땅울림
객체 스터디

Made by 규정

5주차

1

배열 포인터

2

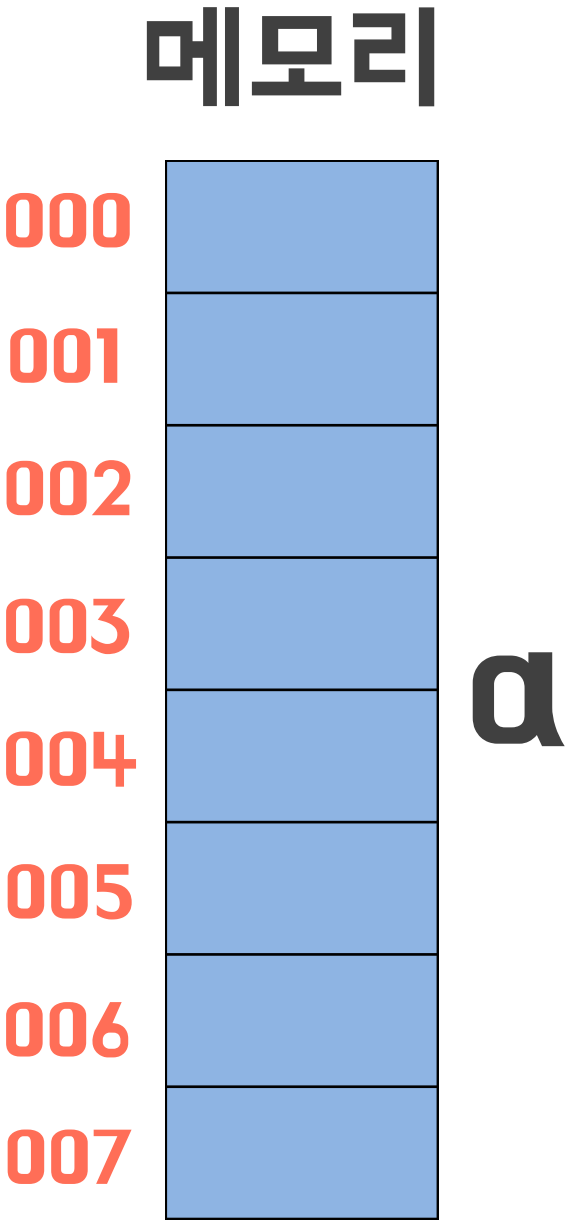
동적 할당

3

구조체

1 포인터

```
int a = 30;
```



1 포인터

포인터?

변수는 변수인데, **주소**를 저장하는 변수!

변수를 선언하면 메모리를 할당받는데,
그 메모리에는 **주소**가 매겨져 있다.

주소를 찾아가면 저장된 변수에 접근할 수 있다.
포인터는 변수의 **주소**를 저장하는 역할

1 포인터

일반 변수는 값을 저장하고,
포인터형 변수는 주소를 저장한다.

1 포인터

*과 &는 한 세트

```
int a = 30;
```

int* ptr과 int *ptr은 똑같음

변수의 주소를 저장하고 싶으면? => int* ptr = &a;

(int *)은 포인터형 변수이므로 **주소만 저장**할 수 있음!

~~int *ptr = a;~~

~~int ptr = &a;~~

1 포인터

출력할 때

앞에 *이 붙어있으면 => 값 출력

앞에 &이 붙어있으면 => 주소 출력

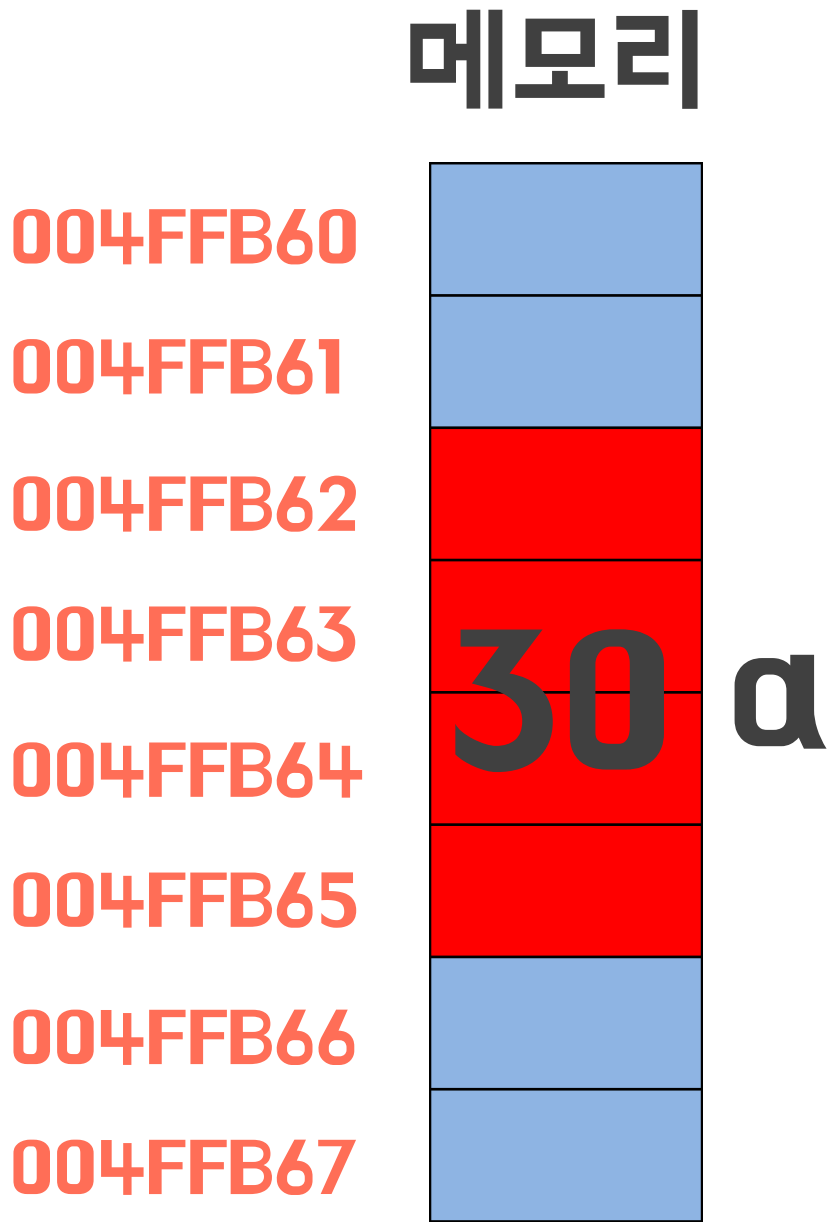
1 포인터

&: 변수의 주소를 알고 싶을 때!

```
int a = 30;

cout << a;   → 30

cout << &a;  → 004FFB62
```



1 포인터

*: 주소의 내용을 알고 싶을 때!

int a = 30; → a == 30

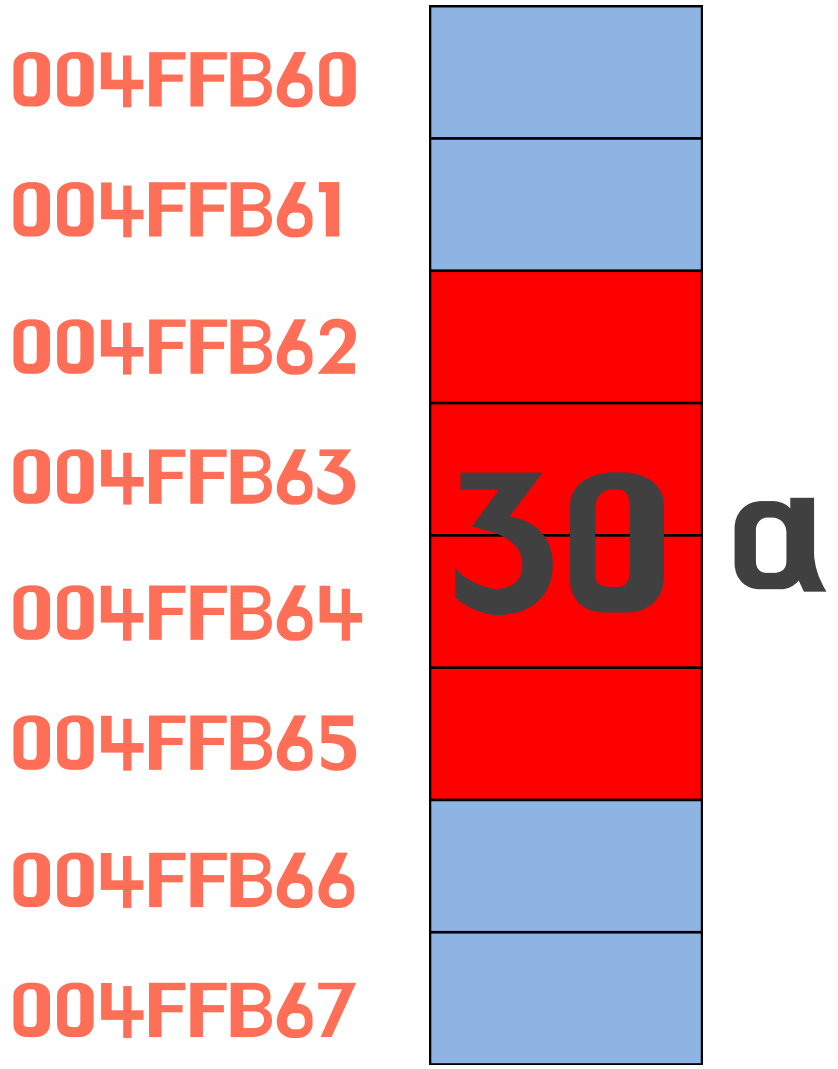
int* ptr = &a; → ptr == &a

cout << a; → 30

cout << &a; → 004FFB62

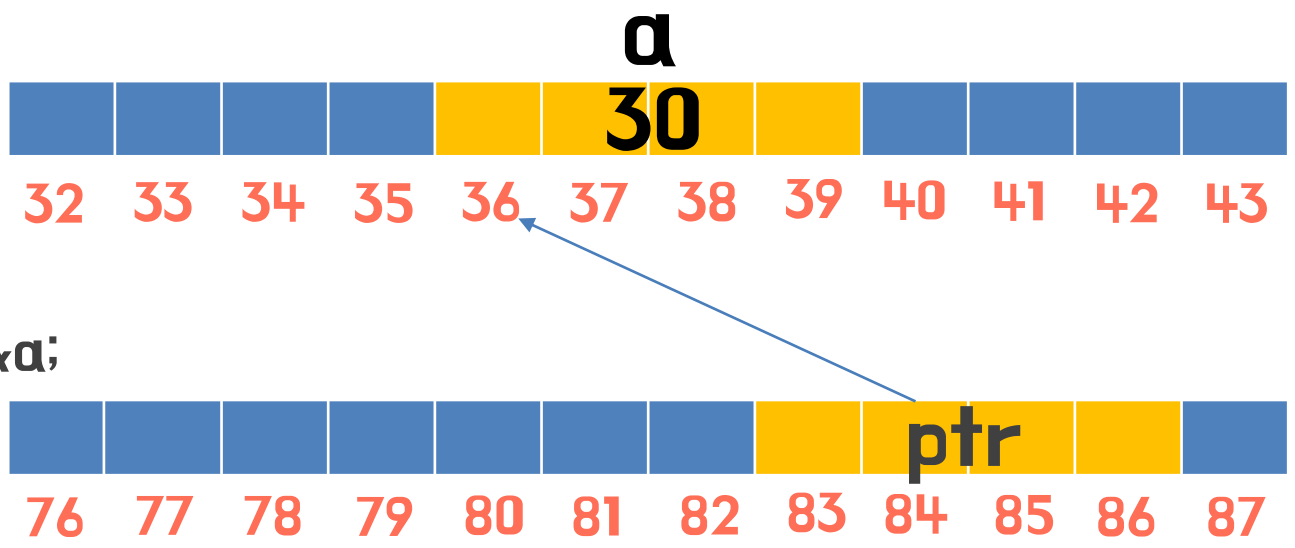
cout << ptr; → 004FFB62

cout << *ptr; → 30



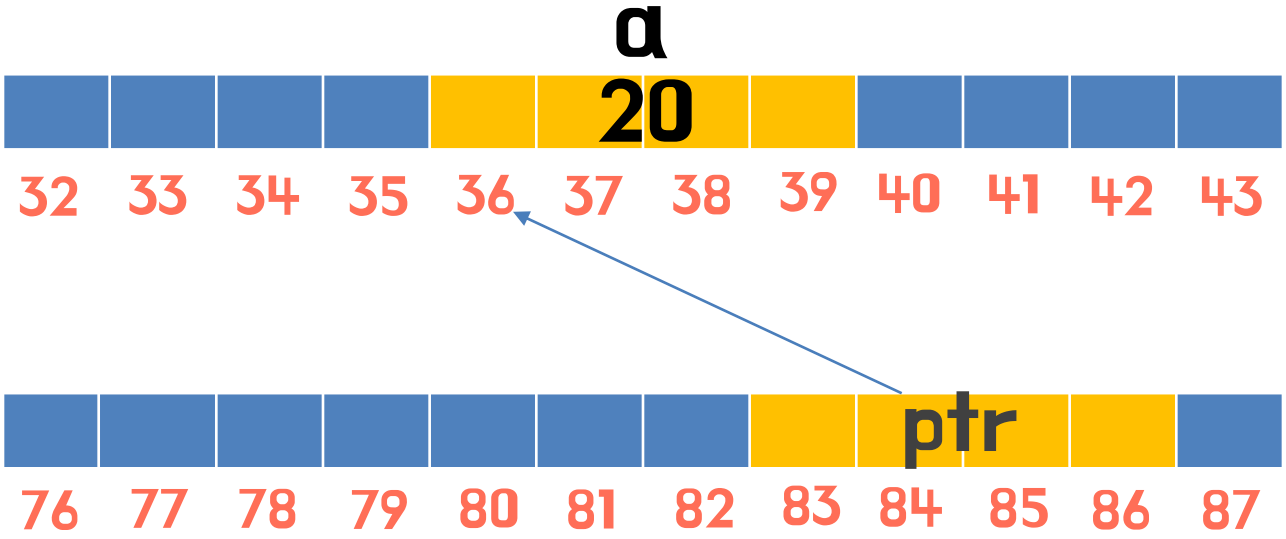
1 포인터

포인터형 변수도 변수이므로 개만의 주소를 가지고 있다.




```
int a = 30;
int *ptr = &a;
```

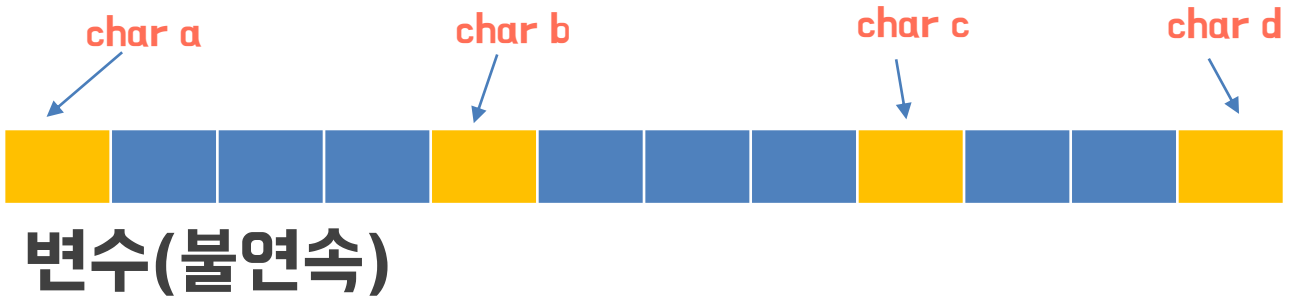
결과값 예상)

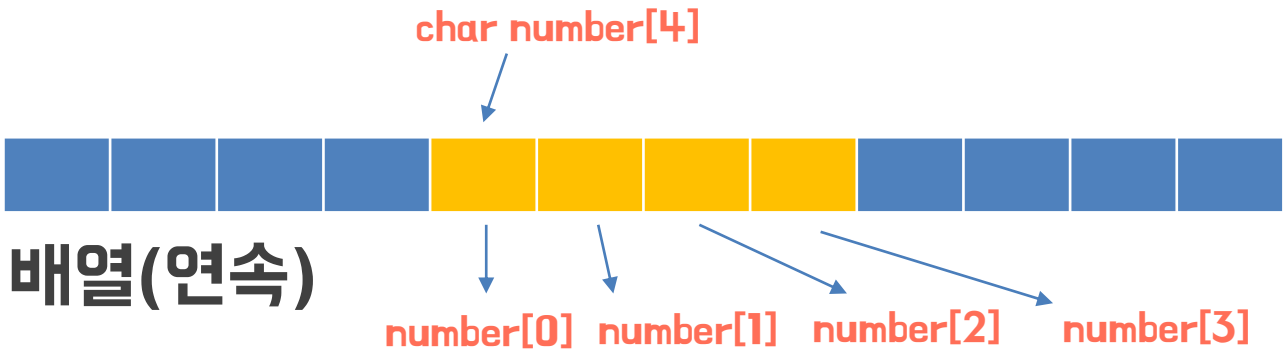


1. cout << a;
2. cout << &a;
3. cout << *a;
4. cout << ptr;
5. cout << &ptr;
6. cout << *ptr;

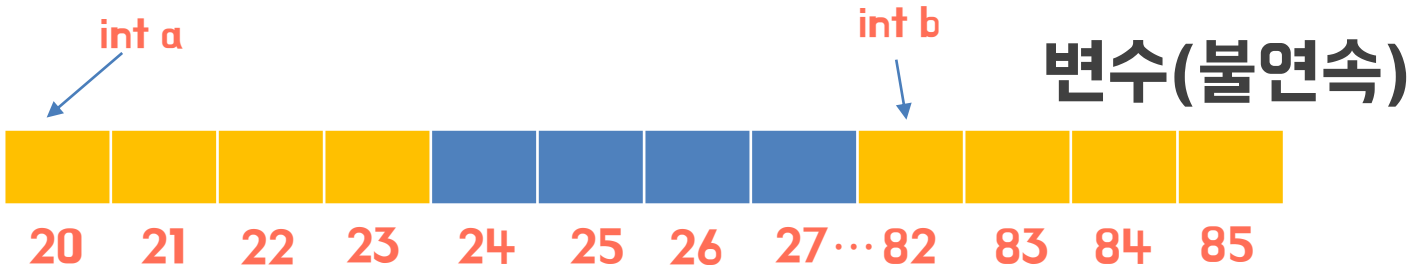
1 포인터

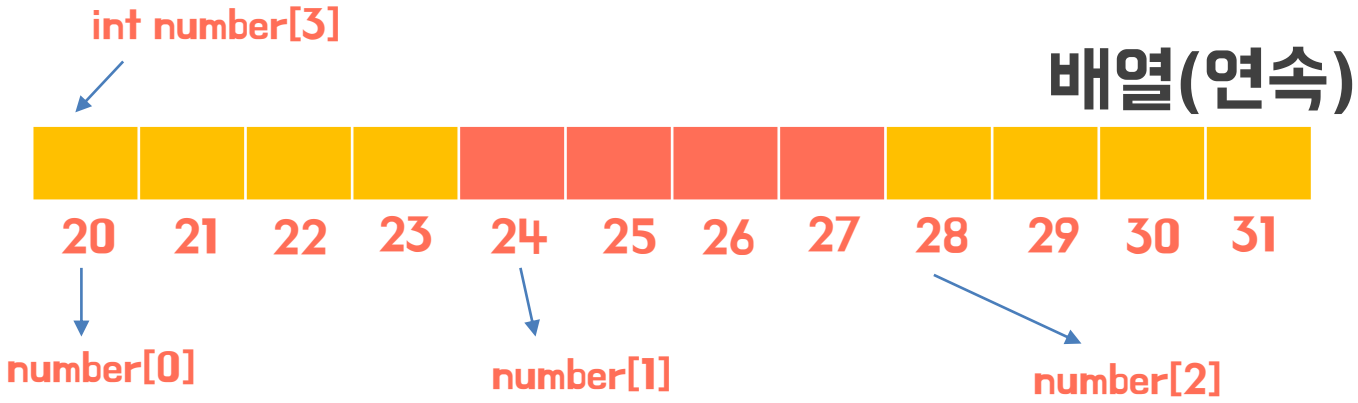
메모리 구조


char a
char b
char c
char d
변수(불연속)


char number[4]
배열(연속)
number[0]
number[1]
number[2]
number[3]

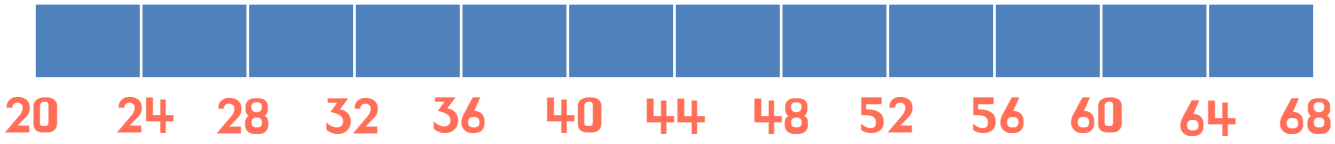
1 포인터


int a
int b
변수(불연속)
20 21 22 23 24 25 26 27 … 82 83 84 85


int number[3]
배열(연속)
20 21 22 23 24 25 26 27 28 29 30 31
number[0]
number[1]
number[2]

1 포인터

배열은 연속적인 메모리 공간을 할당받는다.



int형은 4byte이므로 int a[12]를 선언한다면
4*12 = 48byte가 연속적으로 할당된다.

1 포인터

배열은 연속적인 메모리 공간을 할당받는다.

int num[12];

이 때, num[0], num[1] ... 은 그 요소의 값을 나타내지만
배열의 이름 num은 배열의 시작주소를 나타낸다.

cout << num[0]; ⟶ 1

cout << num; ⟶ 20 (배열의 주소, 실제로는
006FFE2C 와 같은 모양)

1 포인터

```
int num[12];
int *ptr = num;
```

포인터로 배열의 이름을 가리키고 ptr + 1을 하면 자동으로 자료형의 크기만큼 더해져서 다음 요소를 가리키게 된다.

ptr => num[0]의 주소값
ptr + 1 => num[1]의 주소값
ptr + 7 => num[7]의 주소값

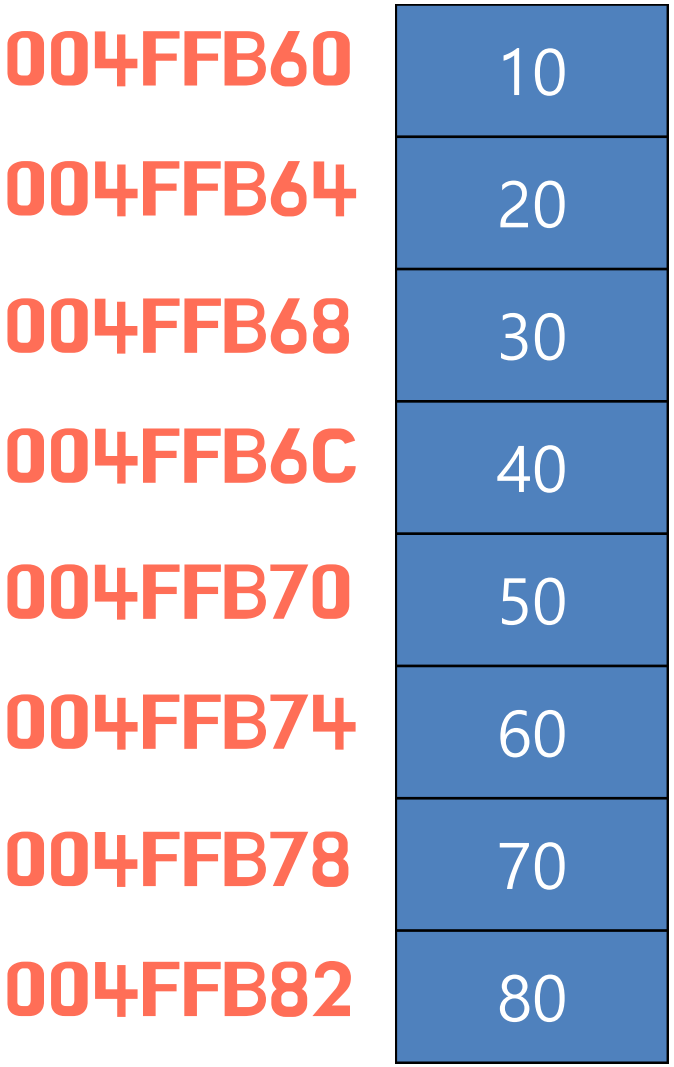
1 포인터

```
cout << num[0];   → 10
cout << num;      → 004FFB60
cout << num + 1   → 004FFB64

int* ptr = num;

cout << ptr;      → 004FFB60
cout << ptr + 1;  → 004FFB64
cout << *ptr;     → 10
cout << *(ptr + 3); → 40
```

int num[8];



1 포인터

정 리

변수의 경우

*으로 포인터형 변수 생성

포인터형 변수는 주소 변수를 저장

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

배열의 경우

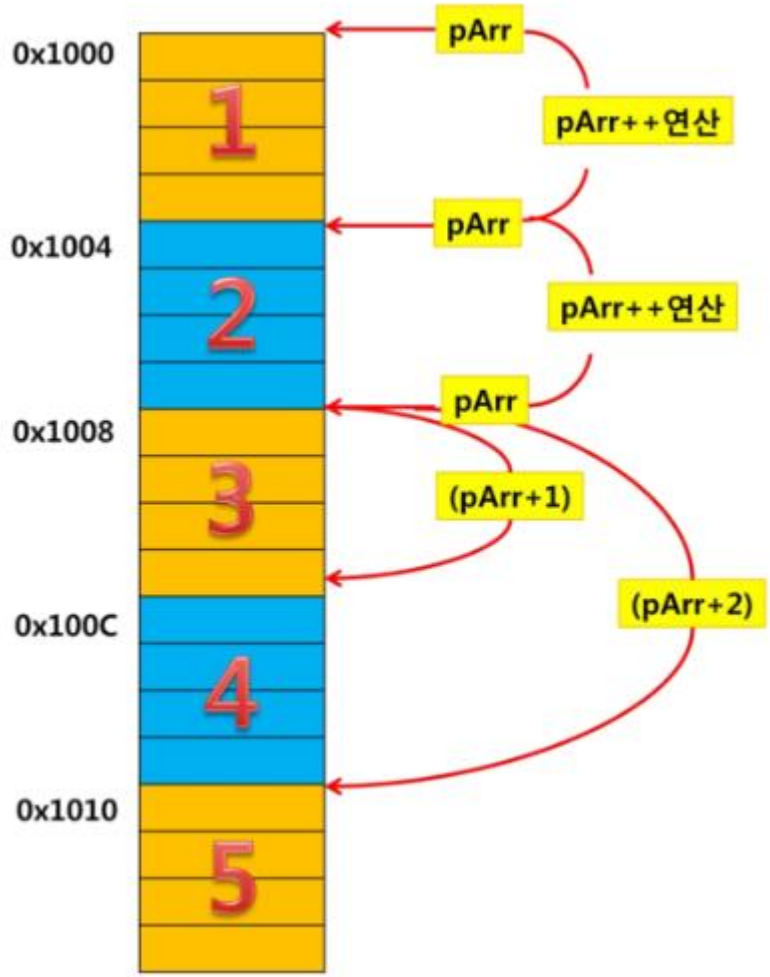
int arr[20];

arr[0], arr[1] 등 원소는 일반 변수

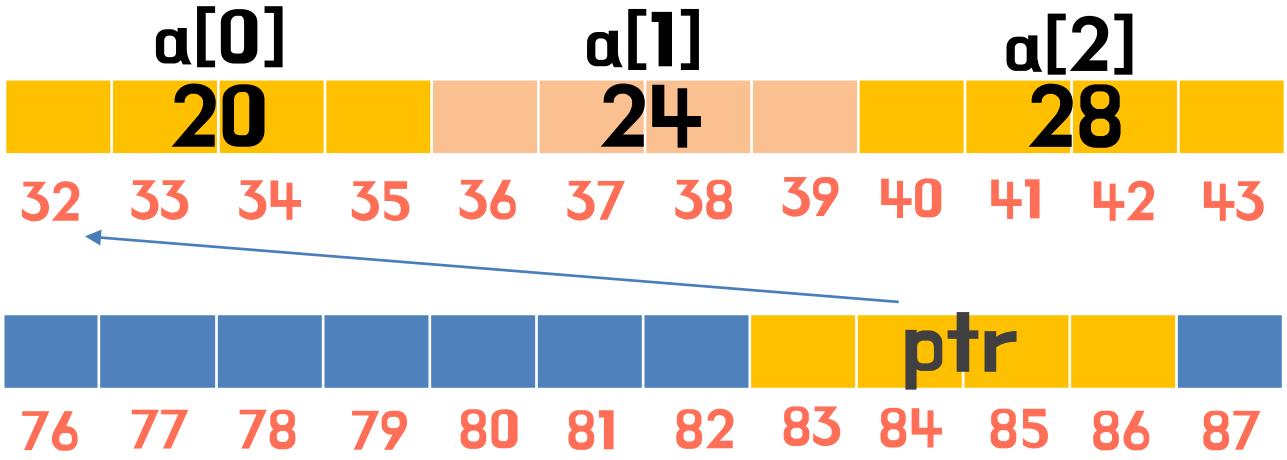
그냥 arr만 쓰면 배열의 주소 변수

```
int *ptr = &arr[0];
int *ptr = arr;
```

1 포인터


0x1000
0x1004
0x1008
0x100C
0x1010
1
2
3
4
5
pArr
pArr++연산
pArr
pArr++연산
pArr
(pArr+1)
(pArr+2)

결과값 예상)



1. int *ptr = &a;
   int *ptr = a;
2. cout << a;
3. cout << &a;
4. cout << *a;
5. cout << ptr;
6. cout << &ptr;
7. cout << *ptr;

ptr++;

8. cout << ptr;
9. cout << &ptr;
10. cout << *ptr;
11. cout << ptr + 1;
12. cout << a + 2;

2 동적 할당

변수의 종류

1. 지역 변수 (지정된 블록 내에서만 사용)

2. 전역 변수 (프로그램 전체에서 사용)

3. static 변수 (지역변수 + 전역변수)

2 동적 할당

지역 변수

지정된 블록 내에서만 사용

값이 유지되지 않음

함수가 실행될 때 메모리에 할당됨

함수가 종료되면 메모리에서 해제됨

2 동적 할당

지역 변수

```
int main()
{
    CallFunction();
    CallFunction();
    CallFunction();
    CallFunction();
}

void CallFunction()
{
    int a = 3;
    cout << a++ << "\n";
}
```

```
C:₩WIND
3
3
3
3
계속하려면
```

값이 유지되지 않음

2 동적 할당

전역 변수

어디서든 사용 가능

값이 유지됨

프로그램이 실행될 때 메모리에 할당됨

프로그램이 종료되면 메모리에서 해제됨

2 동적 할당

전역 변수

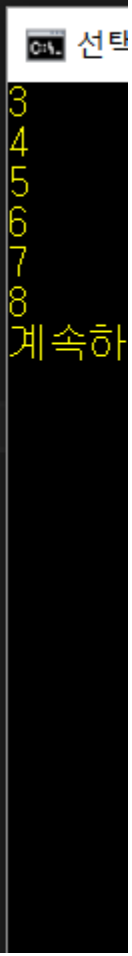
```
int a = 3;

int main()
{
    CallFunction();
    AnotherFunction();
    CallFunction();
    AnotherFunction();
    CallFunction();
    AnotherFunction();
}

void CallFunction()
{
    cout << a++ << "\n";
}

void AnotherFunction()
{
    cout << a++ << "\n";
}
```

프로그램이 실행되는동안 계속 유지



2 동적 할당

static 변수

지정된 블록 내에서만 사용

값이 유지됨

프로그램이 실행될 때 메모리에 할당됨

프로그램이 종료되면 메모리에서 해제됨

2 동적 할당

static 변수

```
int main()
{
    CallFunction();
    AnotherFunction();
    CallFunction();
    AnotherFunction();
    CallFunction();
    AnotherFunction();
}

void CallFunction()
{
    static int a = 3;
    cout << "Call : " << a++ << "\n";
}

void AnotherFunction()
{
    static int a = 10;
    cout << "Another : " << a++ << "\n";
}
```

```
C:₩WINDOWS₩s
Call : 3
Another : 10
Call : 4
Another : 11
Call : 5
Another : 12
계속하려면 아무
```

프로그램이 실행되는동안 계속 유지

2 동적 할당

메모리 영역

Program code 영역 : 실행한 프로그램의 코드를 저장

Data 영역 : 전역변수와 static 변수가 할당,
프로그램 종료시까지 남아있음

Heap 영역 : 동적으로 할당되는 메모리 영역

Stack 영역 : 지역변수와 매개변수가 할당,
함수를 빠져나가면 자동으로 소멸됨

2 동적 할당

동적 메모리 할당

메모리 할당과 해제가 알아서 이루어지는 다른 영역과는 달리, Heap 영역에 동적 할당되는 메모리는 **개발자가 직접** 관리한다.

동적 메모리 할당

동적 할당은 정적 할당에 비해 쓰기 복잡하고 더 느리다.

그렇다면 어떤 경우에? 왜? 사용할까?

2 동적 할당

동적 메모리 할당

1. 프로그램에서 유동적으로 메모리 크기를 할당하고 싶을 때
   (배열의 크기를 상황에 따라 다르게)

2. 매우 많은 크기의 메모리를 할당하고 싶을 때

2 동적 할당

new로 할당하고 delete로 해제!

변수

```
int* a = new int;
*a = 5;
delete a;
```

배열

```
int* arr = new int[10];
arr[3] = 7;
delete[] arr;
```

2 동적 할당

동적 메모리 할당

```
int main()
{
    int* a = new int;

    int N;
    cin >> N;
    int* arr = new int[N];

    arr[2] = 3;

    delete a;
    delete[] arr;
}
```

배열의 크기를 유동적으로 정해줄 수 있다.

3 구조체

구조체

관련 있는 변수들을 묶어서 하나로 관리하는 것

3 구조체

구조체

예를 들어 학생 여러명의 성적을 저장하려고 하면
국어, 영어, 수학 등등 수많은 정보가 필요할 텐데,

```
int korean[100];
int math[100];
int english[100];
```

이런 식으로 배열을 여러 개 만드는
짓은 너무 비효율적이기 때문에
하나로 묶어서 관리하기 위해 구조체
라는 개념이 탄생했다

3 구조체

구조체 정의

```
struct 구조체 이름
{

    담을 변수 1, 2, 3, ...

};
```

구조체 이름이 타입이름이 된다
(int, char처럼)

세미 빼먹지 마셈

3 구조체

구조체 사용

구조체.변수가 기본 모양
(점찍는거만 기억!)

ex) struct.data = 7;
cout << struct.data;

3 구조체

```
struct Student
{
    int korean;
    int math;
    int english;
};

int main()
{
    Student arr[100];

    arr[0].korean = 10;
    arr[0].math = 20;
    arr[0].english = 70;

    arr[1] = { 10, 20, 30 };

    Student student{ 10,20,30 };
}
```

구조체 선언

담길 변수들

Student 타입의 크기가 100인 배열 선언

각각의 변수에 접근

각각의 변수에 접근(다른 방법)

생성과 동시에 초기화

감사합니다.

Made by 규정