Templates in C++

Templates is the type of polymorphism in C++ that allows us to use one function or class to handle many different datatypes.

Templates are the foundation of generic programming which involves writing code in a way that is independent of any particular datatype.

A template is a blueprint or formula for creating generic class or function.

Template concept can be used in two different ways:

(i) Function templates
(ii) Class templates.

(i) Function templates:

Function templates are special functions that can operate with generic types which serves as a pattern for creating other similar functions.

Here, we pass datatype as parameters to prevent redundancy.

(*) Syntax:

```
template < typename T > return_type function_name(parameters)
{
   // body
}
```

template: a keyword that signals compiler about function template.

Eg: While calling,

```
function_name <datatype> arguments;
```

Code is generated after the function is called.

Eg:

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

add<int>(3,4);
```

Also,

```
template <typename T, typename U>
T add (T x, U y)
{
   return (x+y);
}

add <double, int> (3.54, 3);
```

Date. No.

(ii) Class templates:

Class templates are special classes that can operate with generic datatypes which serves as pattern for creating other similar classes.
Here, we write a class that can be used by different datatypes.

(*) Syntax:

```
template <typename T>
class classname{
  access_specifies;
    T data-name;
    Class_name (T data_name): dataname(f) {}
};

class-name <datatype > obj-name (a);
```

While calling

```
typen template <typename T>
  T function_name ( Pair<T> p ){
    return p;
  }.
```

Date. No.

Eg:

```
#include <iostream>

template <typename T>
class Pair {
public:
  T first ; T second;
  Pair() {};
  Pair (T f, T s) : first(f), second(s) {}
};

template <typename T>   [if inside class, no need this ].
T max (Pair<T> p) {
    return (p.first < p.second ? p.second : p.first ); }

int main () {
  Pair<int > p1 (5, 6);
  std:: cout << p1.max << std:: endl;
  Pair<double> p2 (1.2, 3.4);
  Std:: cout << p2.max <<std:: cout ;
```