Passing Functions as Parameters

A function is a set of statements that take inputs, perform some specific computation, and produce output. The idea to use functions is to perform some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs.

The general form of a function is in the below format:

```
return_type function_name([ arg1_type arg1_name, ... ])
{
     // Perform Operations
}
```

Passing a function as an argument is a useful concept in C++. This concept has already been used while passing a custom comparator function as an argument in ***std::sort()*** to sort a sequence of objects as per the need. In this article, we will discuss different ways to design functions that accept another function as an argument.

A function can be passed as a parameter with 3 approaches i.e.

1. **Passing as Pointer**
2. **Using std::function<>**
3. **Using Lambdas**

1. Passing Pointer to a Function

A function can also be passed to another function by passing its address to that function; In simple terms, it could be achieved via pointers.

Example:

```
// C++ program to pass function as a
// pointer to any function
#include <iostream>
using namespace std;

// Function that add two numbers
int add(int x, int y) { return x + y; }
```

```
// Function that multiplies two
// numbers
int multiply(int x, int y) { return x * y; }

// Function that takes a pointer
// to a function
int invoke(int x, int y, int (*func)(int, int))
{
    return func(x, y);
}

// Driver Code
int main()
{
    // Pass pointers to add & multiply
    // function as required
    cout << "Addition of 20 and 10 is ";
    cout << invoke(20, 10, &add) << '\n';

    cout << "Multiplication of 20"
        << " and 10 is ";
    cout << invoke(20, 10, &multiply) << '\n';

    return 0;
}
```

Output

```
Addition of 20 and 10 is 30
Multiplication of 20 and 10 is 200
```

2. Using std::function<>

In C++ 11, there is a ***std::function***<> template class that allows to pass functions as objects. An object of std::function<> can be created as follows.

```
std::function<return_type(arg1_type, arg2-type...)> obj_nam
e
```

```
   //  This object can be use to call the function as below
   return_type catch_variable = obj_name(arg1, arg2);
```

Example:

```
// C++ program to demonstrate the passing
// of functions as an object parameter
#include <functional>
#include <iostream>
using namespace std;

// Define add and multiply to
// return respective values
int add(int x, int y) { return x + y; }
int multiply(int x, int y) { return x * y; }

// Function that accepts an object of
// type std::function<> as a parameter
// as well
int invoke(int x, int y, function<int(int, int)> func)
{
    return func(x, y);
}

// Driver code
int main()
{
    // Pass the required function as
    // parameter using its name
    cout << "Addition of 20 and 10 is ";
    cout << invoke(20, 10, &add) << '\n';

    cout << "Multiplication of 20"
        << " and 10 is ";
    cout << invoke(20, 10, &multiply) << '\n';
```

```
    return 0;
}
```

Output

```
Addition of 20 and 10 is 30
Multiplication of 20 and 10 is 200
```

3. Using Lambdas

Lambdas in C++ provide a way to define inline, one-time, anonymous function objects. These lambdas can be defined in a place where it is required to pass a function as an argument.

Example:

```
// C++ program to pass the function as
// parameter as a lambda expression
#include <functional>
#include <iostream>
using namespace std;

// Function that takes a pointer
// to a function
int invoke(int x, int y,
        function<int(int, int)> func)
{
    return func(x, y);
}

// Driver Code
int main()
{

    // Define lambdas for addition and
    // multiplication operation where
    // we want to pass another function
    // as a parameter

    // Perform Addition
    cout << "Addition of 20 and 10 is ";
```

```
    int k = invoke(20, 10,
                   [](int x,
                       int y) -> int {
                       return x + y;
                   });

    cout << k << '\n';

    // Perform Multiplication
    cout << "Multiplication of 20"
        << " and 10 is ";
    int l = invoke(20, 10,
                   [](int x,
                       int y) -> int {
                       return x * y;
                   });

    cout << l << '\n';

    return 0;
}
```

Output

```
Addition of 20 and 10 is 30
Multiplication of 20 and 10 is 200
```