Generic Programming and Exception handling

Template

- Templates is a new concepts that enable us to define generic classes and function thus provides support for generic programming.
- Templates in C++ is an interesting feature that is used for generic programming and templates in c++ is defined as a blueprint or formula for creating a generic class or a function. Simply put, you can create a single function or single class to work with different data types using templates.
- A template can be used to create a family of classes or functions for examples, a class template for an array class enables us to create arrary or various data type such as int array or float array. We can define template for a function say mul(), that would help us to create various version of mul() for multiplying int ,float and double type values.
- A templates are also called a parameterized calles or function.
- There are two types of templates.

• Function template

• Class template

Function Template

Function template can be used to create a family of function with different argument type.

A single function template can work with different data types.

Any types of function argument is accepted by the function .

Defining a function template contain two steps.

1.Define general data type..

template<class template_name>

2.Defining a function with general data type.

return_type function_name(argument_with _template_name)

{
//body of function }

```
#include<iostream>
using namespace std;
template<class T>
void calculate(T a,T b)
{
T avg,pro;

avg=(a+b)/2;
pro=a*b;
 cout<<"Avarage="<<avg<<endl;
cout<<"Product="<<pro<<endl;
}
int main()
{
int i1=10,i2=20;
float f1=5.5, f2=10.5;
cout<<"Calculation for integral value:"<<endl; calculate(i1,i2);
cout<<"Calculate for floating value:<<endl;
calculate(f1,f2);
return 0;
}
```

Function template with multiple parameters

We can use more than one generic data types in function template. They are declared as comma separated list in following way.

Defining general data type

1.template <class template_name1,class template_name2.....>

2.return_type function_name(arguments_with_template_name1,templ ate_name2 and other)

{
//body of function with type template }

Example function template with multiple parameter and overloading.

```
#include<iostream>
using namespace std;
 template <class T1,class T2>
T1 Max(T1 a,T2 b)
{
if(a>b) return a;
 else return b;
}
template <class T1,class T2,class T3> T1
Max(T1 a,T2 b,T3 c)
{
if(a>b && a>c) return a;
else if(b>a && b>c) return b;
else
return c;
}
int main()
{
int a,b,c;
cout<<"Enter any three number:"<<endl;
cin>>a>>b>>c;
cout<<"Maximum number
among"<<a<<"and"<<b<<"is:"<<Max(a,b)<<
endl;
cout<<"Maximum number
among"<<a<<","<<b<<"and"<<c<<"is:"<<Ma
x(a,b,c);
return 0;
}
```

Class Template

Normally, we would need to create a different class for each data type or create different member variables and functions within a single class. This promotes the coding redundancy and will be hard to maintain, as a change in one class/function should be performed on all class/functions.

However, class template make it easy to reduce the same code for all data types.

The general format of class template is , template<class T>
class class_name
{

//class member with data type T

};

The prefix template <class T> tells the compiler that we are going to declare a template and use T as the type name in the declaration. T may be subtituted by any data type including the user defined types.

While creating object of class, it is necessary to mention which type of data is used in that object.

Syntax for creating object

The syntex dor creating an object if a template class is Class_name<data_type> object_name;

```
#include<iostream> using namespace std;
template <class T> class sample
{
private:
T a,b,sum;
public:
void getData(T x,T y)
{
a=x;
b=y;
sum=a+b;
}
void addData()
{
cout<<"The sum of two number
is:"<<sum<<endl; }
};
int main()
{
sample <int> s1;
sample <float> s2;
s1.setData(5,10);
s2.setData(5.5,10.5);
cout<<"Sum of integral value="<<endl;
s1.addData();
cout<<"Sum of float value="<<endl;
s2.addData();
return 0;
}
```

Standard template library

In order to help the C++ user in generic programming Alexander Stepanov and Menglee of Hwelett-Packard developed a set of general purpose templatized class(data structure) and function (algorithm) that could be used as a standard approach of storing and processing of data. The collection of these generic classes and function is called the standard template library(STL).

It help to save c++ user's time and effort there by helping to produce high quality program.

Component of STL

1.Container’s

2.Algorithm’s

3.Iterator's

Container

A container is an object that actually stores data.It is a way in which data is organized in memory. The STL containers are implemented by template classes and therefore can be easily customized to hold different types of data

As we know containers is an object that actually stores data. The STL defines ten containers which are grouped in three catagories as shown in fig

1.Sequence Container's

These containers stores elements in linear sequence. Element of these containers can be accessd using an iterator.
The STL provide three type of sequence container.

- Vector
- List
- Deque

2.Associative Container's

These container's are designed to support direct access to element using keys. They are not sequential. There are 4 types of associative container's.

- Set
- Multiset • Map
- Multipap

All these containers store data in structure called tree which facilitates fast searching deletion and insertion. However these are very slow for random access and insufficient for sorting.

3.Derived Container's

The STL provides three derived container namely stack, queue and priority queue. These are also called container adaptors.

Stack, queues and priority queus can be created from different sequence containers. The derived containers do not support iterator and therefor we cannot use them for data manipulation. However, they support two member function pop() and push() for implementing, deleting and inserting operations.

Algorithms

- An algorithm is a procedure used to process the data contained in the containers. The STL include many different kind of algorithms to provide support to task such as intitializing searching, coping , sorting and merging. Algorithms are implemented by template function.

Iteration

- An iterator is an object(like pointer) that points to an element in a container. We can use interator to move through the contents of containters. It is handled just like pointer and can be incremented and decremented. Iterator connect algorithm with containers and play a key role in the manipulation of data stored in the containers.

Exception handling

Exception are runtime anomalies or unusual condition that a program may encounter while executing. Anomalies might include condition such as division by zero,access to an array outside of its bound or running out of memory space or disk space.

The exception handling is a mechanism to delete and report an "exceptional circumstance" at runtime, so that appropriate action can be taken it provide a type safe, integrated approach for copying with the unusual predictable problem that arise while execution program.

Types of Exception

• Exception are basically of two type, namely synchronous and asynchronous exceptions.

• Errors such as "Out of range index" and "Over flow" belongs to synchronous type exceptions.

The error that are caused by the event beyond the control of program (such as keyboard interrupts) are called asynchronous exceptions.

The exception handling mechanism in C++ can handle only synchronous exceptions.

Try, Catch, Throw block

1.The keyword try is used to preface a block of statements(surrounded by braces) which may generates exception. This block of statement is known as try block.

2.When an exception is detected, it is thrown using a throw statement in the try blcok.

3. A catch block defined by the keyword catch, catches the exception thrown by the throw statement in the try blcok and handles it appropriately.

```
try {

//block of statement which detects and throw an exceptions

throw exception ; }

catch(type arg) // catch the exception
{
//block of statement that handle exception }
```

```
#include<iostream>
using namespace std;
int main()
{
int a,b,x;
cout<<"Enter value of a and b"<<endl;
cin>>a>>b;
x=a-b;
try
{
if(x!=0) {
cout<<"Result(a/x)="<<a/x<<endl;
}
else
{
throw(x);
}
}
catch(int i)
{
cout<<"Exception ZERO"<<endl;
} cout<<"END";
return 0;
}
```

Multiple Catch Statements

It is possible that a program segment has more than one condition to throw an exception. In such cases, we can associate more than one catch statement with a try(much like conditions in switch statement) as shown below.

```
try
{
//try block
}
catch(type1 arg) {
//catch block1
}
catch(type2 arg) {
//catch block2
}
catch(typeN arg)
{
catch blockN }
```

Example of multiple catch statement

```
#include<iostream>
using namespace std;
 void test(int x){
try {
if(x==1)
throw x;
else if(x==0)
throw 'x';
else if (x==-1)
throw 1.0;
cout<<"End of try block"<<endl;
}
catch(char c){
cout<<"Caught a character "<<endl;
}
catch(int m)
{
cout<<"Catch an integer"<<endl;
 }
catch(double d)
{
cout<<"Caught a double"<<endl;
}
// cout<<"End of try catch system"<<endl; }
int main()
{
cout<<"Testing multiple catches "<<endl;
cout<<"x==1"<<endl;
test(1);
cout<<"x==0"<<endl;
test(0); cout<<"x==-1"<<endl;
 test(-1);
cout<<"x==2"<<endl;
test(2);
return 0;
}
```

Handling Uncaught & Unexpected Exception

- Handling uncaught exceptions in C++ is important to prevent your program from crashing unexpectedly.
- There are a few different ways to handle uncaught exceptions, including:
- Using the set_terminate() function:
- Using the unexpected() function
- Using a global exception handler
- Use try/catch blocks to handle exceptions explicitly
- Log all uncaught exceptions.

- void my_terminate()

```
{
 // Perform any necessary cleanup tasks here.
std::abort();
 }
 int main()
 {
 set_terminate(my_terminate);
 // Code that may throw an exception goes here.
 return 0;
 }
```

Uncaught Exceptions

It is useful to see what happens when we don't handle them. This small program includes an expression that intentionally causes a divide-by-zero error:

```
class Exc0 {
public static void main(String args[]) {
int d = 0;
int a = 42 / d;
}
}
```

Thank You !