



땅울림 객체 스터디

4주차

1

디버깅

2

포인터

3

레퍼
런스

4

배열
포인터

1 디버깅

디버깅

=> 컴퓨터 프로그램의 정확성이나 논리적인 오류(버그)를 찾아내는 과정

1 디버깅

프로그램이 **예상과 다르게** 동작할 때,
어디가 잘못됐는지 찾아서 고치는 과정

1 디버깅

C:\WINDOWS\system32\cmd.exe

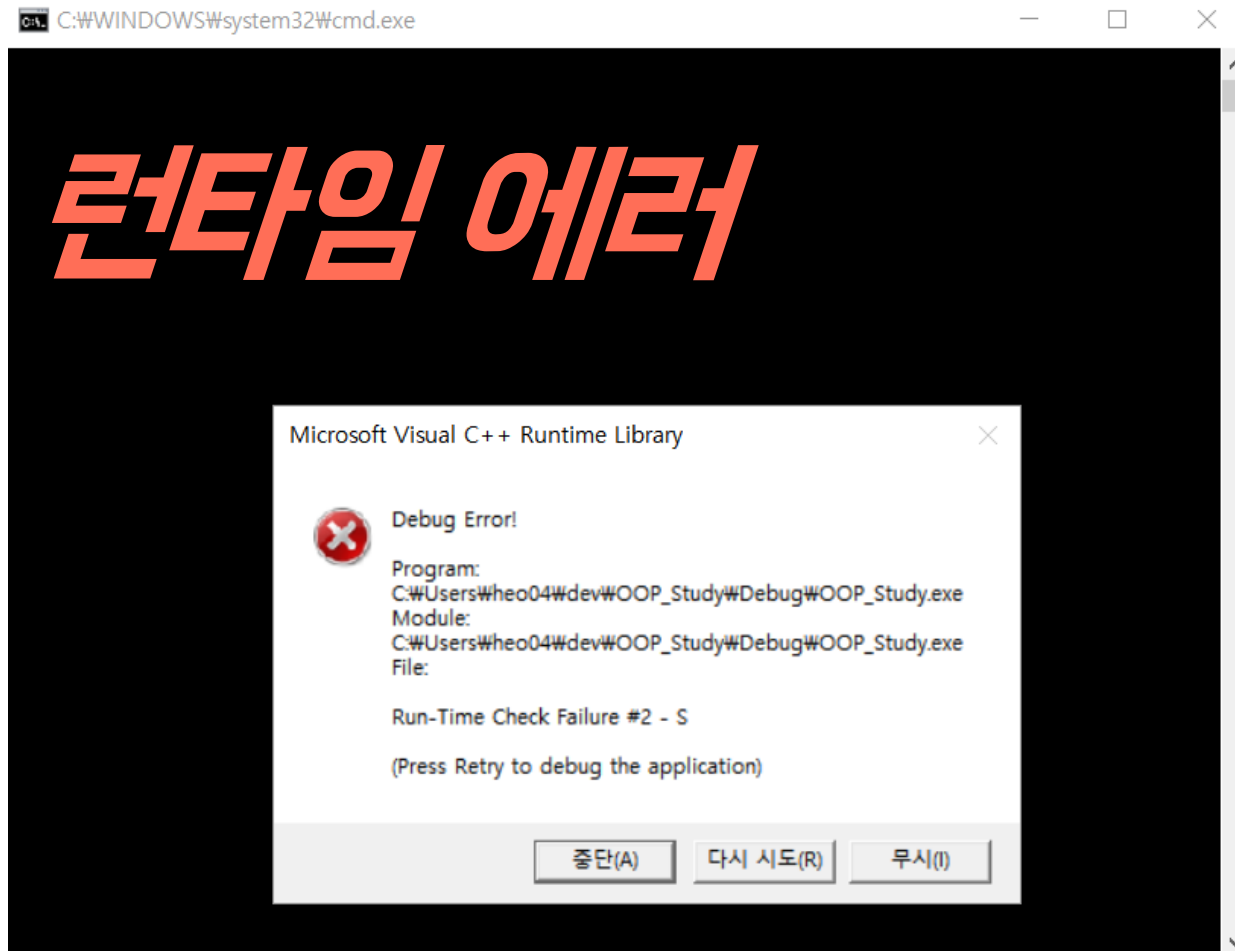
a와 b를 입력하십시오.

2 3

a + b = 6계속하려면 아무 키나 누르십시오 . . .

예상과 다르게 동작할 때

1 디버깅



1 디버깅

```
4  
5 int main()  
6 {  
7     int a, b;  
8  
9     cout << "a와 b를 입력하시오.\n";  
10  
11     cin >> a >> b;  
12  
13     cout << "a + b = " << a * b;  
14  
15     return 0;  
16 }
```

이 정도야 뭐.. 쉽게 고칠 수 있음

1 디버깅

```
501
502     /* 복호화를 위해 ct.bin 파일을 읽어옴 */
503     string encrypted_text = ReadFile("ct.bin");
504
505     /* 복호화 진행 */
506     for (int i = 0; i < encrypted_text.size() / 32; i++)
507     {
508         string p_text = encrypted_text.substr(i * 32, 32);
509         dec += Decrypt(p_text);
510     }
511
512     /* pt2.bin에 복호화 결과 출력*/
513     WriteFile("pt2.bin", dec);
514
515     return 0;
516 }
```

몇 백, 몇 천줄 되는 코드에서 오류가 난다면?

어디가 틀렸는지 찾는데만 하루종일 걸림 => **개 비효율**

1 디버깅

개발은 코드 작성 시간 30% + 디버깅 시간 70%라고
할만큼 프로그래밍에서 가장 중요한 능력

디버깅만 잘해도 코딩 속도가 엄청나게 빨라짐

그리고 문제 풀때나 과제할 때, 너네 스스로 해결할
수 있게 됨

그니까 **무조건** 할 줄 알아야함!!

1 파일 관리

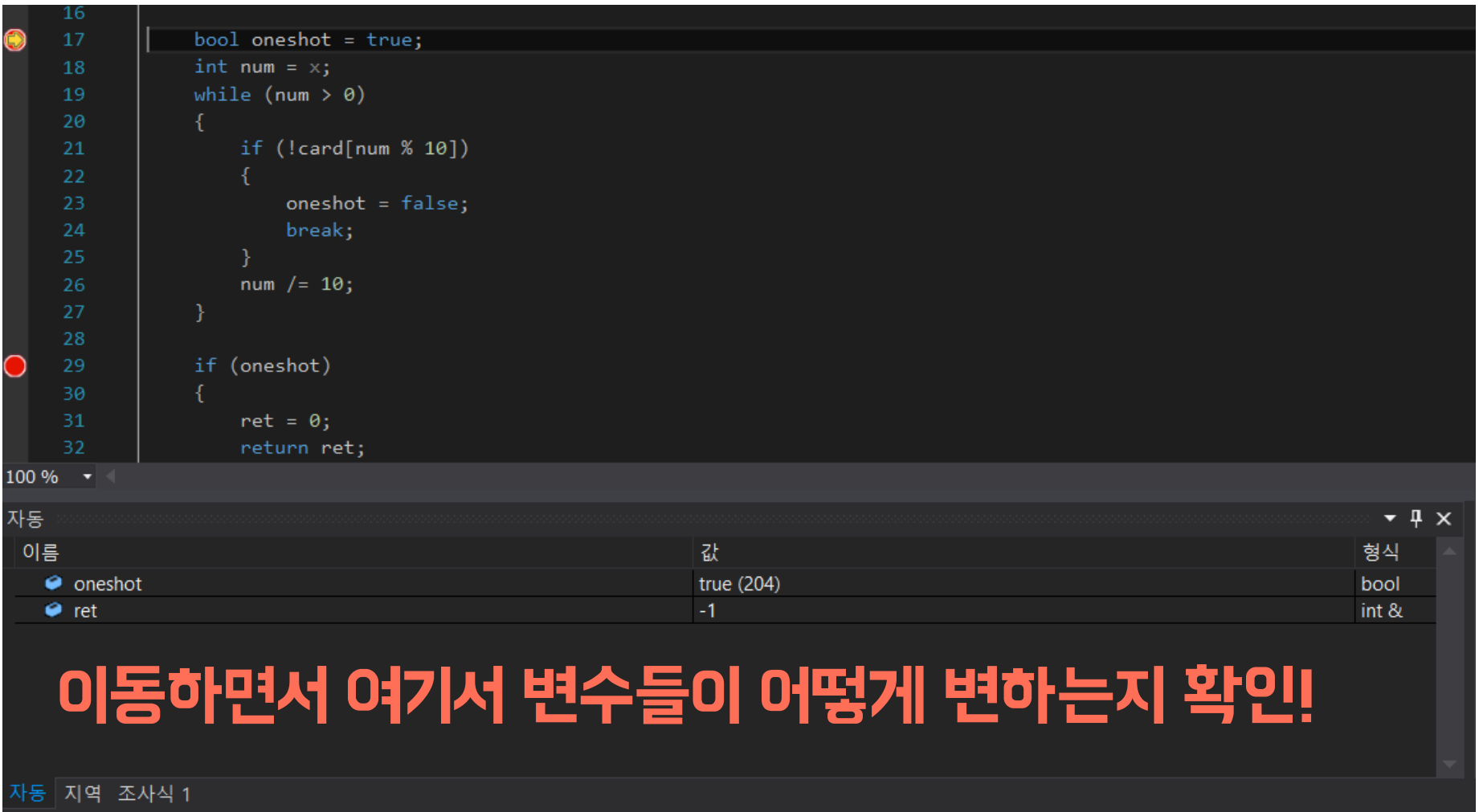
디버깅 하는 방법

=> 중단점과 F5, F10만 기억!

1. 이상하다 싶은 곳
에 중단점 설정
(모르겠으면 아무 데
나 막 찍어도 됨)

```
11 int Solve(int x)
12 {
13     int &ret = DP[x];
14     if (ret != -1)
15         return ret;
16
17     bool oneshot = true;
18     int num = x;
19     while (num > 0)
20     {
21         if (!card[num % 10])
22         {
23             oneshot = false;
24             break;
25         }
26         num /= 10;
27     }
28
29     if (oneshot)
30     {
31         ret = 0;
32         return ret;
33     }
34     int _min = 1e9;
35
```

2. F5 누르면서 다음 중단점으로 이동 (한 줄씩 이동하고 싶으면 F10)



The screenshot shows a C++ IDE with a code editor and a variable watch window. The code editor displays the following code:

```
16  
17 bool oneshot = true;  
18 int num = x;  
19 while (num > 0)  
20 {  
21     if (!card[num % 10])  
22     {  
23         oneshot = false;  
24         break;  
25     }  
26     num /= 10;  
27 }  
28  
29 if (oneshot)  
30 {  
31     ret = 0;  
32     return ret;
```

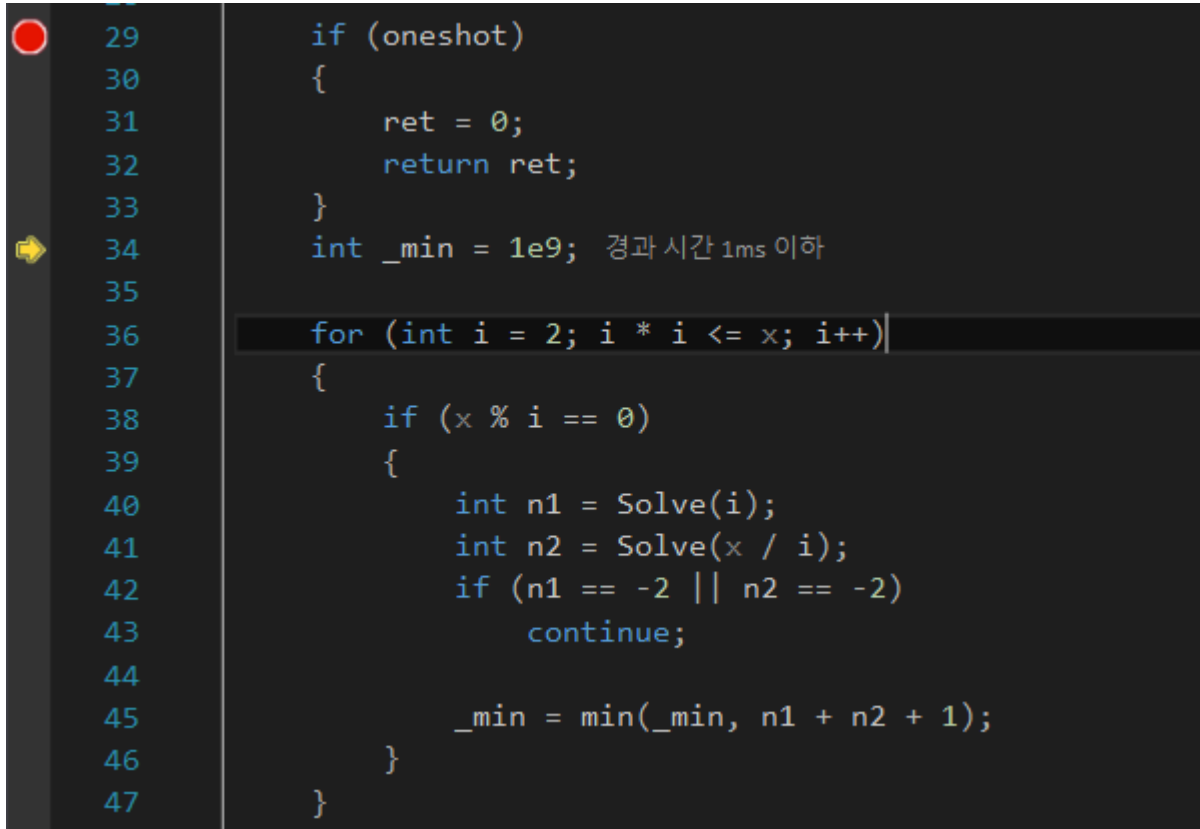
The variable watch window, titled "자동", shows the following variables and their values:

| 이름 | 값 | 형식 |
|---------|------------|-------|
| oneshot | true (204) | bool |
| ret | -1 | int & |

Below the variable watch window, the text "이동하면서 여기서 변수들이 어떻게 변하는지 확인!" is displayed in large, bold, orange font.

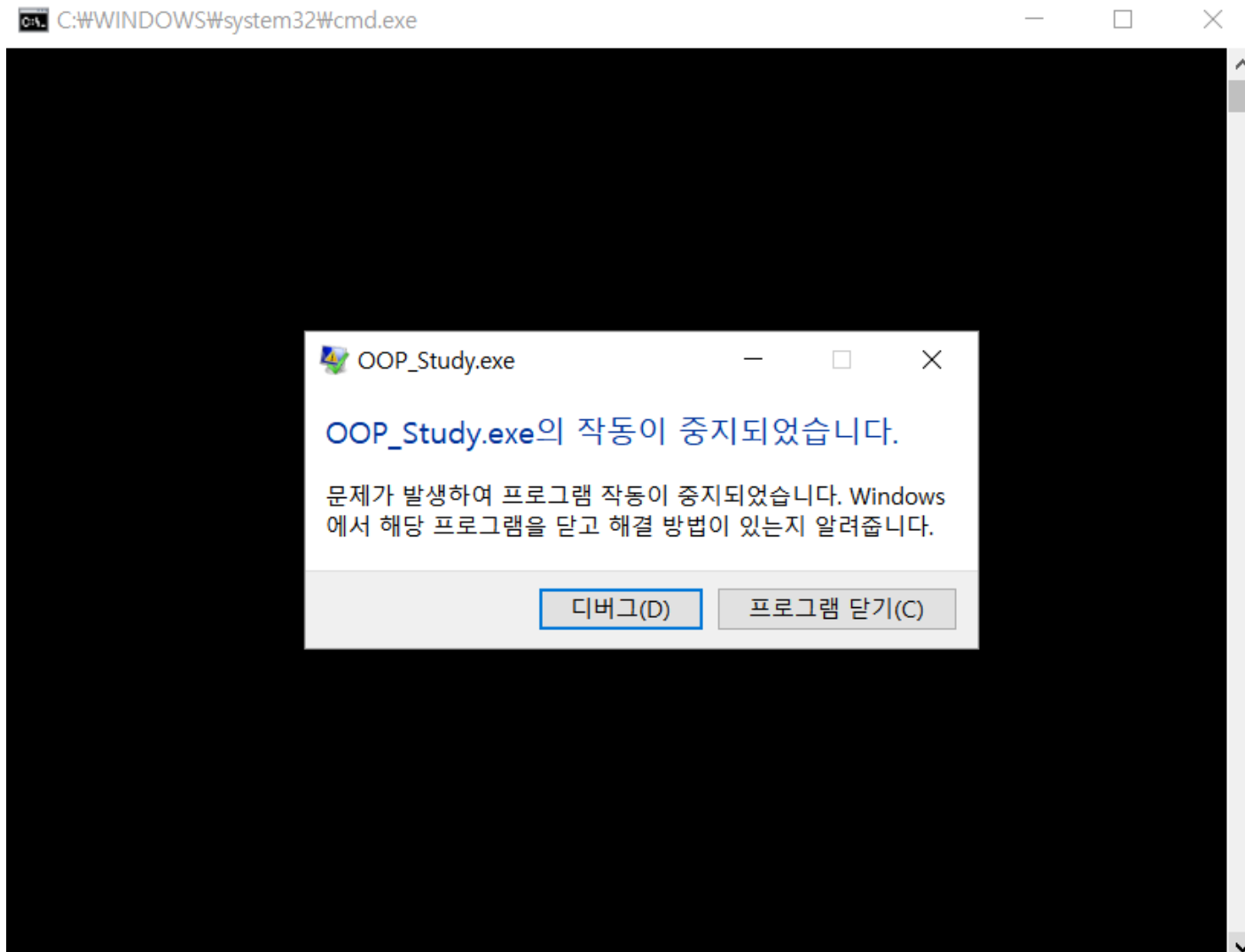
자동 지역 조사식 1

3. 다 검사했으면 Shift + F5 눌러서 종료



```
29     if (oneshot)
30     {
31         ret = 0;
32         return ret;
33     }
34     int _min = 1e9; 경과 시간 1ms 이하
35
36     for (int i = 2; i * i <= x; i++)
37     {
38         if (x % i == 0)
39         {
40             int n1 = Solve(i);
41             int n2 = Solve(x / i);
42             if (n1 == -2 || n2 == -2)
43                 continue;
44
45             _min = min(_min, n1 + n2 + 1);
46         }
47     }
```

디버깅 예시)



여기저기 중단점 다 찍
어보고 F5 누르면서
어디서 에러나는지 확
인하면 끝(3초컷)

```
5  int main()  
6  {  
7      int num[300] = { 0 };  
8      int count[500] = { 0 };  
9      int cost[40] = { 0 };  
10  
11     for (int i = 0; i < 300; i++)  
12     {  
13         num[i] = i;  
14     }  
15  
16     for (int i = 0; i < 500; i++)  
17     {  
18         count[i] = 2 * i;  
19     }  
20  
21     for (int i = 0; i < 10000; i++)  
22     {  
23         cost[i] = 3 * i;  
24     }  
25  
26     return 0;  
27 }
```

1
2
3
4

```

10
11     for (int i = 0; i < 300; i++)
12     {
13         num[i] = i;
14     }
15
16     for (int i = 0; i < 500; i++)
17     {
18         count[i] = 2 * i;
19     }
20
21     for (int i = 0; i < 10000; i++)
22     {
23         cost[i] = 3 * i;
24     }
25
26     return 0;
27 }

```

Microsoft Visual Studio



예외 발생(0x00991F9A, OOP_Study.exe): 0xC0000005: 0x00D40000 위치를 기록하는 동안 액세스 위반이 발생했습니다..

이 예외에 대한 처리기가 있으면 프로그램을 안전하게 계속할 수 있습니다.

☒ 이 예외 형식이 throw되면 중단

[예외 설정 중단 및 열기\(S\)](#)

중단(B)

계속(C)

무시(I)

3번 => 4번 넘어갈때 에러가 발생했으므로
3~4 사이의 코드만 확인하면 됨

100 %

조사식 1

| 이름 | 값 | 형식 |
|------|--|------------|
| num | 4 | int |
| card | 0x00007ff61e25da88 {false, false, true, true, false, false, true, false, false, false} | bool[10] |
| DP | 0x00007ff61de8d180 {-1, -1, 0, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, ...} | int[10000] |
| _min | 1 | int |

자동 지역 조사식 1

Tip : 자신이 원하는 변수만 보고 싶으면 조사식 활용

조사식 1

| 이름 | 값 | 형식 |
|-----|--|------------|
| DP | 0x00007ff61de8d180 {-1, -1, 0, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, ...} | int[10000] |
| [0] | -1 | int |
| [1] | -1 | int |
| [2] | 0 | int |
| [3] | -1 | int |
| [4] | 1 | int |
| [5] | -1 | int |
| [6] | -1 | int |

자동 지역 조사식 1

Tip : 배열은 이 버튼 누르면 자세히 볼 수 있음



```

5  int main()
6  {
7      int num = 1;
8
9      for (int i = 0; i < 50; i++)
10     {
11         num *= 2;
12     }
13
14     return 0;
15 }

```



**Tip : 반복문에서 변수가 어떻게 변하는지 보고 싶으면
반복문 내부에 중단점 설정!**

조사식 1

| 이름 | 값 |
|---|---|
|  i | 0 |
|  num | 1 |

i : 0일 때 => num : 1

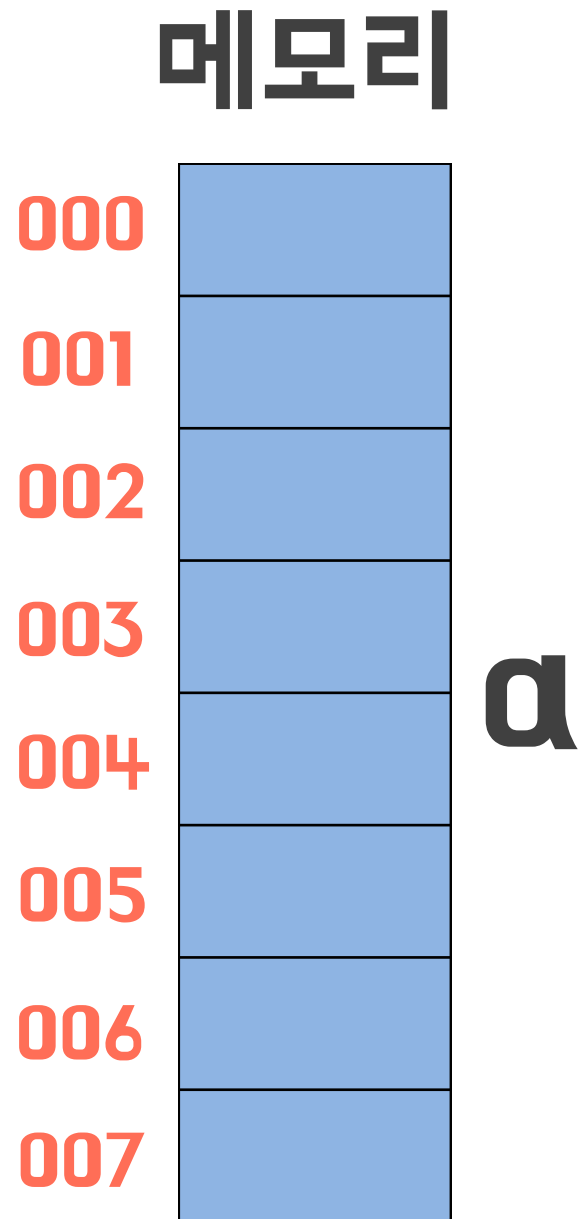
조사식 1

| 이름 | 값 |
|---|----|
|  i | 5 |
|  num | 32 |

i : 5일 때 => num : 32

2 포인터

```
int a = 30;
```



2 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

2 포인터

포인터형 변수 만들기

변수는 각각의 타입에 맞는 값을 넣어줘야함!

(int)형 변수는 정수를 저장, (char)형 변수는 문자를 저장

(int *)형 변수는 int변수의 주소를 저장

(double *)형 변수는 double변수의 주소를 저장

2 포인터

**일반 변수는 값을 저장하고,
포인터형 변수는 일반 변수의 주소를 저장한다.**

2 포인터

*과 &는 한 세트

```
int a = 30;
```

변수의 주소를 저장하고 싶으면? => `int* ptr = &a;`

(`int *`)은 포인터형 변수이므로 **주소만 저장**할 수 있음!

~~`int *ptr = a;`~~

~~`int ptr = &a;`~~

2 포인터

출력할 때

앞에 *이 붙어있으면 => 값 출력

앞에 &이 붙어있으면 => 주소 출력

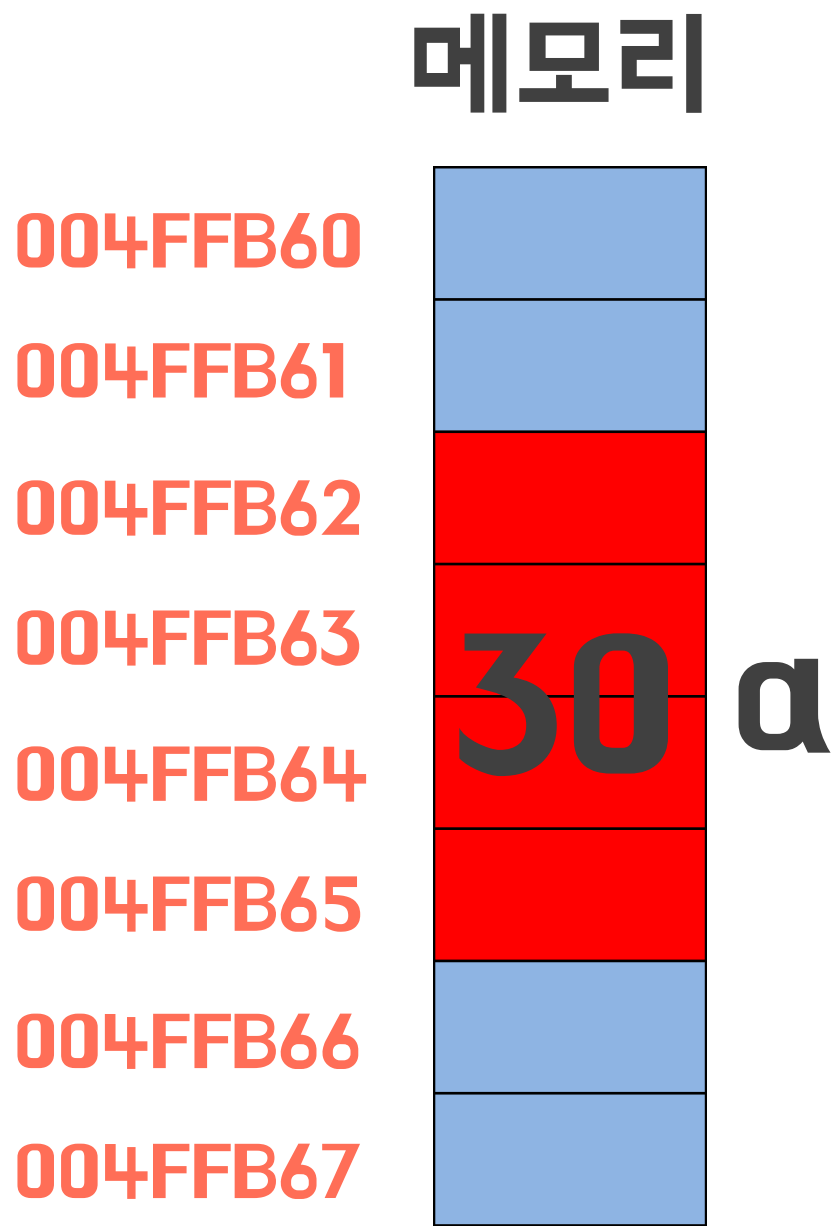
2 포인터

&: 변수의 주소를 알고 싶을 때!

`int a = 30;`

`cout << a;` → **30**

`cout << &a;` → **004FFB62**



2 포인터

*****: 주소의 내용을 알고 싶을 때!

```
int a = 30;
```

```
cout << a;           → 30
```

```
cout << &a;          → 004FFB62
```

```
cout << *(&a);       → 30
```

004FFB60

004FFB61

004FFB62

004FFB63

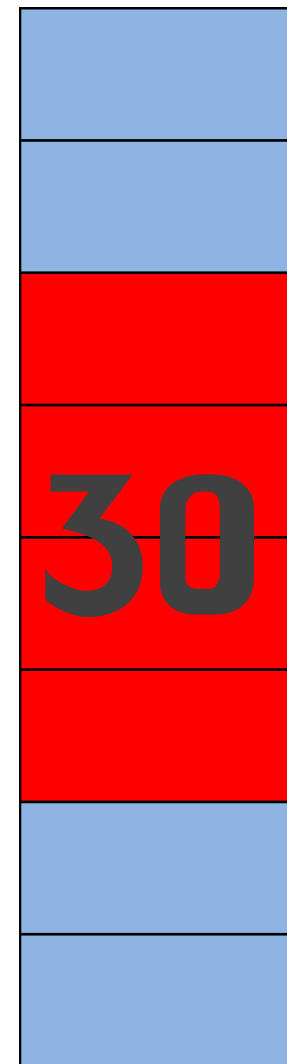
004FFB64

004FFB65

004FFB66

004FFB67

메모리



2 포인터

*****: 주소의 내용을 알고 싶을 때!

```
int a = 30;
```

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

```
Ptr == &a
```

```
cout << a;           → 30
```

```
cout << &a;          → 004FFB62
```

```
cout << *(&a);        → 30
```

```
cout << *ptr;          → 30
```

```
cout << &(*ptr);      → 004FFB62
```

004FFB60

004FFB61

004FFB62

004FFB63

004FFB64

004FFB65

004FFB66

004FFB67

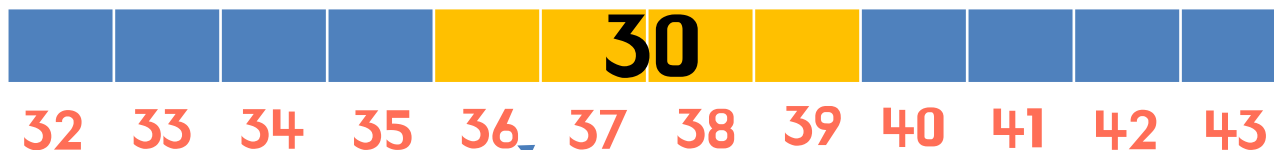
메모리



2 포인터

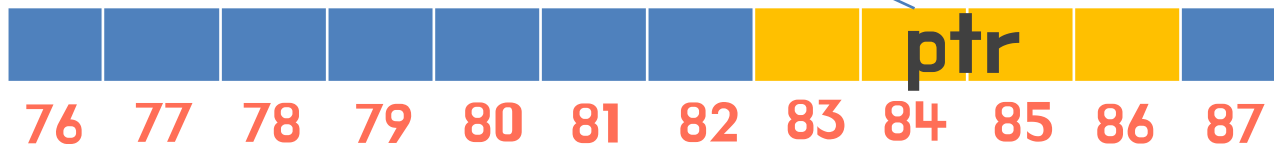
포인터형 변수도 변수이므로
개만의 주소를 가지고 있다.

a

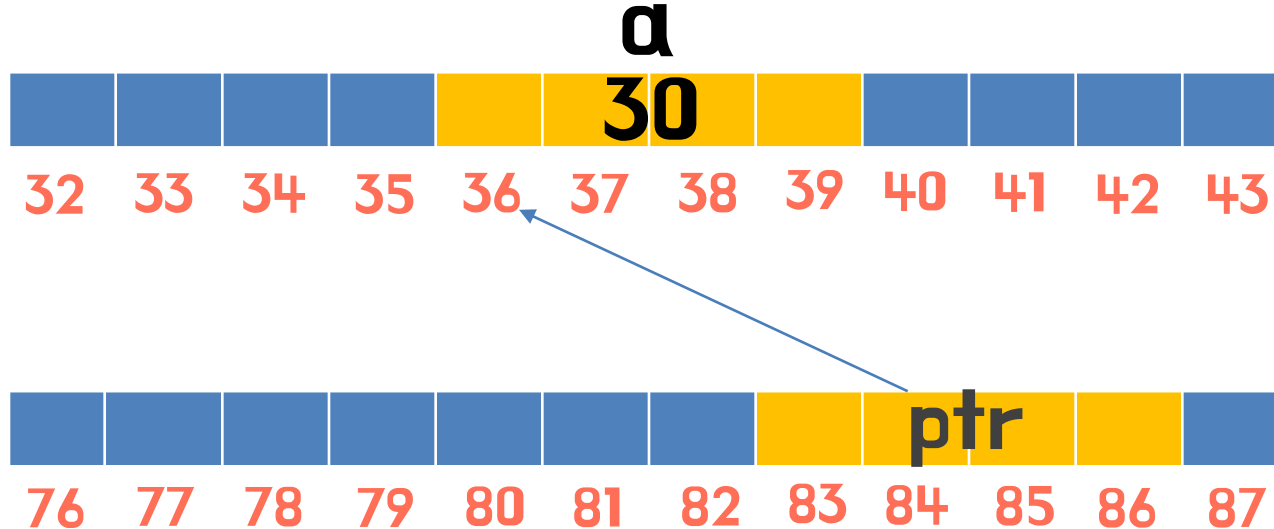


`int a = 30;`

`int *ptr = &a;`



결과값 예상)



1. `cout << a;` **30**
2. `cout << &a;` **36**
3. `cout << *a;` **X**
4. `cout << ptr;` **36**
5. `cout << &ptr;` **83**
6. `cout << *ptr;` **30**

3 레퍼런스

레퍼런스?

=> 변수에 또 다른 별명을 붙여주는 것

3 레퍼런스

값에 의한 대입

```
int b = 20;  
int r = b;
```

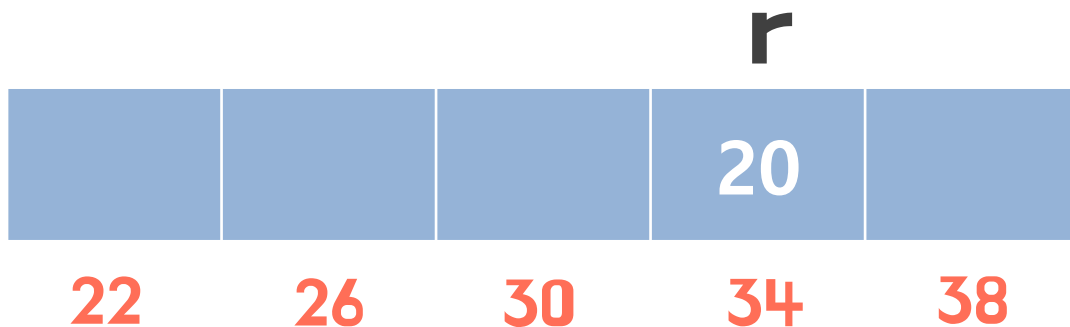
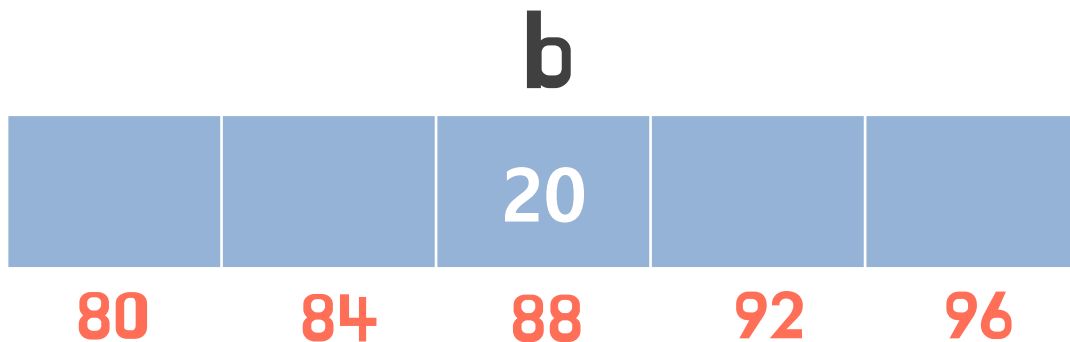
```
cout << b; → 20
```

```
cout << r; → 20
```

```
r += 10;
```

```
cout << b; → 20
```

```
cout << r; → 30
```

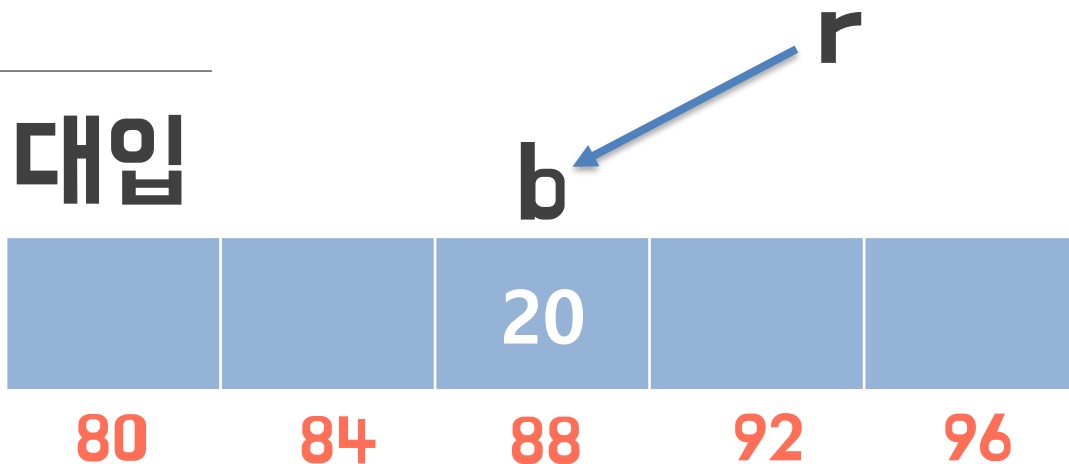


별도의 메모리

3 레퍼런스

레퍼런스에 대한 대입

```
int b = 20;  
int& r = b;
```



```
cout << b; —→ 20
```

```
cout << r; —→ 20
```

```
r += 10;
```

```
cout << b; —→ 30
```

```
cout << r; —→ 30
```

3 레퍼런스

Call by Value

vs

Call by Reference

3 레퍼런스

Call by Value

```
int main()
{
    int a = 30;
    int b = 70;

    swap(a, b);

    cout << "a : " << a << ", b : " << b;
}

void swap(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

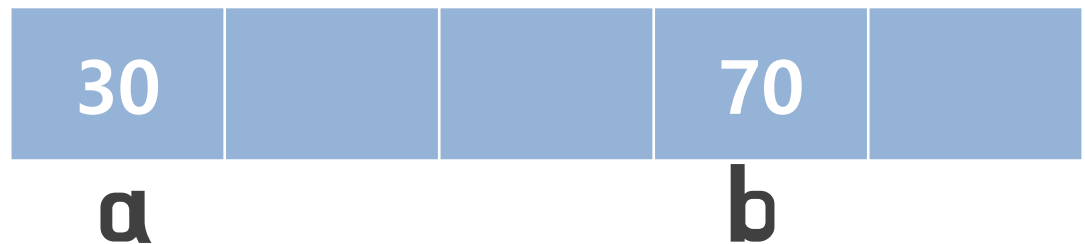
a : 30 b : 70

값이 변하지 않음!

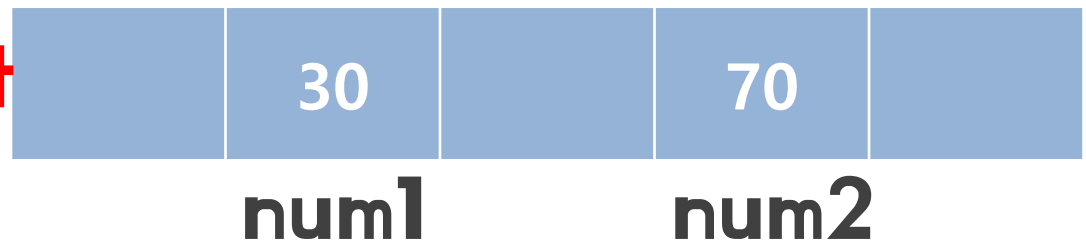
3 레퍼런스

Call by Value

main 함수의 메모리



swap 함수의 메모리



swap 함수에서 바꿔봤자
main 함수에는 영향x

3 레퍼런스

Call by Value

```
int main()
{
    int a = 30;
    int b = 70;

    swap(a, b);

    cout << "a : " << a << ", b : " << b;
}
```

```
void swap(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

int num1 = a;
int num2 = b;

**num1, num2에 a,b를 단순 대입하고
num1과 num2를 바꾼 것
So, a와 b에는 영향이 없음**

3 레퍼런스

Call by Reference

```
int main()
{
    int a = 30;
    int b = 70;

    swap(a, b);

    cout << "a : " << a << ", b : " << b;
}

void swap(int &num1, int &num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

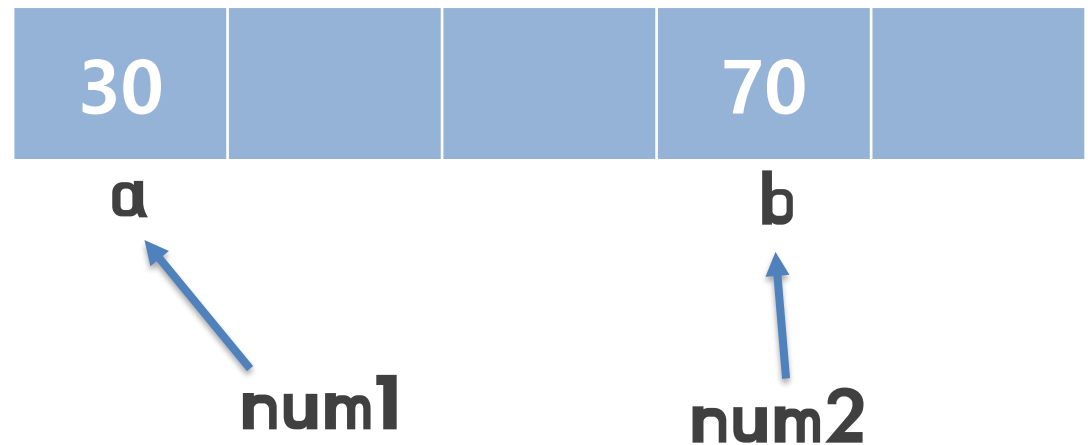
a : 70 b : 30

값이 변함!

3 레퍼런스

Call by Reference

main 함수의 메모리



swap 함수에서 바뀌도
a,b를 바꾸는 거랑 똑같음

3 레퍼런스

Call by Reference

```
int main()
{
    int a = 30;
    int b = 70;

    swap(a, b);

    cout << "a : " << a << ", b : " << b;
}
```

```
void swap(int &num1, int &num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

num1과 num2가 a와 b의 별명
num1, num2를 바꿔도
a와 b를 바꾸는 것과 똑같음

int &num1 = a;
int &num2 = b;

3 배열

정리

선언할 때

*으로 포인터형 변수 생성

포인터형 변수는 주소 변수를 저장

&로 레퍼런스형 변수 생성

일반 변수랑 똑같이 사용

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

사용할 때

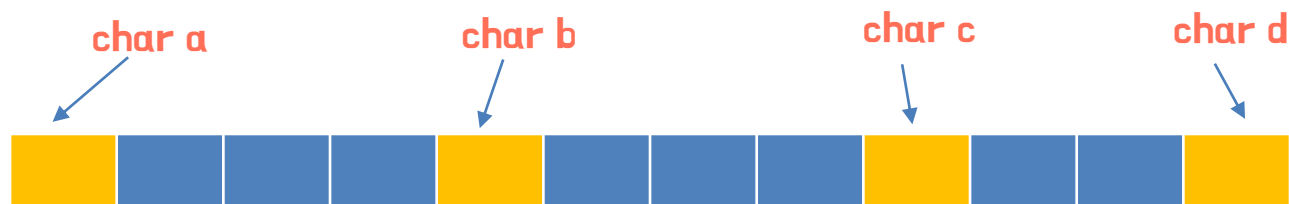
*으로 주소 변수에 접근, 값 받아옴

&으로 일반 변수에 접근, 주소 받아옴

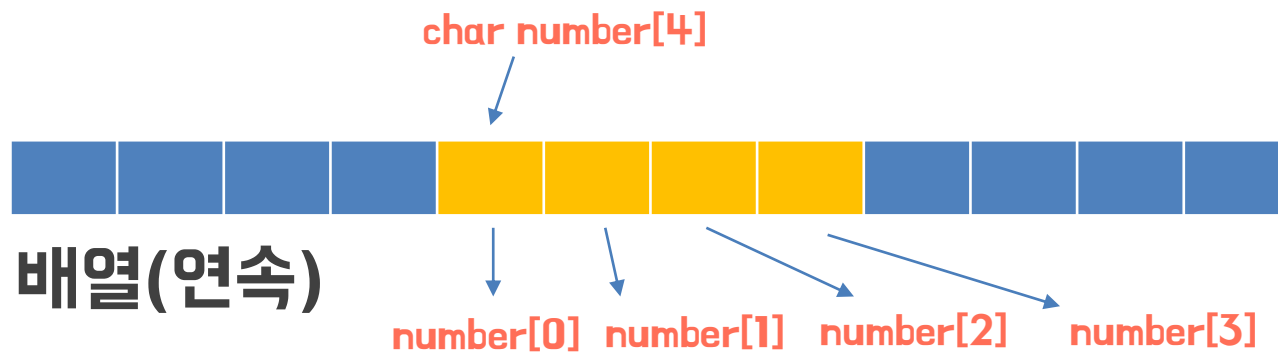
```
cout << *ptr;  
cout << &a;
```


4 배열과 포인터

메모리 구조

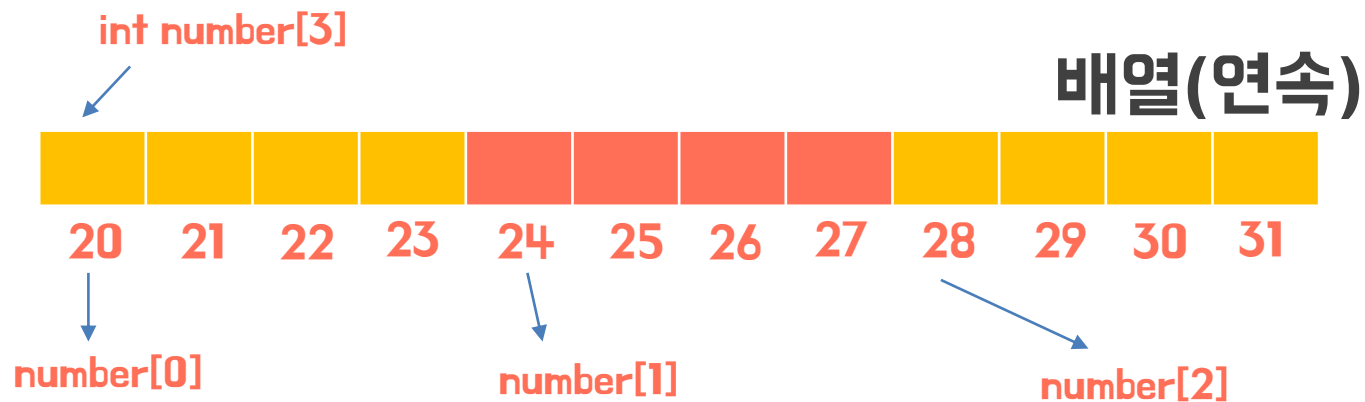
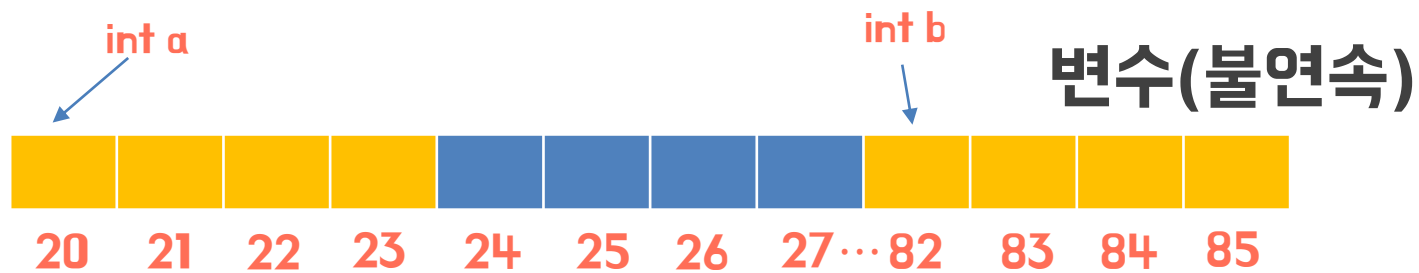


변수(불연속)



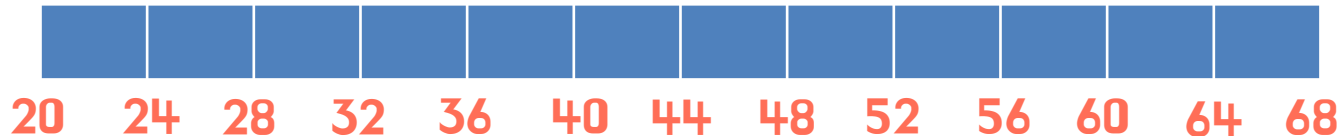
배열(연속)

4 배열과 포인터



4 배열과 포인터

배열은 연속적인 메모리 공간을 할당받는다.



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

4 배열과 포인터

배열은 연속적인 메모리 공간을 할당받는다.

`int num[12];`

| | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | |
| 20 | 24 | 28 | 32 | 36 | 40 | 44 | 48 | 52 | 56 | 60 | 64 | 68 |

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

`cout << num[0];` → 1

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

4 배열과 포인터

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

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

$ptr \Rightarrow num[0]$ 의 주소값

$ptr + 1 \Rightarrow num[1]$ 의 주소값

$ptr + 7 \Rightarrow num[7]$ 의 주소값

4 배열과 포인터

int num[8];

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**

004FFB60

10

004FFB64

20

004FFB68

30

004FFB6C

40

004FFB70

50

004FFB74

60

004FFB78

70

004FFB82

80



감사합니다.

Made by 규정
