

# 프로그래밍 기초 I

포인터 변수와 함수 오버로드

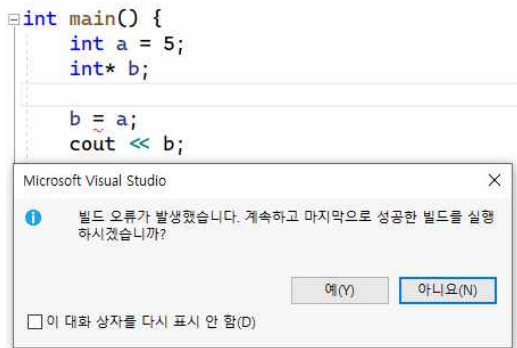
교수명 : 김근형  
제출자명 : 홍유성  
학번 : 20193463

## 1. 포인터 변수

포인터 변수는 데이터의 값을 갖는 다른 변수와 달리, 값이 아닌 해당 데이터가 저장된 주소를 저장하기 위한 저장공간이며, 기호로 ‘ \* ’을 사용한다. 값과 주소는 다른 개념이라는 것을 조심해야 한다. 포인터 변수는 주소를 저장하는 공간이므로 값, 또는 값을 가진 변수를 대입 복사할 수 없으며 주소만 저장할 수 있다.

주소란?

말 그대로 해당 데이터가 저장된 곳의 주소를 의미한다. 모든 데이터는 메모리에 저장공간이 할당될 시 고유한 주소 값을 함께 가지며 주소를 이용해 저장 장소의 구분과 수정 등이 가능하다. 주소는 해당 변수의 자료형이 가지는 저장공간을 바이트 단위로 나눠 구분한다.



주소를 담는 포인터 변수에 값을 저장해 오류가 생기는 모습이다.

포인터 변수 사용하기 위해서는 주소연산자인 ‘&’를 알아야 한다. ‘&’는 기호로는 참조변수이지만 연산자로 사용될 때에는 해당 변수가 저장된 주소를 가지게 된다.

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

0000003D7DCFF864

```
int main() {  
    int a = 5;  
    int* b;  
  
    b = &a;  
    cout << b;  
}
```

연산자 ‘&’를 사용해 값 형태인 a를 주소 형태로 치환했기에 포인터 변수에 저장하고 출력할 수 있다.

```
int main() {
    int a = 5;
```

이처럼 주소연산자를 사용해 해당 변수의 주소를 바로 출력할 수도 있다.

```
cout << &a;
```

Microsoft Visual Studio 디버그  
0000008D5E2FF894

```
int main() {
    int a[] = { 1,2,3,4,5 };
    int* b;

    b = a;

    cout << b << endl;

    b = &a[0];

    cout << b << endl;
```

Microsoft Visual Stud  
000000C8D9EFF8C8  
000000C8D9EFF8C8  
C:\Users\Foryoucom  
디버깅이 중지될 때  
하도록 설정합니다.  
이 창을 닫으려면 0

배열의 이름은 해당 배열의 처음 시작 주소를 뜻하므로 포인터 변수에 저장할 수 있다. 처음 시작 주소란 index 값이 0인 a[0]의 주소이며 index를 포함한 변수 a[0]은 정수 1을 가진 값의 형태이기 때문에 주소연산자를 이용해 포인터 변수에 저장 후 출력한 결과 배열의 이름과 배열의 처음 시작 공간은 같은 의미라는 것을 알 수 있다.

```
int main() {
    int a = 10;

    cout << &a << endl;
    {
        int a = 10;

        cout << &a << endl;
    }
```

주소는 해당 저장공간이 갖는 고유한 값이기에 저장공간의 이름이 같아도 변수의 유효 범위가 다르면 서로 다른 저장공간을 사용함을 알 수 있다.

이 주소를 통해 컴파일러는 이름이 같은 저장공간들을 서로 구별할 수 있다고 생각할 수 있다.

Microsoft Visual Studio 디버그 콘솔

0000005A450FF944  
0000005A450FF964

## 역참조변수 '\*'

연산자 '\*'는 해당 메모리 주소에 저장되어있는 값을 가지는 연산자이다.  
역참조변수를 포인터 변수가 갖는 주소에 저장되어있는 값을 가져오고, 수정할 수 있다.

```
int main() {  
    int a = 10;  
    int* b;  
  
    b = &a;  
    cout << *b;  
  
}
```

Microsoft Visual Studio 디버거 콘솔

포인터 변수 b에 변수 a의 주소를 저장하고 b가 가지는 주소에 저장되어있는 값을 '\*'을 이용해 나타냈다.

포인터 변수도 자료형을 가지고 있다. 특정 저장공간의 주소를 가져올 시 해당 저장공간의 자료형에 해당되는 바이트 수만큼 주소를 저장할 저장공간을 참조해야 하기 때문이다.

```
int main() {  
    double a = 10;  
    int* b;  
  
    b = &a;  
    cout << *b;  
  
}
```

Microsoft Visual Studio

빌드 오류가 발생했습니다. 계속하고 마지막도 하시겠습니까?

포인터 변수와 그 주소를 가져올 일반 변수와의 자료형이 일치하지 않아 에러가 발생한다.

```
int main() {  
    int a = 10;  
    int* b;  
  
    b = &a;  
    cout << *&a;  
  
}
```

Microsoft Visual Studio 디버거 콘솔

역참조변수와 주소연산자를 같이 사용할 수 있다.  
이 경우는 '&'를 통해 변수 a의 저장공간의 주소를 가졌다가 다시 '\*'를 통해 해당 주소에 저장된 값을 불러왔기 때문에 a에 저장된 10이 출력됨을 알 수 있다.

&(주소연산자) : 해당 변수가 메모리에 저장되어있는 주소를 얻음

\*(역참조 연산자) : 해당 주소에 저장되어있는 값을 얻음

이렇게 정리해볼 수 있다. 단 주소연산자는 값을 주소의 형태로 바꾸기 때문에 일반 변수에도 사용할 수 있지만, 역참조 연산자는 주소를 값의 형태로 바꾸기 때문에 주소연산자가 쓰인 변수나 포인터 변수에만 사용할 수 있다는 것을 주의해야 한다.

```
int main() {
    int a = 10;
    int* b;

    b = &a;
    cout << &b << endl;
    cout << b << endl;
}
```

Microsoft Visual Studio 디버그 콘솔

```
00000036F9AFF688
00000036F9AFF664
```

두 출력문이 다른 값이 나온 이유를 알아보자.  
첫 번째 출력문은 a의 주소를 가진 포인터 변수 b가 저장된 주소를 출력하고 있다.

두 번째 출력문은 포인터 변수 b가 가진 a의 주소를 출력하고 있다.

따라서 두 출력문은 서로 다른 저장공간의 주소를 출력하고 있기에 다른 값이 나올 수밖에 없다.

증감 연산자가 사용된 포인터 변수)

```
int main() {
    int a = 10;
    int* b;

    b = &a;
    cout << b << endl;
    b++;
    cout << b << endl;
}
```

Microsoft Visual Studio 디버그 콘솔

```
00000084F4FEF974
00000084F4FEF978
```

```
int main() {
    short a = 10;
    short* b;

    b = &a;
    cout << b << endl;
    b++;
    cout << b << endl;
}
```

Microsoft Visual Studio 디버그 콘솔

```
000000EE2F7CFA44
000000EE2F7CFA46
```

포인터 변수 b에 일반 변수 a의 주소를 가지게 한 후 b의 값을 1 증가시키는 연산을 수행하였다. 증가하기 전과 후를 비교해보면 해당 자료형이 가지는 바이트 단위로 주소값이 증가함을 알 수 있다. 따라서 포인터 변수는 다른 변수의 주소를 가져올 때 주소를 바이트 단위로 참조해 가져오기 때문에 자신과 같은 자료형의 변수만 주소를 가질 수 있다고 생각해볼 수 있다.

```

int main() {
    short a = 10;
    short* b;

    b = &a;
    cout << b << endl;
    ++b;
    cout << b << endl;
    *(++b) = 20;
    cout << b << endl;
}

```

예외가 발생함

b의 주소값이 1 증가한 주소에 값을 대입하고 실행하였지만 런타임 에러가 발생했다. 해당 주소는 선언된 저장공간이 없기 때문에 값을 저장할 공간이 없어 발생한다고 생각해 볼 수 있다.

배열과 증감 연산자가 사용된 포인터 변수)

```

int main() {
    int a[] = { 1,2,3,4,5 };
    int* b;

    b = a;

    cout << *b << endl;
    b++;
    cout << *b << endl;
}

```

Microsoft Visual Studio 디버그 콘솔

```

1
2

```

배열의 시작 주소를 갖는 포인터 변수 b를 1 증가시키고 해당 주소에 저장되어 있는 값을 출력시켜보았더니 배열의 두 번째 요소가 출력되었다. 이를 통해 배열은 각 요소를 자료형에 맞는 바이트 단위로 연속적으로 저장함을 알 수 있다.

문자열을 가진 배열과 포인터 변수)

```

int main() {
    char a[] = { "Hello World" };
    char* b;

    b = a;

    cout << b << endl;
    cout << a << endl;
    cout << &a[0] << endl;
}

```

Microsoft Visual Studio 디버그 콘솔

```

Hello World
Hello World
Hello World

```

해당 배열의 주소를 가진 포인터 변수와 해당 배열의 이름, 배열의 첫 번째 요소의 주소를 출력해보았는데 모두 같은 결과가 나왔다. 이는 세 개 모두 같은 의미를 가지고 있음을 뜻한다. 문자열이 저장된 배열은 시작 주소가 입력되면 모두 출력된다는 것과 배열의 이름은 시작 주소를 의미하는 것을 알기에 당연한 결과라 볼 수 있

다.

참조변수와 포인터 변수의 응용)

```
int main() {  
    int a;  
    int& ref = a;  
    int* poi;  
    int b = 10;  
    int arr[] = { 1,2,3,4,5 };  
  
    ref = b;  
    poi = arr;  
  
    cout << b << endl;  
    cout << ref << endl;  
    cout << a << endl;  
}
```

Microsoft Visual Studio 디버그 콘솔

```
10  
10  
10
```

```
int main() {  
    int a;  
    int& ref = a;  
    int* poi;  
    int b = 10;  
    int arr[] = { 1,2,3,4,5 };  
  
    ref = b;  
    poi = arr;  
  
    cout << &b << endl;  
    cout << &ref << endl;  
    cout << &a << endl;  
}
```

Microsoft Visual Studio 디버그 콘솔

```
0000008592CFF7B4  
0000008592CFF754  
0000008592CFF754
```

세 출력문은 모두 같은 값을 가지지만 b는 다른 주소를 가지고 있다. 이는 참조변수 ref는 a와 같은 저장공간을 사용하고 일반 변수 b는 이미 자기만의 저장공간을 따로 할당받았기 때문이다.

ref는 변수 a의 저장공간을 함께 사용하고 이 ref에 b의 값을 저장했기 때문에 b의 값이 a의 저장공간에도 저장된다. 따라서 세 변수는 값의 변화가 서로 공유되기 때문에 같은 값을 가지게 된다.

```
int main() {  
    int a;  
    int& ref = a;  
    int* poi;  
    int b = 10;  
  
    ref = b;  
    poi = &a;  
  
    cout << *poi << endl;  
}
```

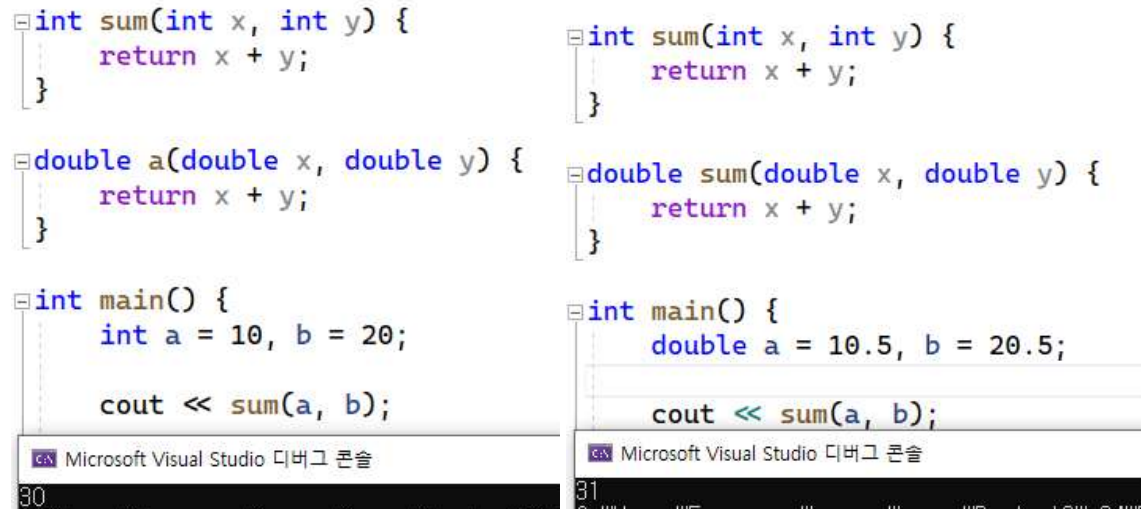
Microsoft Visual Studio 디버그 콘솔

```
10
```

포인터 변수 poi는 일반 변수 a의 주소를 가지고 있고 a는 ref과 같은 저장공간을 사용한다. 이 ref에 b의 값이 저장되어 a에는 b의 값인 10이 저장되고 a의 주소를 가진 poi에 역참조 연산자를 취해 해당 값을 출력해 10이 나왔다.



## 2. 함수 오버로드



```
int sum(int x, int y) {  
    return x + y;  
}  
  
double a(double x, double y) {  
    return x + y;  
}  
  
int main() {  
    int a = 10, b = 20;  
  
    cout << sum(a, b);  
}
```

```
int sum(int x, int y) {  
    return x + y;  
}  
  
double sum(double x, double y) {  
    return x + y;  
}  
  
int main() {  
    double a = 10.5, b = 20.5;  
  
    cout << sum(a, b);  
}
```

C++에서 추가된 기능으로 함수를 이름이 아닌 인자의 형태로 구분해 해당 함수로 이동한다. 인자가 실수면 실수를 매개변수로 사용하는 함수로 이동하고 인자가 정수면 정수를 매개변수로 사용하는 함수로 이동하는 방식이다. 이를 통해 매개변수의 형태만 다르고 기능은 같은 함수를 선언할 때 일일이 이름을 다르게 설정할 필요가 없어 프로그램 작성이 수월해진다. 이를 함수 오버로드, 또는 오버로딩 이라고 부른다.

이상으로 10주차 이론강의 정리를 마칩니다. 끝.