CER4C3
Abstraction and Paradigms of Programming

Bachelor of Engineering
Computer Engineering II nd Year IV th Semester

Generic Programming

Generic programming is a technique where generic types are used as parameters in algorithms so that they can work for a variety of data types.

Let’s say you want to create a function such that it can work on integers and floating point numbers as well.

Let’s say that function determines the greater number among two given numbers.

C++ Functions to determine the maximum of two numbers

```
Int max(int x , int y){
return (x>y)?x:y
}
Float  max(float x, float y){
return (x>y)?x:y
}
Int main(){
Cout<<max(2,3);
Cout<<max(2.23,4.41);
}
```

This is nothing but, method overloading basically..

Let’s say we want to create a single function that should be able to operate on both integer and floating point numbers.

Generic Function

We can do so with the help of generic functions. What is a generic function? A generic function defines a general set of operations applied to various types of data. The type of data, the function is going to operate upon is passed to it as a parameter.

Through generic functions, a single general procedure can be applied to range of data.

Advantage of using generic functions

Nature of algorithm is defined independent of the type of data.

Generic Functions

What we are saying is, when we are creating a generic function, we are creating a function that can automatically overload itself.

In order to create a generic function that is a function which can work with any type of data we need to use a keyword known as template.

template <typename MyType> void display(MyType x){

Cout<< “you have passed ”<<x<<endl;

}

Here either the typename or class keyword can be used to create a placeholder for the data type we are going to hold.

Template – keyword

Typename – keyword

Mytype – Any name for the identifier

Return type and name of the function

Generic Function

```
template <typename Type> void display(Type x){
Cout<< “you have passed ”<<x<<endl;
}
Int main(){
Display(10);
Display(20.4);
Display(“CS2 A”);
Return 0;
}
```

Generic Functions

Accepting Multiple Parameters and Returning Values :

```
#include<iostream>
using namespace std;
template <typename T> void display(T , T ) {
cout<< x << " and "<< y<<endl;
}

int main(){
display(2,3);
display(2.3,4.5);
return 0;
}
```

Generic Functions

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

cout<< max(20,30)<<endl;;
cout<< max(12.2,13.4)<<endl;;
}

template <class T> T max (T x , T y){
return (x>y)?x:y;
}
```

Generic Function

Passing both generic and standard parameters –

```
#include<iostream>
using namespace std;
template <typename T> void display (T x, int y){
        for (int i=1;i<=y;i++){
                cout<<x<<endl;
        }
        }
int main(){
        display(8,5);
        display(3.14,2);
        display("CS2 A",3);
}
```