder with the values declared with tuple .

Code:

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
   tuple<int,string,double>tx;
   tx=make_tuple(100,"Kamal",3.5);
   cout<<get<0>(tx)<<endl;
   cout<<get<1>(tx)<<endl;
   cout<<get<2>(tx)<<endl;

   get<2>(tx)=3.7;
   cout<<get<2>(tx)<<endl;
   tuple<int,string,double>bx;
   bx=make_tuple(25,"Abul",2.5);
   bx.swap(tx);
   cout<<get<0>(tx)<<endl;
   cout<<get<1>(tx)<<endl;
   cout<<get<2>(tx)<<endl;
}
```

Output:



```
100
Kamal
3.5
3.7
25
Abul
2.5

Process returned 0 (0x0)   execution time : 0.353 s
Press any key to continue.
```

Topic 6 [STL: stack class]

Problem statement: Write a program to create and manipulate Stack using stack class and Perform the following operations using the specified method.

i) use push() method to push data
ii) use pop() method to pop data
iii) use top() method to display top element
iv) use empty() method to check whether stack is empty or not

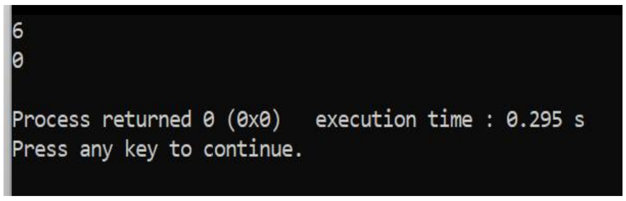
Theory :

Stacks are a type of container adaptors with LIFO(Last In First Out) type of working, where a new element is added at one end (top) and an element is removed from that end only. Stack uses an encapsulated object of either vector deque (by default) or list (sequential container class) as its underlying container, providing a specific set of member functions to access its elements.

Code :

```
#include <bits/stdc++.h>
using namespace std;
int main(){

 stack<int>st;
 st.push(5);
 st.push(6);
 st.push(7);
 st.pop();
 cout<<st.top()<<endl;
 cout<<st.empty()<<endl;

}
```

Output :



```
6
0

Process returned 0 (0x0)   execution time : 0.295 s
Press any key to continue.
```

Topic 7 [STL: queue class]

Problem statement: Write a program to create and manipulate Queue using queue class. Perform the following operations using the specified method.

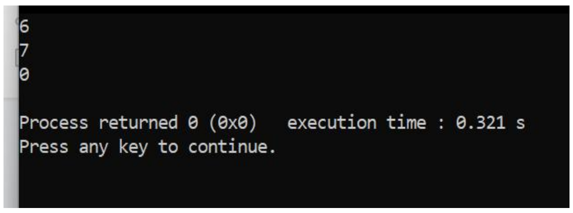
i) use push() method to push data
ii) use pop() method to pop data
iii) use front() method to display front element
iv) use back() method to display rear element
v) use empty() method to check whether queue is empty or not

Theory : Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both
its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed
first.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int main(){

 queue<int>st;
 st.push(5);
 st.push(6);
 st.push(7);
 st.pop();
 cout<<st.front()<<endl;
 cout<<st.back()<<endl;
 cout<<st.empty()<<endl;
}
```

Output :



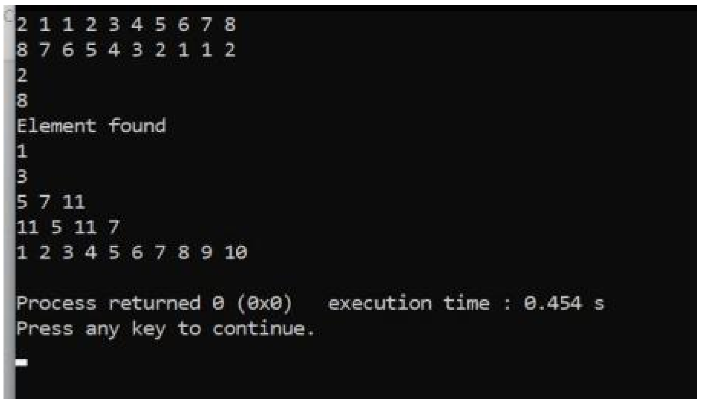
```
6
7
0

Process returned 0 (0x0)   execution time : 0.321 s
Press any key to continue.
```

Topic 8 [STL: list class]

Theory:
Lists are sequence containers that allow constant time insert and erase operations anywhere within the
sequence, and iteration in both directions. List containers are implemented as doubly-linked lists; Doubly linked lists can store each of the elements they contain indifferent and unrelated storage locations.The ordering is kept internally by the association to each element of a link to the element preceding it and a link to the element following it.They are verysimilar to forward_list: The main difference being that forward_list objects are single-linked lists, and thus they can only be iterated forwards, in exchange for being somewhat smaller and more efficient.
Code :

```
#include <bits/stdc++.h>
using namespace std;
void display(list<int>&li){
list<int>::iterator it;
for(it=li.begin();it !=li.end();it++){
cout<<*it<<" ";
}
cout<<endl;
}
void displayRev(list<int>&li){
list<int>::reverse_iterator it;
for(it=li.rbegin();it!=li.rend();it++){
cout<<*it<<" ";
}
cout<<endl;
}
bool even(const int& value) { return (value % 2) == 0; } //predicate function
int main(){
list<int>li,li2,li3;
int arr[10]={1,2,3,4,5,6,7,8,9,10};
for(int i=1;i<=8;i++){
li.push_back(i);
}
li.push_front(1);
li.push_front(2);
display(li);
displayRev(li);
cout<<li.front()<<endl;
cout<<li.back()<<endl;
li.pop_back();
li.pop_front();
auto it=find(li.begin(),li.end(),5);
if(it!=li.end()){
cout<<"Element found"<<endl;
}
else cout<<"Not found"<<endl;
int x=11;
li.insert(it, x); //inserting before 5
it++;
li.insert(it, x); //inserting after 5
cout<<count(li.begin(),li.end(),2)<<endl;
cout<<count_if(li.begin(),li.end(),even)<<endl;
auto itt=find(li.begin(),li.end(),6);
if(itt!=li.end()){
li.erase(it);
}
auto it1=li.begin();
auto it2=li.begin();
advance(it2,4); //deleting first 4 elements
li.erase(it1,it2);
li.remove(3);
li.remove_if(even);
li2.assign(li.begin(),li.end());
li3.assign(arr,arr+10);
li.sort();
li.unique();
display(li);
display(li2);
display(li3);
}
```

<u>Output:</u>



```
2 1 1 2 3 4 5 6 7 8
8 7 6 5 4 3 2 1 1 2
2
8
Element found
1
3
5 7 11
11 5 11 7
1 2 3 4 5 6 7 8 9 10

Process returned 0 (0x0)   execution time : 0.454 s
Press any key to continue.
```