

게임공학과 20203324 최종현

3주차

변수 선언이란?

메모리에 저장공간을 할당하는 것이다.

프로그래머가 변수를 선언하면, 컴파일러는 이 정보를 바탕으로 해당 변수가 어떤 종류의 데이터를 저장할 것이며, 얼마나 많은 메모리 공간이 필요한지 결정한다.

이는 선언된 **데이터 타입**에 따라 달라진다.**int**는 **4바이트**
double는 **8바이트**.

할당된 메모리 공간은 **메모리 주소**를 가지게 된다.

메모리 주소 == 집 주소 라고 생각하면 편하다.

우리는 변수 이름을 통해 이 메모리 공간에 접근하지만, 실제로는 컴퓨터 내부적으로 메모리 주소를 사용하여 데이터를 읽고 쓴다.

즉 변수 선언은 우리가 메모리 주소를 전부 다 알 수 없으니 프로그래머가 알아보기 쉽게 이름표를 붙이는 것과 같다.

우리는 변수를 총 4가지를 배웠다.

일반 변수, 참조 변수, 포인터 변수, 배열 변수

일반 변수란 무엇일까?

우리가 흔히 쉽게 변수를 선언할 때 쓰는 것이 일반 변수 선언이다. `int age=10;` 10은 age라는 변수에 정수를 대입연산자를 통해 값을 저장한 것이다.

이 일반 변수의 메모리 주소에 우리가 접근하고싶을 때는 어떻게 해야할까?

바로 포인터 변수를 통해서 이 일반 변수의 메모리 주소를 알 수 있다. `*`(간접 참조 연산자)와 `&`(주소 반환 연산자)를 통해서 우리는 일반 변수의 메모리 주소를 알 수 있다.

여기서 포인터 변수는 데이터를 가질 수 없다. 오직 주소값만 저장할 수 있는 변수이다. 그렇기 때문에 포인터 변수에 값을 저장하려고 하면 오류가 생긴다.

```
int* ptr2 = 20;
```

이 포인터 변수는 어떻게 선언해야할까?

```
int a = 10;  
int* ptr = &a;  
int* ptr2;
```

이런식으로 사용을 하면 된다. `&`를 통해서 a의 변수 메모리 주소를 알 수 있다. 포인터 변수는 주소값을 저장하기 때문에 초기화를 하지 않아도 오류가 발생하지 않는다.

그렇다면 이제 참조 변수는 무엇일까?

참조 변수는 방금 봤던 &이 기호를 사용한다.

컴퓨터는 대입연산자 =을 기준으로 왼쪽에 * , & 등이 있다면 기호로 인식하고 오른쪽에 위치하면 연산자로 인식을 한다.

참조 변수는 이미 존재하는 변수에 대한 또 다른 이름이다.

마치 사람에게 여러 개의 별명이나 호칭이 있는 것과

비슷하다고 생각하면 된다. 참조 변수를 선언해도 새로운

메모리 공간이 할당되는 것이 아닌, 기존 변수가 가지고 있는 메모리 공간에 대해 새로운 이름을 부여하는 것이다.

참조 변수의 가장 큰 특징은 1. 반드시 초기화 되어야 한다.

어떤 변수를 참조할 것인지 명시해야 한다. 즉, 선언과 동시에 초기화를 해야 한다. 2. 참조 변수를 통해 원본 변수에

접근하고 수정할 수 있다. 참조 변수를 통해 값을 읽거나 수정하면, 원본 변수의 값도 함께 변경된다.

```
int& ref = *ptr;  
int& ref2 = a;  
int& ref3;
```

이처럼 일반 변수를 참조 할 수 있고, 포인터 변수 또한 참조 할 수 있다. 초기화가 이루어지지 않으면 저렇게 ; 사용해도 오류가 발생한다.

마지막 배열 변수는 무엇일까?

배열 변수는 동일한 데이터 타입의 여러 개의 데이터를 연속된 메모리 공간에 저장하기 위해 사용되는 자료 구조이다.

배열 변수의 특징으로는 배열의 모든 요소는 반드시 같은 데이터 타입이어야 한다. 배열의 요소들은 메모리 상에 순서대로 붙어서 저장된다.

배열은 선언할 때 크기가 결정되며, 프로그램 실행 중에는 크기를 변경할 수 없다.(동적 배열 예외)

컴퓨터는 원래는 덧셈밖에 하지 못한다. 그렇다면 뺄셈을 어떻게 하는것일까?

컴퓨터는 2의 보수라는 개념을 이용한다. 그렇다면 2의 보수는 무엇일까? 1. 먼저 1의 보수를 구한다. 주어진 이진수의 모든 비트를 반전시킨다. 0은 1로 1은 0으로 바꾼다.

2. 1의 보수에 1 더하면 2의 보수가 된다.

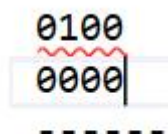
우리가 쉽게 계산하는 $4-2$ 는 $4+(-2)$ 와 같다. 컴퓨터는 이 원리를 이용하여 뺄셈을 덧셈으로 처리한다.

4를 이진수로 표현해보자 0100이다. 2의 이진수는 0010 이다.

이제 -2를 표현하기 위해 2의 2의 보수를 구해보자

비트를 전부 반전시키면 1101이 된다. 여기서 1을 더하면 0000이 된다. 이것이 -2를 표현하는 4비트 이진수이다.

0100과 0000을 더한다면


$$\begin{array}{r} 0100 \\ + 0000 \\ \hline \end{array}$$

10010

이렇게 나온다 가장 왼쪽의 자리 올림(1)은 무시한다. 그러면 결과적으로 2라는 답이 나온다.

실습

```

int main()
{
    int a = 10;
    int* ptr = &a;
    int* ptr2;
    int& ref = *ptr;    // 참조 변수를 통해 a의 값을 바꿀 수 있다.
    cout << ptr << endl;
    *ptr = 30; // 이때의 * 는 연산자이다.
    cout << *ptr << endl; // 30
    cout << a << endl; // 30
    ref = 40;
    cout << ref << endl; // 40
    cout << a << endl; // 40
    cout << endl;

    // 이중 포인터
    int** ptr1 = &ptr;
    cout << ptr1 << endl; // ptr1의 주소
    cout << *ptr1 << endl; // *ptr1이 가리키는 값(ptr의 주소)
    cout << **ptr1 << endl; // **ptr1이 가리키는 값(a의 값)
}

```

// 배열 변수

```

cout << endl;
int arr[3] = { 1,2,3 };

cout << arr[0] << endl;
cout << arr[1] << endl;
cout << arr[2] << endl;
arr[0] = 10;
cout << arr[0] << endl;
cout << sizeof(arr) << endl; // 4바이트 자료형인 int형을 3개를 가지고 있기 때문에 12가 나온다.
for (int i = 0; i < 3; i++) // for문을 통해 순회하며 arr배열의 주소값을 출력하게 한다.
{
    cout << &arr[i] << endl; // 결과적으로 4바이트만큼 띄워서 출력이 될것이다.
}

```

```
000000661E77F8B4
30
30
40
40

000000661E77F8D8
000000661E77F8B4
40

1
2
3
10
12
000000661E77F958
000000661E77F95C
000000661E77F960
```

```
int a1;
double d1;
char c1;

// sizeof()
// 특정 데이터 타입이 얼마나 많은 메모리를 사용하는지 알 수 있다.
// 배열 전체의 크기를 바이트 단위로 얻을 수 있다.
// 이를 통해 배열의 요소 개수를 계산할 수 있다.
// 구조체와 클래스가 메모리에서 차지하는 총 크기를 알 수 있다.
cout << "int 자료형의 크기: " << sizeof(int) << " 바이트" << std::endl;
cout << "double 자료형의 크기: " << sizeof(double) << " 바이트" << std::endl;
cout << "char 자료형의 크기: " << sizeof(char) << " 바이트" << std::endl;
cout << "int 변수 a1의 크기: " << sizeof(a1) << " 바이트" << std::endl;
cout << "double 변수 d1의 크기: " << sizeof(d1) << " 바이트" << std::endl;
cout << "char 변수 c1의 크기: " << sizeof(c1) << " 바이트" << std::endl;
cout << "배열 arr 크기: " << sizeof(arr) << " 바이트" << std::endl;
```

```

unsigned char c2;
cout << "unsigned char c2의 크기" << sizeof(c2) << endl;
// char 자료형은 일반적으로 1바이트(8비트)의 크기를 가지고 있다.
// unsigned 키워드가 붙어있기 때문에 부호가 없는 (non-negative) 정수형이다. 음수 값을 표현할 수 없다.
// 8비트로 표현할 수 있는 모든 비트를 값의 크기를 나타내는 데 사용하므로, 0부터 255까지의 정수 값을 표현한다.
//
// 사용하는 이유
// 일반적으로 -128부터 127까지의 값을 표현하거나
// 0부터 255까지의 값을 표현할 수 있다.
// 어떤 범위를 가지는지는 컴파일러 및 플랫폼에 따라 다르다.
// 그렇기 때문에 확실하게 양의 정수만 표현하기 위해서 unsigned 를 사용한다.
// 예시) 이미지 처리 데이터의 픽셀 값은 0~255 사이의 값을 가지므로 unsigned를 사용해야 한다.

```

```

int 자료형의 크기 : 4 바이트
double 자료형의 크기 : 8 바이트
char 자료형의 크기 : 1 바이트
int 변수 a1의 크기 : 4 바이트
double 변수 d1의 크기 : 8 바이트
char 변수 c1의 크기 : 1 바이트
배열 arr 크기 : 12 바이트
unsigned char c2의 크기 1

```