윈도우프로그래밍

3. C++ 기초(3)

2018. 3.23.

심미나 교수

목 차

I. C와 C++ 비교(4)

C와 C++ 비교

const

- [문제1] 키워드 const는 어떠한 의미를 갖는가?
  다음 문장들을 대상으로 이를 설명하시오.

```
const int num = 10;

const int * ptr1 = &val1;

int * const ptr2 = &val2;

const int * const ptr3 = &val3;
```

C와 C++ 비교

const

- [문제1] 키워드 const는 어떠한 의미를 갖는가?
  다음 문장들을 대상으로 이를 설명하시오.

```
const int num = 10;
 → 변수 num을 상수화!

const int * ptr1 = &val1;
 → ptr1이 가리키는 대상 즉, val1의 값을 상수화!

int * const ptr2 = &val2;
 → 포인터 ptr2를 상수화! 즉, val2를 가리킬 목적으로만 선언!

const int * const ptr3 = &val3;
 → 포인터 ptr3를 상수화! & ptr3가 가리키는 대상 즉, val3 값을 상수화!
```

C와 C++ 비교

메모리공간

- [문제2] 실행중인 프로그램의 메모리 공간
  - 실행중인 프로그램은 운영체제로부터 메모리 공간을 할당받는다.
  - 이는 크게 데이터, 스택, 힙 영역으로 나뉘는데, 각각의 영역에는 어떠한 형태의 변수가 할당되는지 설명하시오.

C와 C++ 비교

메모리공간

- **[문제2] 실행중인 프로그램의 메모리 공간**
  - **실행중인 프로그램은 운영체제로부터 메모리 공간을 할당받는다.**
  - **이는 크게 데이터, 스택, 힙 영역으로 나뉘는데, 각각의 영역에는 어떠한 형태의 변수가 할당되는지 설명하시오.**

- 데이터(global): 전역변수가 저장되는 영역
  프로그램이 종료될 때 비로소 해제되는 영역
- 스택(auto): 지역변수 및 매개변수가 저장되는 영역
  프로그램 실행과정(함수호출)에 필요한 메모리공간(자동 할당/해제)
- 힙(dynamic): malloc/free함수호출에 의해 프로그램이 실행되는 과정에서
  동적으로 할당되는 공간
  시점에 맞게 프로그래머가 공간 할당/해제를 지정해야 함

C와 C++ 비교

Call-by-value, Call-by-reference

- [문제3] Call-by-value, Call-by-reference
  - 함수의 호출형태는 크게 '값에 의한 호출'과 '참조에 의한 호출'로 나뉜다.
  - 이 둘을 나누는 기준이 무엇인지, swap함수(두 int형 변수의 값을 교환하는 함수)를 예로 들어 설명하시오.

```
Void SwapByValue(                    )
{


}

//Call-by-value
```

```
Void SwapByRef(                      )
{


}

//Call-by-reference
```

C와 C++ 비교

Call-by-value, Call-by-reference

- [문제3] Call-by-value, Call-by-reference
  - 함수의 호출형태는 크게 '값에 의한 호출'과 '참조에 의한 호출'로 나뉜다.
  - 이 둘을 나누는 기준이 무엇인지, swap함수(두 int형 변수의 값을 교환하는 함수)를 예로 들어 설명하시오.

```
Void SwapByValue(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}

//Call-by-value
```

→ 값을 전달
→ 이 경우 외부에서 값에 접근(변경)불가

```
Void SwapByRef(int *ptr1, int *ptr2)
{
    int temp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = temp;
}

//Call-by-reference
```

→ 주소값을 전달
→ 이 경우 포인터연산에 의해 밖의 값 접근(참조) 가능, 값 변경도 가능

II. C와 C++ 비교(5)

C와 C++ 비교

C++의 참조자(Reference)

- **참조연산자(&)를 사용하여 기존에 이미 선언된 변수에 붙이는 별칭**
  - 즉, 참조변수(별칭)를 선언하는 것으로 별도로 기억공간 할당 안함

(형식) 자료형 &'별칭으로 사용할 변수명' = '이미 선언된 변수명'

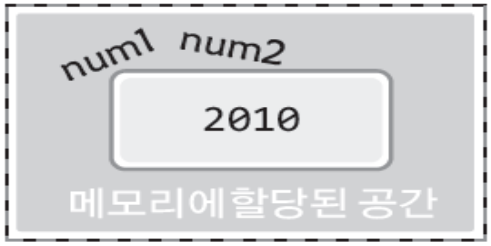
  - 참조연산자는 주소연산자로 사용되는 &와 구분됨
    - 참조연산자는 변수선언시 사용되며, 이미 선언이 끝난 변수에 사용된 &기호는 주소연산자

  - (예시) 변수의 선언 vs. 참조자(참조변수)의 선언

```
int num1 = 2010;
```


```
int num1 = 2010;
int num2 = num1;
```

```
int num1 = 2010;
int &num2 = num1;
```



C와 C++ 비교

C++의 참조자(Reference)

- (EX3-01) 참조자의 선언

EX3-01

```
int main(void)
{
    int num1=1020;
    int &num2=num1;
    num2=3047;
    cout<<"VAL: "<<num1<<endl;
    cout<<"REF: "<<num2<<endl;
    cout<<"VAL: "<<&num1<<endl;
    cout<<"REF: "<<&num2<<endl;
    return 0;
}
```

ⓘ num2는 num1의 참조자

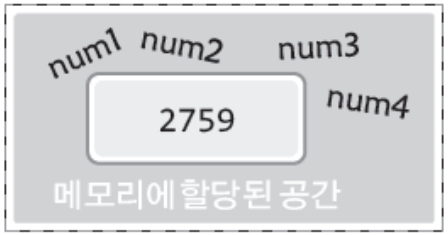
실행 결과

```
VAL: 3047
REF: 3047
VAL: 0012FF60
REF: 0012FF60
```

- 참조자의 수 제한없음
- 참조자를 대상으로 한 참조자 선언도 가능

```
int num1=2759;
int &num2=num1;
int &num3=num2;
int &num4=num3;
```

```
int num1=2759;
int &num2=num1;
int &num3=num1;
int &num4=num1;
```



C와 C++ 비교

C++의 참조자(Reference)

- **참조자 선언의 범위**
  - ① 참조자는 선언과 동시에 참조의 대상이 명시되어야 함

```
(잘못된 예) int &ref;
```

  - ② 참조의 대상은 기본적으로 변수이어야 함(상수 대상의 참조 불가)

```
(잘못된 예) int &ref=20;
```

  - ③ 참조자는 참조의 대상을 변경할 수 없음

```
(잘못된 예) int &ref=num1;
            int &ref=num2;
```

  - ④ 포인터처럼 NULL로 초기화할 수 없음

```
(잘못된 예) int &ref=NULL;
```

C와 C++ 비교

C++의 참조자(Reference)

- (EX3-02) 배열 요소에 대한 참조자의 선언

EX3-02

```
int main(void)
{
    int arr[3]={1, 3, 5};
    int &ref1=arr[0];
    int &ref2=arr[1];
    int &ref3=arr[2];

    cout<<ref1<<endl;
    cout<<ref2<<endl;
    cout<<ref3<<endl;
    return 0;
}
```

ⓘ 배열의 요소도 변수 성향을 가지므로 참조자 선언 가능

실행 결과

```
1
3
5
```

C와 C++ 비교

C++의 참조자(Reference)

- (EX3-03) 포인터 변수에 대한 참조자의 선언

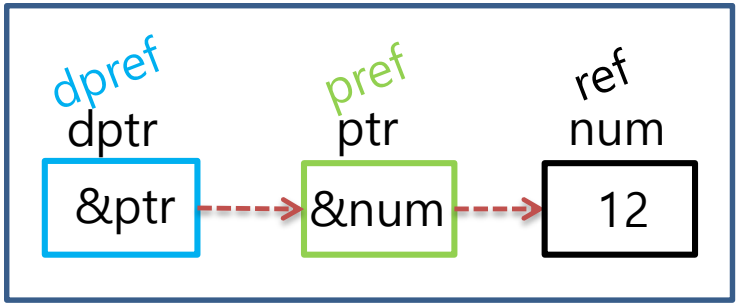
EX3-03

```
int main(void)
{
    int num=12;
    int *ptr=&num;
    int **dptr=&ptr;

    int &ref=num;
    int *(&pref)=ptr;
    int **(&dpref)=dptr;

    cout<<ref<<endl;
    cout<<*pref<<endl;
    cout<<**dpref<<endl;
    return 0;
}
```

ⓘ Ptr과 dptr 역시 주소값을 저장하는 포인터 변수이므로 참조자 선언 가능



실행 결과

```
12
12
12
```

C와 C++ 비교

C++의 참조자(Reference)와 함수

- Call-by-value & Call-by-reference
  - 값을 전달하면서 호출하는 함수 & 주소값을 전달하면서 호출하는 함수
  - 전자는 함수 외에 선언된 변수에 대한 접근이 불가능
  - 후자는 호출된 함수를 수행하면서 전달된 주소의 메모리 공간에 대한 접근이 가능

```
void SwapByValue(int num1, int num2)
{
    int temp=num1;
    num1=num2;
    num2=temp;
}   // Call-by-value
```

ⓘ 전달받은 값을 지역변수 num1에 복사하여 수행

```
void SwapByRef(int * ptr1, int * ptr2)
{
    int temp=*ptr1;
    *ptr1=*ptr2;
    *ptr2=temp;
}   // Call-by-reference
```

ⓘ 전달받은 값이 저장된 원래 메모리 공간에 대한 주소에 직접 접근

C와 C++ 비교

C++의 참조자(Reference)와 함수

- Call-by-value & Call-by-reference
  - (예시) Call-by-reference의 명확한 구분

```
int num1 = 10;
int num2 = 20;

SwapByRef(&num1, &num2)
{

}
```

ⓘ Call-by-reference

```
int num1 = 10;
int num2 = 20;

int *p1 = &num1;
int *p2 = &num2;

SwapByRef(p1, p2)
{

}
```

ⓘ P1, P2는 주소값을 갖는 변수로 주소값을 전달받음
ⓘ 그러나 p1, p2를 이용하여 "외부의 메모리 공간에 접근"하여 값을 변경하는 형태 아님

ⓘ Call-by-value

C와 C++ 비교

C++의 참조자(Reference)와 함수

- Call-by-value & Call-by-reference
  - (예시) Call-by-reference의 명확한 구분

```
int * SimpleFunc(int * ptr)
{
    return ptr+1;
}
```

ⓘ Ptr 자기 자신의 값(value)

ⓘ Call-by-value

```
int * SimpleFunc(int * ptr)
{
    if(ptr==NULL)
        return NULL;
    *ptr=20;
    return ptr;
}
```

ⓘ Ptr이 가리키는 값(reference)

ⓘ Call-by-reference

C와 C++ 비교

C++의 참조자(Reference)와 함수

- 참조자 이용한 Call-by-reference
  - "외부 메모리 공간에 접근"하기 위한 방법으로 주소값 전달하는 방법 외
  - 참조자를 매개변수로 전달받는 방식으로 Call-by-reference 구현
  - 매개변수에 선언된 참조자는 선언과 동시에 초기화

※ C++은 매개변수가 값인지 주소값인지 만으로는 call-by-value인지 call-by-reference인지 구분 안됨. 따라서 개발자들은 여전히 포인터 사용한 참조 형태 선호

```
int main(void)
{
    int val1 = 10;
    int val2 = 20;

    SwapByRef2(val1, val2);
    cout<<"val1:  "<<val1<<endl;
    cout<<"val2:  "<<val2<<endl;

    return 0;
}

void SwapByRef2(int &ref1, int &ref2)
{
    int temp = ref1;
    ref1 = ref2;
    ref2 = temp;
}
```

ⓘ int &ref = val1

C와 C++ 비교

C++의 참조자(Reference)와 함수

- const 참조자
  - 함수내에서 참조자 통해 값을 변경하지 않을 경우, const 선언의 장점

  1) 함수의 원형 선언만으로 값의 변경이 일어나지 않음을 명확히 판단 가능
     - C++에서는 함수의 정의형태와 호출형태를 보아도 값의 변경 유무를 알 수 없고, 함수의 몸체를 확인해야 하므로 불편함
     - 이 경우, 해당 함수내에서 참조자를 이용한 값의 변경은 허용하지 않겠다는 의미를 명확히 하기 위함

```
// 함수의 호출 형태
int num = 24;
HappyFunc(num);

// 함수의 정의 형태
void HappyFunc(int &ref)
{
    …  // 함수의 몸체
}
```

```
// const 참조자 명시

void HappyFunc(const int &ref)
{
    …
}
```

ⓘ HappyFunc 함수 내에서 참조자 ref를 이용한 값의 변경은 허용하지 않겠다는 의미
ⓘ 즉, 참조만 하고 변경은 안할 경우 명확히 표시

C와 C++ 비교

C++의 참조자(Reference)와 함수

- const 참조자
  - 2) 실수로 인한 값의 변경 방지 가능

(예시)

```
const int num=20;
int &ref=num;
ref+=10;
cout<<num<<endl;
```

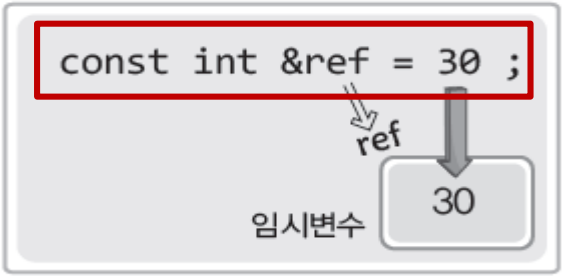
```
const int num=20;
const int &ref=num;
const int &ref=50;
```

ⓘ 한번 const 선언한 변수에 대한 참조자 선언은 모두 const로 선언

ⓘ Num을 const 선언했음에도 ref를 통한 값의 변경을 허용한 문제

C++의 참조자(Reference)와 함수

- const 참조자
  - 3) 상수에 대한 참조가 가능
    - 상수를 const 참조자로 참조할 경우, 이름없는 상수를 참조 가능
    - 이름없는 상수를 메모리 공간에 임시 저장 즉, 행을 바꿔도 소멸시키지 않음
    - 따라서 아래와 같은 형태의 함수 구현 가능



(예시)

```
const int a=30;
const int b=40;
Adder(a,b);
```

Adder(30,40); 호출 시
아래와 같이 편리하게 구현

```
int Adder(const int &num1, const int &num2)
{
    return num1+num2;
}
```

ⓘ 따라서 참조자를 매개변수로 하면서 상수 전달 가능!

C와 C++ 비교

C++의 참조자(Reference)와 함수

- **다양한 참조 형태**
  - (1) 반환형이 참조이고, 반환도 참조로 받는 경우

```
int& RefRetFuncOne(int &ref)
{
    ref++;
    return ref;
}

int main(void)
{
(1) int num1 = 1;
    int &num2 = RefRetFuncOne(num1);
    num1++;
    num2++;
    …
}
```

ⓘ ref는 지역적 참조자
따라서 (3)이후 소멸

(3) 함수 반환/반환값 저장
int &num2 = ref;

(2) 인자전달
int &ref = num1;

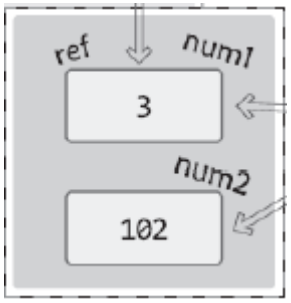
C와 C++ 비교

C++의 참조자(Reference)와 함수

- **다양한 참조 형태**
  - (2) 반환형은 참조이나, 반환은 변수로 받는 경우

ⓘ ref는 지역적 참조자
따라서 (3)이후 소멸

(3) 함수 반환/반환값 저장
int num2 = ref;




```
int& RefRetFuncOne(int &ref)
{
    ref++;
    return ref;
}

int main(void)
{
(1) int num1 = 1;
    int num2 = RefRetFuncOne(num1);
    num1+=1;
    num2+=100;
    cout<<"num1: "<<num1<<endl;
    cout<<"num2: "<<num2<<endl;
    return 0;
}
```

(2) 인자전달
int &ref = num1;

C와 C++ 비교

C++의 참조자(Reference)와 함수

- **다양한 참조 형태**
  - **(3) 참조를 대상으로 값을 반환하는 경우**

ⓘ 반환형이 참조형이 아니라면,
ⓘ 참조자를 반환하든, 변수에 저장된 값이 반환되든 값이 반환

```
int RefRetFuncTwo(int &ref)
{
    ref++;
    return ref;
}

int main(void)
{
(1) int num1 = 1;
    int num2 = RefRetFuncTwo(num1);
    num1+=1;
    num2+=100;
    cout<<"num1: "<<num1<<endl;
    cout<<"num2: "<<num2<<endl;
    return 0;
}
```

(2) 인자전달
`int &ref = num1;`

(3) 함수 반환/반환값 저장
`int num2 = ref;`
ⓘ 참조자의 값 반환

C++의 참조자(Reference)와 함수

- 다양한 참조 형태
  - (3) 참조를 대상으로 값을 반환하는 경우
    - 반환형이 참조형인 경우, 반환되는 대상은 참조자 그리고 변수로 받을 수 있음

```
int num2 = RefRetFuncOne(num1);    (O)
int &num2 = RefRetFuncOne(num1);   (O)
```

    - 반환형이 값의 형태인 경우, 반환되는 대상은 참조자로 받을 수 없음 (상수의 참조자 불가)

```
int num2 = RefRetFuncTwo(num1);    (O)
int &num2 = RefRetFuncTwo(num1);   (X)
```

C++의 참조자(Reference)와 함수

- **다양한 참조 형태**
  - (4) 잘못된 참조의 반환
    - 지역변수(변수의 값)로 참조되는 형태로 구현하지 말아야 함

```
int& RetuRefFunc(int n)
{
      int num=20;
      num+=n;
      return num;
}
```

```
int &ref=RetuRefFunc(10);
```

```
int &ref = num; 이 아니라,
int &ref = 30;로 반환되는 형태
```

III. C와 C++ 비교(6)

C와 C++ 비교

C++의 new & delete

- New연산자로 Heap 공간에 동적 할당
- 이렇게 할당된 메모리공간은 반드시 delete 함수를 호출하여 소멸
  - C와 달리, 할당할 공간의 크기를 바이트 단위로 계산할 필요 없음
  - 또한, 연산특성 때문에 객체생성과 소멸과정에서 malloc & free와 큰 차이가 있음

```
(형식) (자료형 포인터) 포인터변수명 = new (공간할당할 데이터 자료형)
       delete (포인터변수명)
```

- Cf. C의 경우
  - Malloc & free 함수 사용하며, 자료형이 아닌 할당할 크기 만큼의 byte를 계산하여 공간 할당함
  - 따라서, 자료형이 결정되지 않은 void형으로 포인터 반환

```
(형식) (포인터형) 포인터변수명 = (포인터형) malloc(sizeof(자료형))
       ex) int *ptr1 = (int *) malloc(sizeof(int))
```

C와 C++ 비교

C++의 new & delete

- new 예시
  - int * ptr1 = new int; (int형 변수 할당)
  - double * ptr2 = new double; (double형 변수 할당)
  - int * arr1 = new int[3]; (길이가 3인 int형 배열 할당)
  - double * arr2 = new double[7]; (길이가 7인 double형 배열 할당)

- delete 예시
  - delete ptr1; (int형 변수 소멸)
  - delete ptr2; (double형 변수 소멸)
  - delete []arr1; (int형 배열 소멸)
  - delete []arr2; (double형 배열 소멸)

※ 소멸대상이 배열인 경우 배열명 앞에 []표시

C와 C++ 비교

C++의 new & delete

- **참조자를 활용한 힙공간 접근**
  - 변수와 같은 (값의 변경이 가능한) 대상에 대해서는 참조자의 선언이 가능
  - 따라서, 힙 영역에 할당된 변수에 대해서도 참조자 선언이 가능
  - 즉, C에서는 힙 접근시 포인터만 사용했으나 C++에서는 참조자 활용한 접근 가능

```
int *ptr = new int;

int &ref = *ptr;

ref = 20;

cout<<*ptr<<endl;
```

C와 C++ 비교

C++의 표준헤더

- 표준C함수도 사용 가능
  - 표준C에 대응하는 표준C++함수는 C++ 문법 기반으로 변경 및 확장됨
  - 단, 가능한 C++의 표준함수를 사용하는 것이 바람직함

```
#include <stdio.h> → #include <cstdio>
#include <stdlib.h> → #include <cstdlib>
#include <math.h> → #include <cmath>
#include <string.h> → #include <cstring>
```

```
int abs(int num); (표준C의 abs함수)

→ long abs(long num);
   float abs(float num);
   double abs(double num);
   long double abs(long double num);
   (대응하는 C++ 표준 abs함수)
```

IV. 실습

실습3-01

교재 - 예제 5-9.(p.205)

참조 변수 선언하기

```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 int a=10;
06 int &b = a;
07 cout<<" a = "<<a<<" b = "<<b<<endl;
08 b+=300;
09 cout<<" b = "<<b<<endl;
10 cout<<" a = "<<a<<endl;
11 }
```

실습3-02

교재 - 예제 5-10.(p.206)

참조에 의한 전달 방식으로 두 변숫값을 교환하는 함수 작성하기

```
01 #include <iostream>
02 using namespace std;
03 void swap(int &x, int &y);
04 void main()
05 {
06 int a=10, b=20;
07 cout<<" a => "<< a <<" b => "<< b <<"₩n";
08 swap(a, b);
09 cout<<" a => "<< a <<" b => "<< b <<"₩n";
10 }
11 void swap(int &x, int &y)
12 {
13 int t;
14 t=x;
15 x=y;
16 y=t;
17 }
```

과제3

파일명 "과제3_분반_학번_이름"으로 제출

- **과제3**
  - 과제3-1 : 소스코드(ex301_학번.cpp), 실행결과화면(ex301_학번.jpg)
  - 과제3-2 : 보고서(HWP, MS Word) 파일 작성(A4 11포인트, 1장)
- **제출 시 주의사항**
  - 실행결과 마지막에는 "학과, 학년, 분반, 학번, 이름" 출력할 것
    - (형식: 컴퓨터공학부 2학년 1반, 2017000번, 홍길동입니다.)
  - 각 예제의 소스코드(*.cpp)와 실행결과화면(*.jpg 등), 보고서파일(hwp or ms word)을 한 개의 zip파일로 만들어 제출할 것

과제3

과제3-1

- **과제3-1**
  - 다음 문제에서 제시하는 프로그램을 작성하고, 실행결과를 제출하시오.

    - 다음은 변수 num에 대한 상수선언이다.

      const int num = 12;

    - 포인터 변수(ptr)를 선언해서 변수 num을 가리키게 하고, 이 포인터 변수를 참조하는 참조자(ref)를 선언하라.
    - 그리고 선언된 포인터 변수와 참조자를 각각 이용하여 num에 저장된 값을 출력하는 프로그램을 완성하라.

과제3

과제3-2

- **과제3-2**
  - C에서 배운 malloc & free 함수의 용도와 형식을 작성하시오.
  - 그리고 malloc & free를 활용한 코드를 포함하여 프로그램 예시를 하나 작성하시오.
  - (A4 파일로 작성)

감사합니다

mnshim@sungkyul.ac.kr