2022 Data Structure HW2

TaeYong Kim

September 24, 2022

1 INTRODUCTION

템플릿(template)이란, 함수(function)나 클래스(class)를 개별적으로 작성하지 않아도, 여러 자료형(data type)으로 사용할 수 있게 만들어 놓은 틀이다. 템플릿은 두 가지로 나뉘어지는데, 함수 템플릿(function template)과 클래스 템플릿(class template)으로 나뉘어진다.

1.1 템플릿(template)이란?

템플릿(function template)이란, 함수나 클래스를 만들 때 함수의 자료형을 모호하게 설정하여, 자료형과 상관없이 함수의 기능이 같게 만드는 것이다.

1.2 템플릿(template)의 예시

예를 들어, 2개의 인자를 받아서 2개를 더한 값을 반환하는 함수는 다음과 같다.

```
int sum(int a, int b) {
    return a + b;
}
double sum(double a, double b) {
    return a + b;
}
```

이렇게 되면, 같은 기능의 함수를 자료형만 다르게 여러번 작성해야하는 문제가 발생한다. 이를 해결하기 위해 함수 템플릿을 사용한다. 함수 템플릿을 적용한 더하기 함수는 다음과 같다.

```
template <typename T>
T sum(T a, T b) {
    return a + b;
}
```

typename을 T로 두고 함수 템플릿을 사용한 더하기 함수를 작성하였다.

이렇게 템플릿을 사용하면, 함수나 클래스의 자료형을 크게 고려하지 않고 프로그래밍을 할 수 있다.

2 HOMEWORK

2.1 HW2A

조건
함수 선언은 한 번만 가능하다.

출력결과
3
7.7
19.31

소스코드

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

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

int main() {
    int intX = 1, intY = 2;
    double dbX = 3.3, dbY = 4.4;
    float fX= 09.24, fY = 10.07;

    cout << add(intX, intY) << endl;
    cout << add(dbX, dbY) << endl;
    cout << add(fX, fY) << endl;

    return 0;
}
```

설명
add 함수는 자료형과 상관없이 x와 y를 받아 둘의 합을 구하는 함수이다. 자료형과 상관없이 입력을 받고 출력을 하기 위해 함수 템플릿을 사용하였다.

2.2 HW2B

조건
1. 클래스 생성은 한 번만 가능하다.
2. 함수는 선언과 정의를 분리한다. 3. 함수 정의는 클래스 외부에서 가능하다.

출력결과
(3, 5)

(3.3, 5.5)

소스코드

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

template <typename T>
class CPoint {
public:
  T x;
  T y;

  CPoint(T x, T y);
  void Move(T x, T y);
  void Print();
};

template <typename T>
CPoint<T>::CPoint(T x, T y) {
  this->x = x;
  this->y = y;
}

template <typename T>
void CPoint<T>::Move(T x, T y) {
  this->x = (this->x) + x;
  this->y = (this->y) + y;
}

template <typename T>
void CPoint<T>::Print() { cout << '(' << x << ", " << y << ')' << "\n"; }

int main() {
  CPoint<int> P1(1, 2);
  CPoint<double> P2(1.1, 2.2);

  P1.Move(2, 3);
  P2.Move(2.2, 3.3);
  P1.Print();
  P2.Print();

  return 0;
}
```

설명

CPoint 클래스는 자료형과 상관없이 x와 y를 입력받아 기존의 값과 각각 더하는

함수와 이를 출력하는 함수를 가지고 있다. 자료형에 상관없이 작동하기 위해서 클래스 템플릿을 각각 사용하였다. 또한 함수의 선언과 정의를 분리하여 클래스 외부에서 함수를 정의하였다.

2.3 HW2C

조건
1. 클래스 생성은 한 번만 가능하다.
2. 함수는 선언과 정의를 분리한다.
3. 함수 정의는 클래스 외부에서 가능하다

출력결과
(9, 24)
(10.07, 22.59)

소스코드

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

template <typename T> class CPoint {
public:
  T x;
  T y;
  const char *s;

  CPoint(T x, T y);
  void Move(T x, T y);
  template <typename CP>
  friend ostream &operator<<(ostream &os, const CPoint<CP> &cp);
};


template <typename T>
CPoint<T>::CPoint(T x, T y) {
  this->x = x;
  this->y = y;
}

template <typename T>
void CPoint<T>::Move(T x, T y) {
  this->x = (this->x) + x;
  this->y = (this->y) + y;
}
```

```
template <typename T>
ostream &operator<<(ostream &os, const CPoint<T> &cp) {
  os << "(" << cp.x << ", " << cp.y << ")" << "\n";
  return os;
}

int main() {
  CPoint<int> P1(1, 11);
  CPoint<double> P2(1.1, 2.2);

  P1.Move(8, 13);
  P2.Move(8.97, 20.39);
  cout << P1 << P2;

  return 0;
}
```

설명

HW2B와 마찬가지로 클래스 템플릿을 사용하였다. HW2C의 문제는 operator overloading이다. 이를 해결하기 위해 CPoint 클래스 내부에 friend로 ostream을 선언하고, 여기에도 템플릿을 적용시켰다. 그리고 클래스 외부에서 left shift 연산자를 조건에 맞게 출력하게 정의하였다.