CSE 1203

Object Oriented Programming [C++]

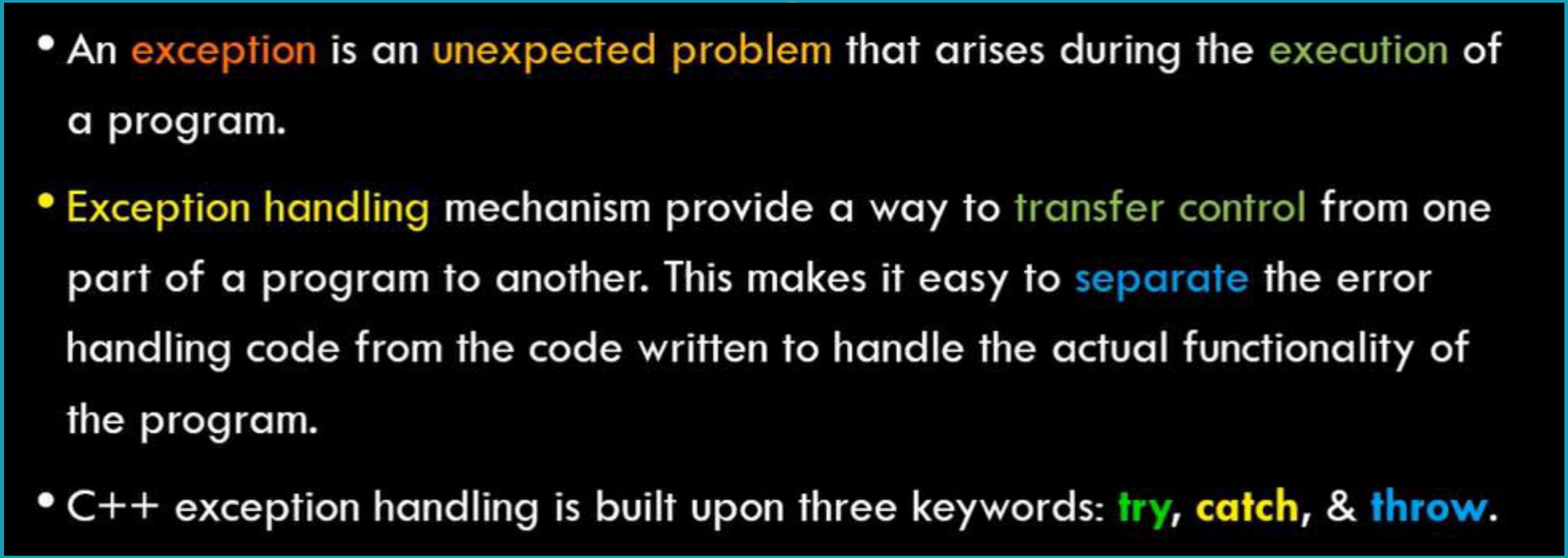
Chapter 4: Advanced Topics-1

Learning Objectives

To know about:

- Exception Handling
- Template Class & Function
- STL in C++

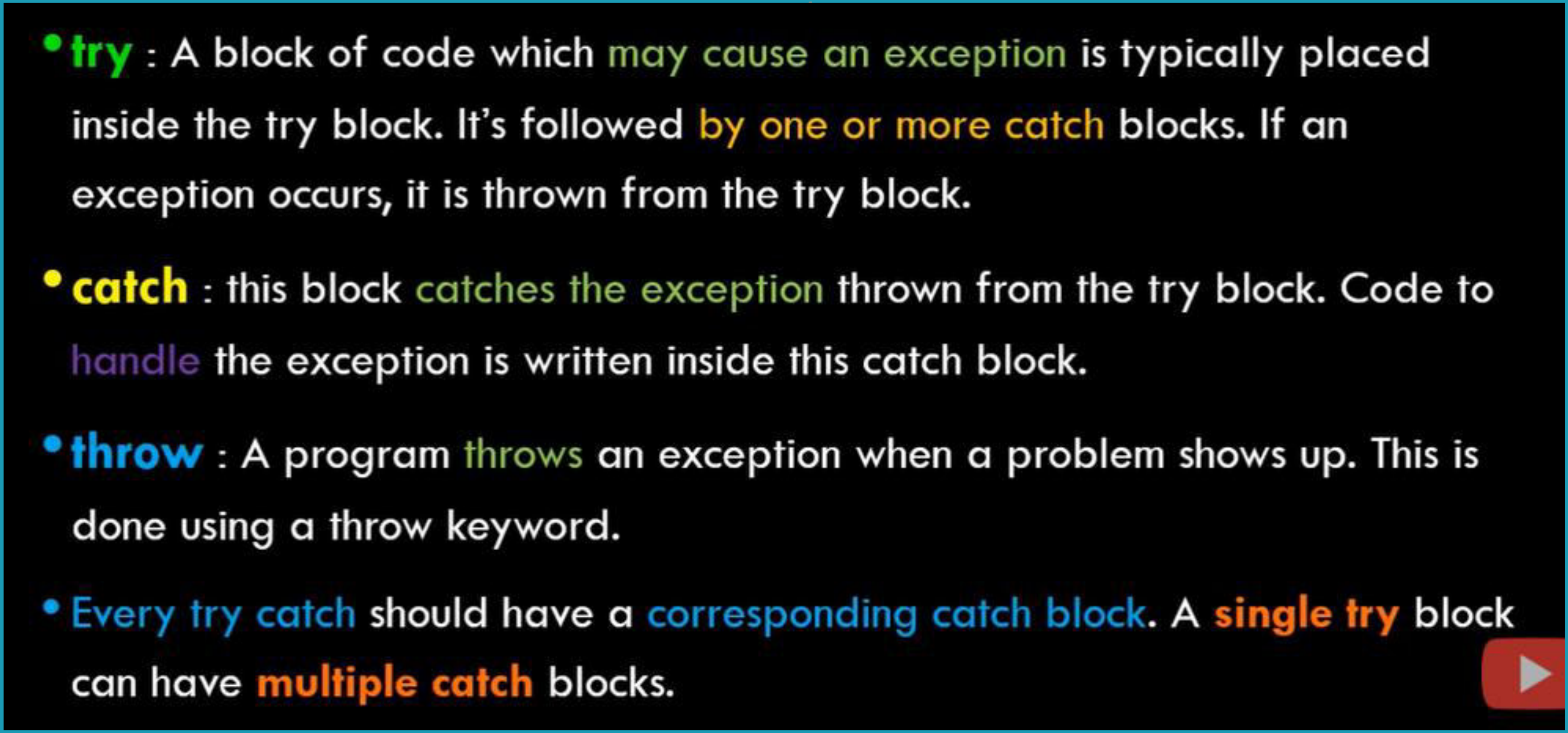
Exception Handling



- An exception is an unexpected problem that arises during the execution of a program.
- Exception handling mechanism provide a way to transfer control from one part of a program to another. This makes it easy to separate the error handling code from the code written to handle the actual functionality of the program.
- C++ exception handling is built upon three keywords: try, catch, & throw.

When it occurs
Suppose you are trying to access an index of an array which is not exist or divide by zero error at runtime or want to open a file which does not exists etc.

Exception Handling



- **try** : A block of code which may cause an exception is typically placed inside the try block. It's followed by one or more catch blocks. If an exception occurs, it is thrown from the try block.
- **catch** : this block catches the exception thrown from the try block. Code to handle the exception is written inside this catch block.
- **throw** : A program throws an exception when a problem shows up. This is done using a throw keyword.
- Every try catch should have a corresponding catch block. A **single try** block can have **multiple catch** blocks.

Exception Handling

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

int main()
{
  int n,d,r;
  cout<<"Enter n & d:";
  cin>>n>>d;
  r=n/d;
 cout<<"Division="<<r<<endl;
}
```

This program creates an error when d=0, actually it crashes.
It situation should be avoided

```
#include <iostream>
using namespace std;
int main()
{
  int n,d,r;
  cout<<"Enter n & d:";
  cin>>n>>d;
```

```
try{
    if(d==0){
        throw "This divide error";
    }
    r=n/d;

cout<<endl<<"Division="<<r<<endl;
  }
  catch(char const *ex){
   cout<<ex<<endl;
  }
```

Exception Handling

```
#include <iostream>
#include <stdexcept>
using namespace std;
int main()
{
  try{
        throw
runtime_error("Runtime Error");
  }
  catch(char const *ex){
   cout<<ex<<endl;
  }
  catch(int ex){
   cout<<"Integer
Error"<<ex<<endl;
  }
  catch(runtime_error e){
   cout<<e.what();
  }
}
```

Multiple catch block

The argument of throw matches the catch block and executes

Here write

`throw “String Error”` for 1st catch

`throw 20` for 2nd catch

`throw runtime_error(“Runtime error”)` for 3rd catch block

For all errors call catch(…)

```
catch(...){
   cout<<"For all errors"<<endl;
  }
```

Exception Handling

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

void Test()
throw(int,char,runtime_error){
 throw 20;
}
int main()
{
  try{
        Test();
  }
catch(int ex){
   cout<<"Integer Error"<<ex<<endl;
  }
catch(...){
   cout<<"For all errors"<<endl;
  }
}
```

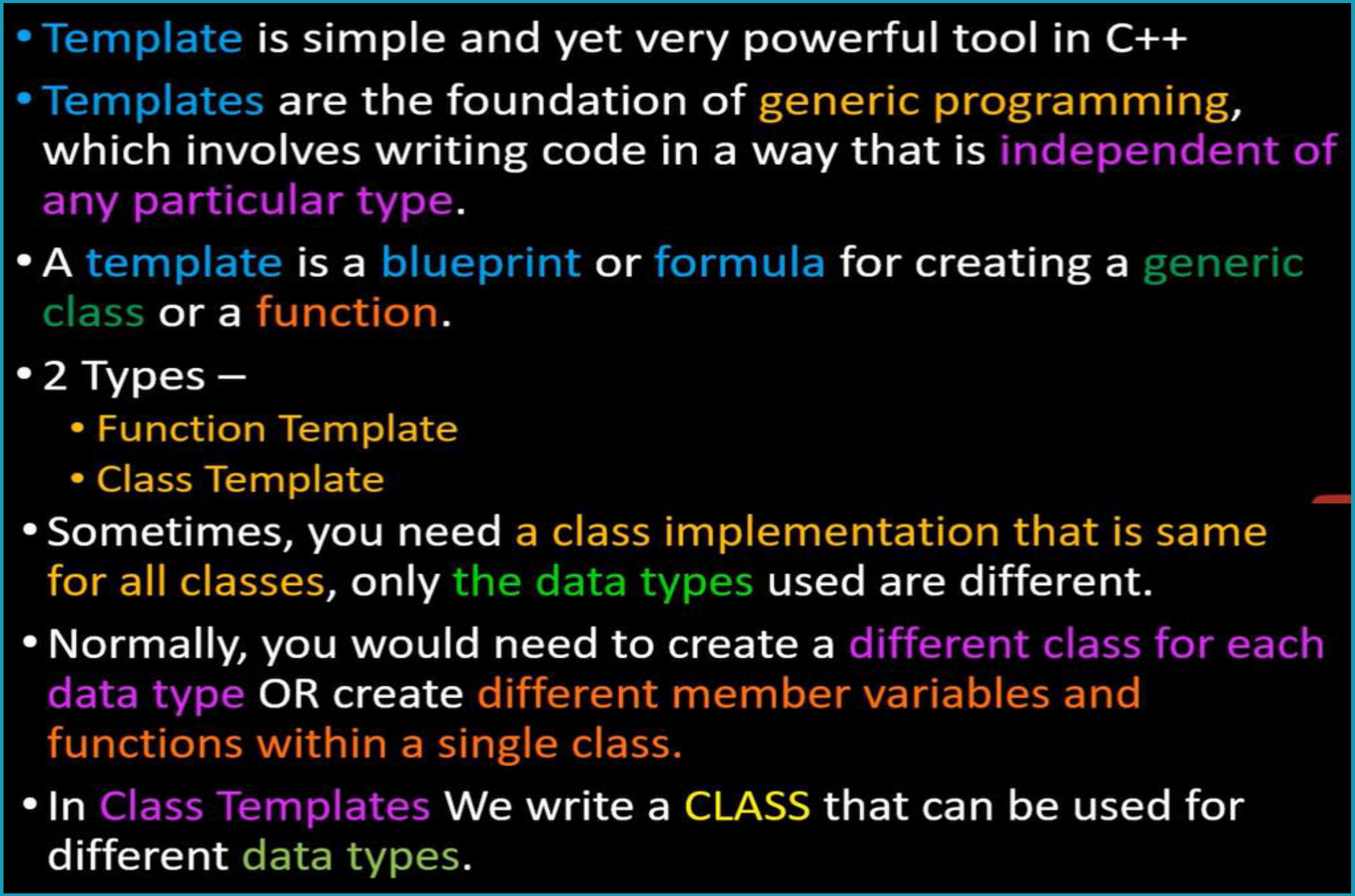
Exceptions in Function

The function Test() should be called from try block..

In the function definication different types of throw arguments are written.

According to the throw argument, appropriate catch block would be called.

Class Template



- Template is simple and yet very powerful tool in C++
- 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.
- 2 Types –
  - Function Template
  - Class Template
- Sometimes, you need a class implementation that is same for all classes, only the data types used are different.
- Normally, you would need to create a different class for each data type OR create different member variables and functions within a single class.
- In Class Templates We write a CLASS that can be used for different data types.

Class Template

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
template<typename T> class A{
         T x;
         T y;
         public:
                  void setData(T x,T y){
                           this->x=x;
                           this->y=y;
                  }
                  T getSum(){
                           return x+y;
                  }
};
int main()
{
  A <int>a;
  a.setData(15,20);
  cout<<"Sum="<<a.getSum();

  A <float>b;
  b.setData(10.5,20);
  cout<<"Sum="<<a.getSum();
}
```

Here in the main() object type can be changed according to the type of data.

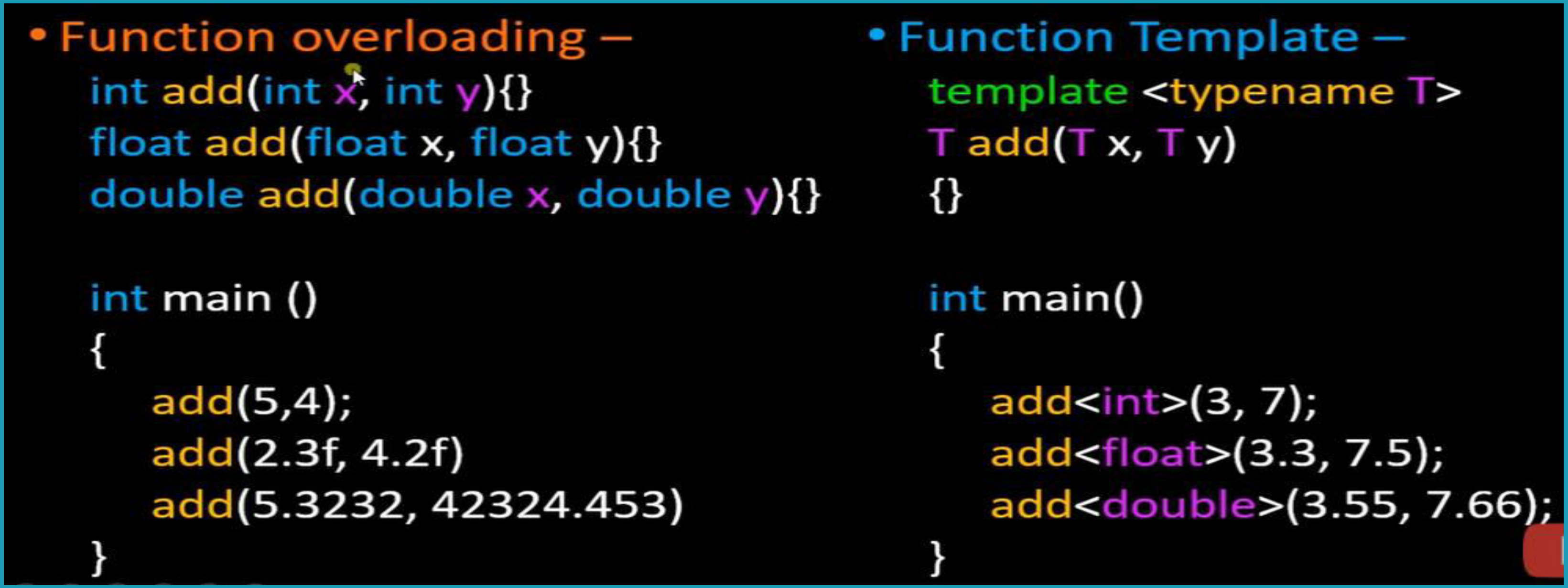
Class Template (Multiple Placeholder)

```
#include <iostream>
#include<stdexcept>
#include<bits/stdc++.h>
#include<array>
using namespace std;
template<typename K,typename U>
class A{
  K x;
  U y;
  public:
  void setData(K a,U b){
    x=a;
    y=b;
  }
  U Sum(){
    return(x+y);
  }
};

int main()
{
  A<int,double> ob1;
  ob1.setData(10,20.5);
  cout<<ob1.Sum();
}
```

Here multiple placeholder T & U are used.

Function Template



- Function overloading –

```
int add(int x, int y){}
float add(float x, float y){}
double add(double x, double y){}

int main ()
{
    add(5,4);
    add(2.3f, 4.2f)
    add(5.3232, 42324.453)
}
```

- Function Template –

```
template <typename T>
T add(T x, T y)
{}

int main()
{
    add<int>(3, 7);
    add<float>(3.3, 7.5);
    add<double>(3.55, 7.66);
}
```

```
template<typename T>
T Add(T x,T y){
 return x+y;
}
int main()
{
  cout<<"Sum="<<Add<int>(10,20)<<endl;
  cout<<"Sum="<<Add<float>(10.5,20.1)<<endl;
  cout<<"Sum="<<Add<double>(10.25,20.12);
}
```

STL in C++

STL: Standard Template Library

It consists of three components

i) **Container** : where data stored like array, link list, stack etc

ii) **Iterator:** move forward/backward in the container (different containers have different Iterator)

iii) **Algorithm:** different algorithms are available like searching, sorting etc

Extremly useful for competative programming

STL in C++ (Array)

The array is a collection of homogeneous objects and this array container is defined for constant size arrays or (static size).
In order to utilize arrays, we need to include the array header: `#include <array>`

```cpp
#include<iostream>
#include<array> // for array of STL
using namespace std;
int main() {

  // construction uses aggregate initialization
  // double-braces required
  array<int, 5> ar1{{3, 4, 5, 1, 2}};
  array<int, 5> ar2 = {1, 2, 3, 4, 5};
  array<string,3> ar3 = {"Raj", "Dha","Chi"};

  cout << "Sizes of arrays are" << endl;
  cout << ar1.size() << endl;
  cout << ar2.size() << endl;
  cout << ar3.size() << endl;

  cout << "\nInitial ar1 : ";
  for (auto i : ar1)
    cout << i << ' ';
}
```

https://cplusplus.com/reference/array/array/

STL in C++ (Array)

Member Functions of Array Template

Following are the important and most used member functions of array template.

i) `at` function

This method returns value in the array at the given range. If the given range is greater than the array size, out_of_range exception is thrown. Here is a code snippet explaining the use of this operator :

ii) `[ ] Operator`

The use of operator [ ] is same as it was for normal arrays. It returns the value at the given position in the array. Example : In the above code, statement cout << array1[5]; would print 6 on console as 6 has index 5 in array1.

iii) `front()` function

This method returns the first element in the array.

iv) `back()` function

This method returns the last element in the array. The point to note here is that if the array is not completely filled, back() will return the rightmost element in the array.

v) `fill()` function

This method assigns the given value to every element of the array,

STL in C++ (Array)

Member Functions of Array Template contd

vi) `swap()` function

This method swaps the content of two arrays of same type and same size.

vii) `empty()` function

This method can be used to check whether the array is empty or not.

Syntax: array_name.empty(), returns true if array is empty else return false.

viii) `size()` function

This method returns the number of element present in the array.

ix) `max_size()` function

This method returns the maximum size of the array.

For array, size() and max_size() will always give the same result.

x) `begin()` function

This method returns the iterator pointing to the first element of the array.

xi) `end()` function

This method returns an iterator pointing to an element next to the last element in the array

STL in C++ (Array Exmaple)

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

int main()
{
    array<int,5>ax;
    ax={10,20,30,40,50};
    cout<<"The array: ";
    for(auto x:ax)
        cout<<x<<" ";
    cout<<endl;
    cout<<"Element at index 3:"<<ax[3]<<endl;
    cout<<"Element at index 3:"<<ax.at(3)<<endl;
    cout<<"Front Element:"<<ax.front()<<endl;
    cout<<"Back Element:"<<ax.back()<<endl;
    cout<<"Size of ax:"<<ax.size()<<endl;
    cout<<"Size of ax:"<<ax.max_size()<<endl;
    cout<<"Address of 1st Element:"<<ax.begin()<<endl;
    cout<<"Address of last Element:"<<ax.end()<<endl;
    ax.fill(7);
    cout<<"The array: ";
    for(auto x:ax)
        cout<<x<<" ";
}
```

STL in C++ (pair class)

Pair is used to combine together two values that may be of different data types. Pair provides a way to store two heterogeneous objects as a single unit. It is basically used if we want to store tuples. The pair container is a simple container defined in **<utility>** header consisting of two data elements or objects.

- The first element is referenced as ‘first’ and the second element as ‘second’ and the order is fixed (first, second).
- Pair can be assigned, copied, and compared. The array of objects allocated in a map or hash_map is of type ‘pair’ by default in which all the ‘first’ elements are unique keys associated with their ‘second’ value objects.
- To access the elements, we use variable name followed by dot operator followed by the keyword first or second.

Syntax:

```
pair (data_type1, data_type2) Pair_name
```

1) make_pair(): This template function allows to create a value pair without writing the types explicitly.

Syntax: Pair_name = make_pair (value1,value2);

2) swap: This function swaps the contents of one pair object with the contents of another pair object. The pairs must be of the same type.

Syntax: pair1.swap(pair2) ;

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

int main()
{
  pair<int,string>a;
  pair<int,string>b;
  a=make_pair(100,"Zaman");
  b=make_pair(200,"Aslam");
  //a={200,"Belal"};
  //a.second="Kamal";
  a.swap(b);
  cout<<a.first<<" "<<a.second;

}
```

```
int main() {
 pair<int,pair<int,int>>p;// graph
 p.second.first=5;
}
```

STL in C++ (tuple class)

What is a tuple?

A tuple is an object that can hold a number of elements. The elements can be of different data types. The elements of tuples are initialized as arguments in order in which they will be accessed.

Operations on tuple :-

1. **get()** :- get() is used to access the tuple values and modify them, it accepts the index and tuple name as arguments to access a particular tuple element.
2. **make_tuple()** :- make_tuple() is used to assign tuple with values. The values passed should be in order with the values declared in tuple.
3. **swap()** :- The swap(), swaps the elements of the two different tuples.
3. **tie()** :- copy the values of tuple elements to the corresponding variables

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

int main()
{
  int p; string q;double r;
  tuple<int,string,double>a,b;
  a=make_tuple(100,"Zaman",500.5);
  b=make_tuple(200,"Belal",500.5);
  cout<<get<0>(a);
  //get<1>(a)="Kamal";
  cout<<get<1>(a);
  a.swap(b);
  cout<<get<1>(a);
  tie(p,q,r)=a;
  cout<<p<<q<<r;
}
```

THANK YOU