ROBIT
ROBOT SPORT GAME TEAM

C Language - Pointer

ROBIT
ROBOT SPORT GAME TEAM

What is a pointer?

Definition

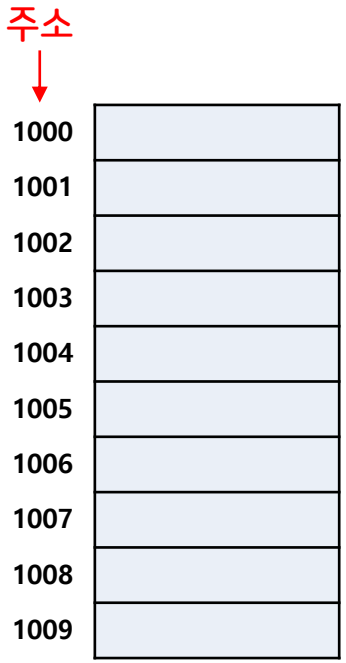
- 포인터(pointer)

  어떤 다른 값이 저장되어 있는 메모리 주소를 가지고 있는 상수/변수

메모리의 구조

- 변수는 메모리에 저장된다.

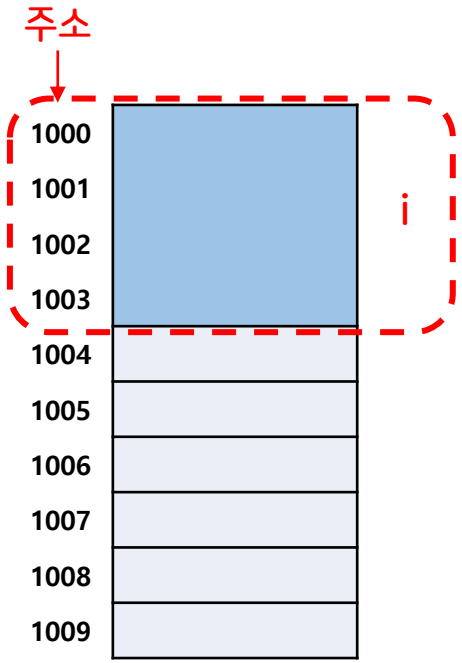
- 메모리는 바이트 단위로 액세스된다.
  - 첫번째 바이트의 주소는 n, 두번째 바이트는 n+1,…



변수와 메모리

- 변수의 자료형에 따라서 차지하는 메모리 공간이 달라진다.
- char형 변수: 1바이트, int형 변수: 4바이트,…

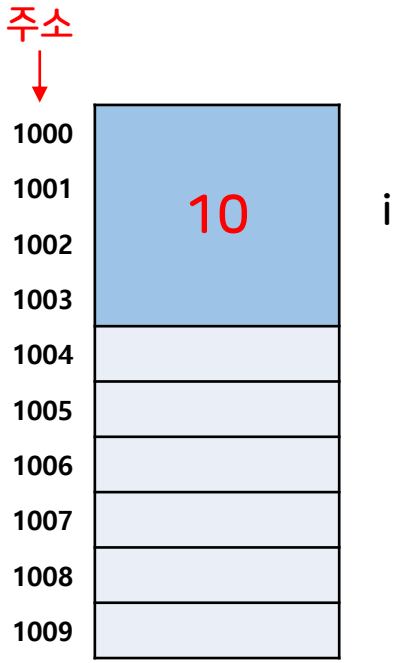
```
int main(void)
{
  int i;
}
```



변수와 메모리

- 변수의 자료형에 따라서 차지하는 메모리 공간이 달라진다.
- char형 변수: 1바이트, int형 변수: 4바이트,…

```
int main(void)
{
  int i;
  i = 10;
}
```

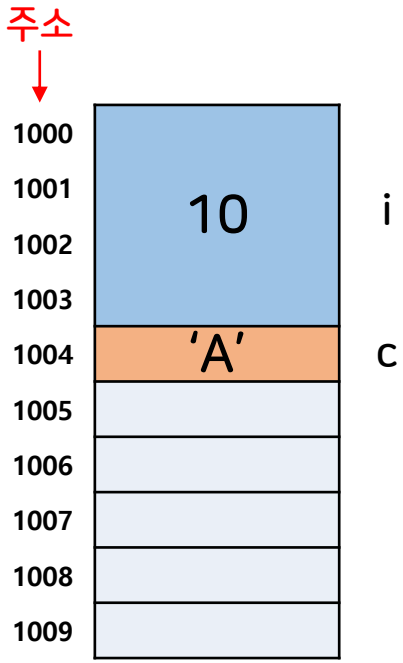


변수와 메모리

ROBIT
ROBOT SPORT GAME TEAM

- 변수의 자료형에 따라서 차지하는 메모리 공간이 달라진다.
- char형 변수: 1바이트, int형 변수: 4바이트,…

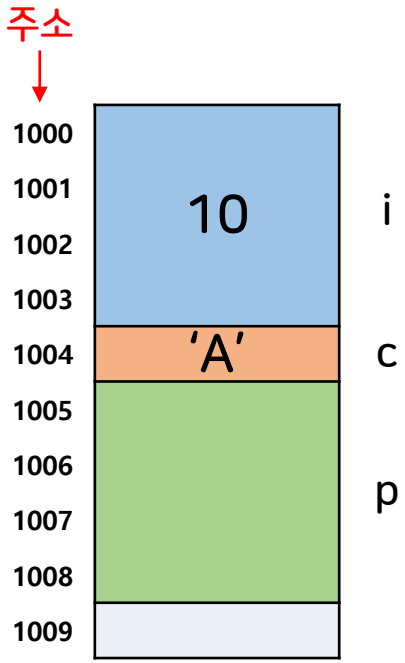
```
int main(void)
{
  int i=10;
  char c='A';
}
```



변수와 메모리

- Int (4바이트) → 정수
- char (1바이트) → 문자
- * (4바이트)

```
int main(void)
{
  int i=10;
  char c='A';
}
```

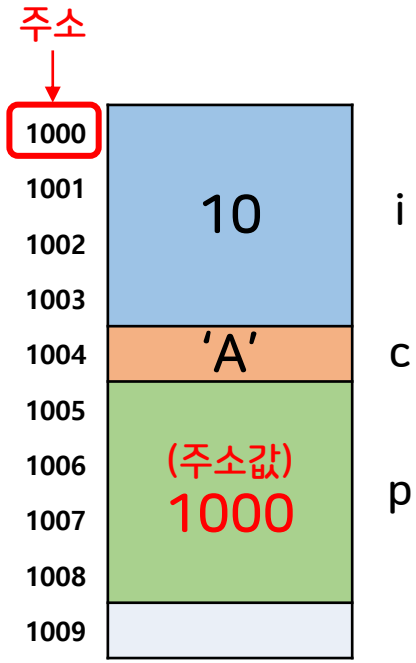



```
int main(void)
{
  * p;
}
```

변수와 메모리

- Int (4바이트) → 정수
- char (1바이트) → 문자
- * (4바이트) → 주소값

```
int main(void)
{
  int i=10;
  char c='A';
}
```

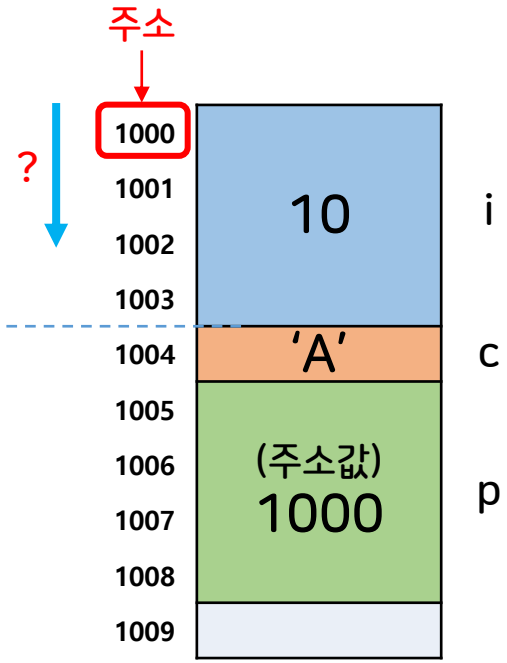



```
int main(void)
{
  * p;
  p = (주소값)1000;
}
```

변수와 메모리

- Int (4바이트) → 정수
- char (1바이트) → 문자
- * (4바이트) → 주소값

```
int main(void)
{
  int i=10;
  char c='A';
}
```

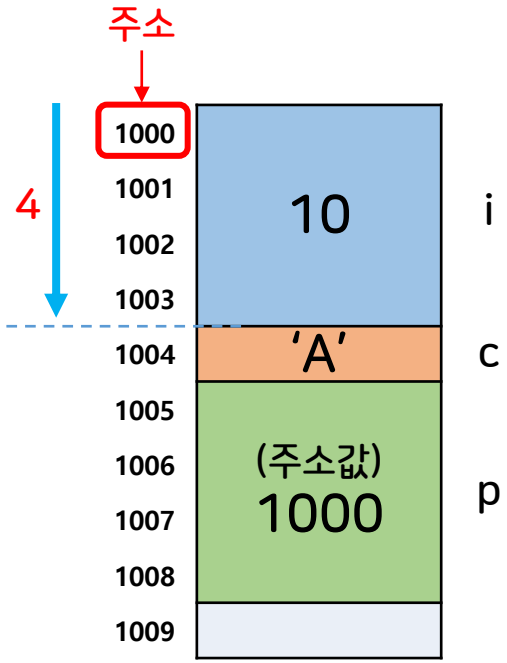



```
int main(void)
{
  * p;
  p = (주소값)1000;
}
```

변수와 메모리

- Int (4바이트) → 정수
- char (1바이트) → 문자
- * (4바이트) → 주소값

```
int main(void)
{
  int i=10;
  char c='A';
}
```



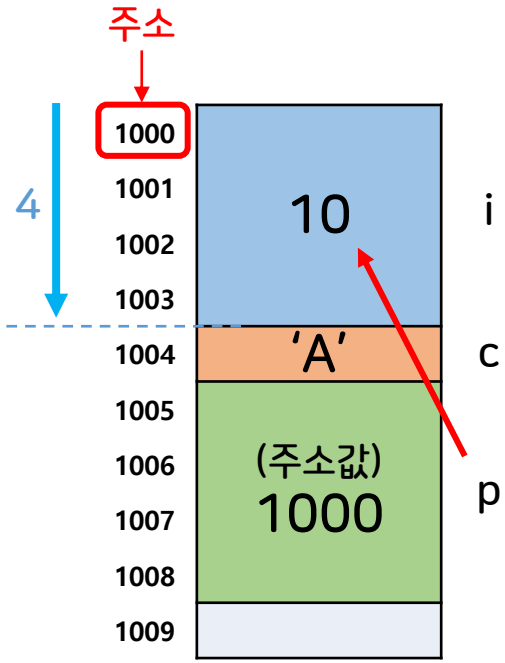
"int형 변수의 주소값을 참조하는 포인터"

```
int main(void)
{
  int* p;
  p = (주소값)1000;
}
```

변수와 메모리

- Int (4바이트) → 정수
- char (1바이트) → 문자
- * (4바이트) → 주소값

```
int main(void)
{
  int i=10;
  char c='A';
}
```

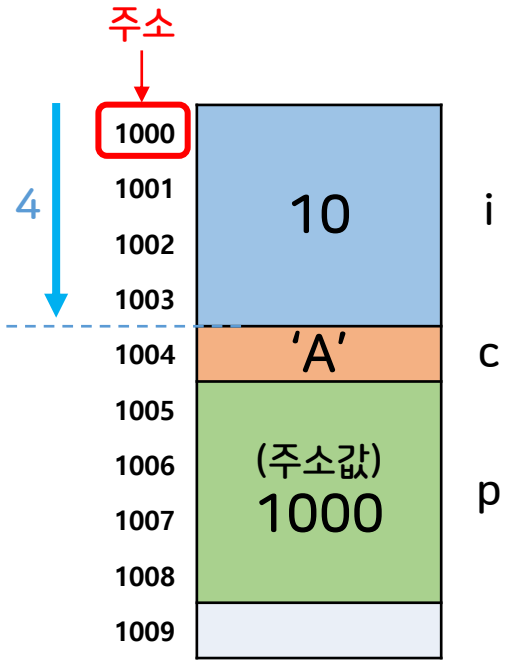



```
int main(void)
{
  int* p;
  p = (주소값)1000;
}
```

참조 연산자

- 참조 연산자 (*) 를 이용하여 포인터 변수가 가지고 있는 값(주소값)이 가르키는 메모리 주소의 값을 읽거나 수정할 수 있다.

```
int main(void)
{
  int i=10;
  char c='A';
}
```




```
int main(void)
{
  int* p;
  p = (주소값)1000;
}
```

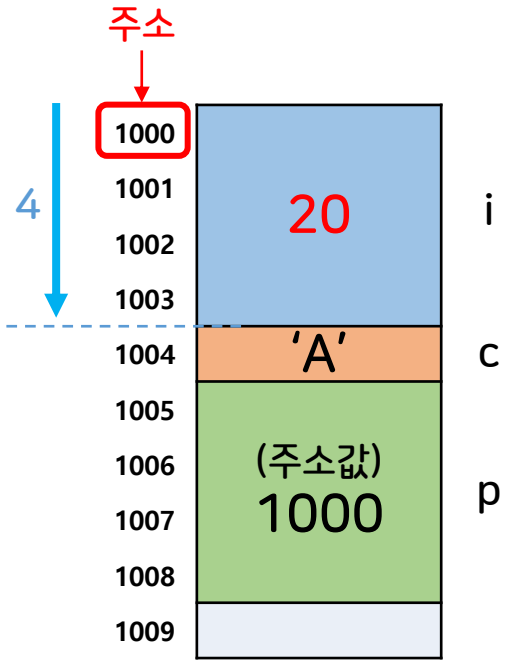
```
int main(void)
{
  printf(" *p : %d ", *p);
}
```

*p : 10

참조 연산자

- 참조 연산자 (*) 를 이용하여 포인터 변수가 가지고 있는 값(주소값)이 가르키는 메모리 주소의 값을 읽거나 수정할 수 있다.

```
int main(void)
{
  int i=10;
  char c='A';
}
```




```
int main(void)
{
  int* p;
  p = (주소값)1000;
}
```

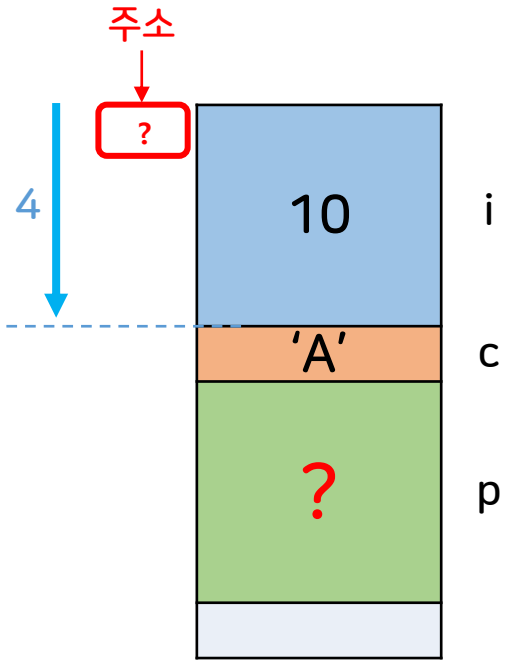
```
int main(void)
{
  *p = 20;
  printf(" i : %d ", i);
}
```

```
 i : 20
```

주소 연산자

- 주소 연산자 (&) 를 이용하여 변수의 주소값을 얻을 수 있다.

```
int main(void)
{
  int i=10;
  char c='A';
}
```

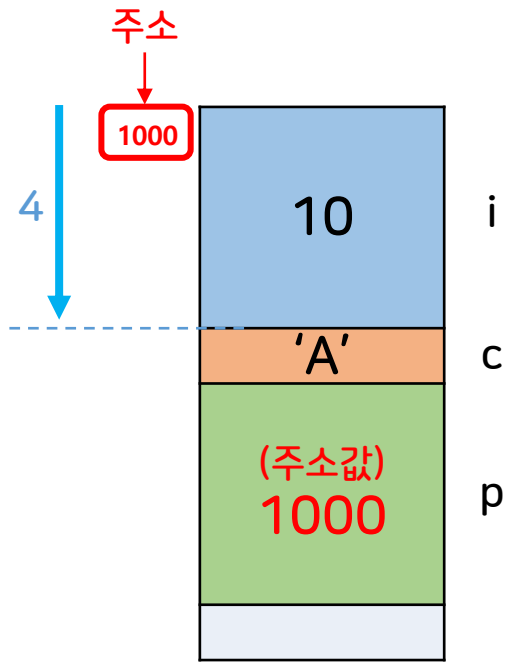



```
int main(void)
{
  int* p;
  p = ?;
}
```

주소 연산자

- 주소 연산자 (&) 를 이용하여 변수의 주소값을 얻을 수 있다.

```
int main(void)
{
  int i=10;
  char c='A';
}
```




```
int main(void)
{
  int* p;
  p = &i;
}
```

포인터의 선언

ROBIT
ROBOT SPORT GAME TEAM

```
int main(void)
{
  int*  p1;
  int  *p2;
}
```

=

```
int main(void)
{
  int  *p1, *p2;
}
```

포인터의 선언

ROBIT
ROBOT SPORT GAME TEAM

```
int main(void)
{
  int*  p1;
  int  *p2;
}
```

≠

```
int main(void)
{
  int*  p1,  p2;
}
```

포인터의 선언

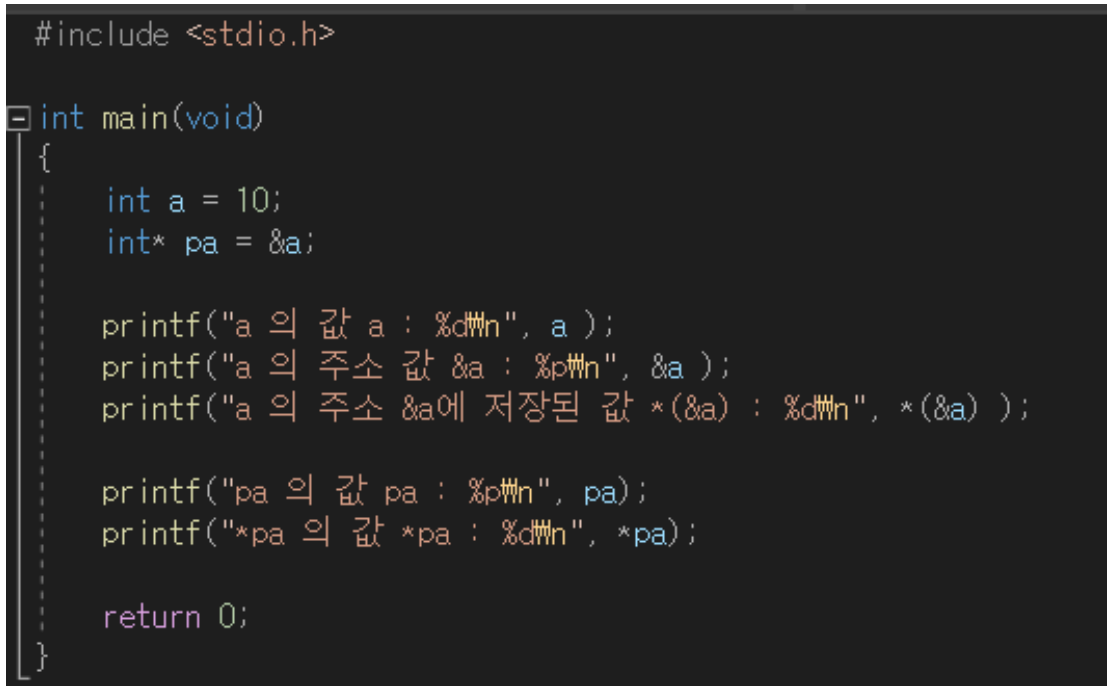
ROBIT
ROBOT SPORT GAME TEAM

```
int main(void)
{
  int*  p1;
  int  p2;
}
```

=

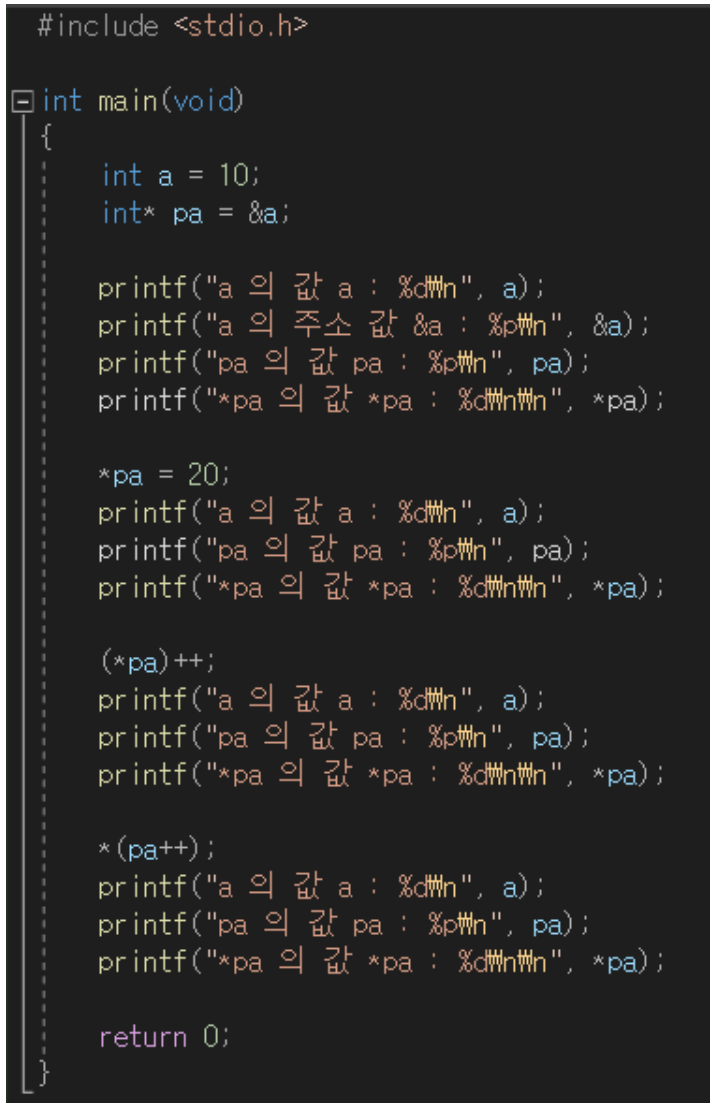
```
int main(void)
{
  int*  p1,  p2;
}
```

실습 1



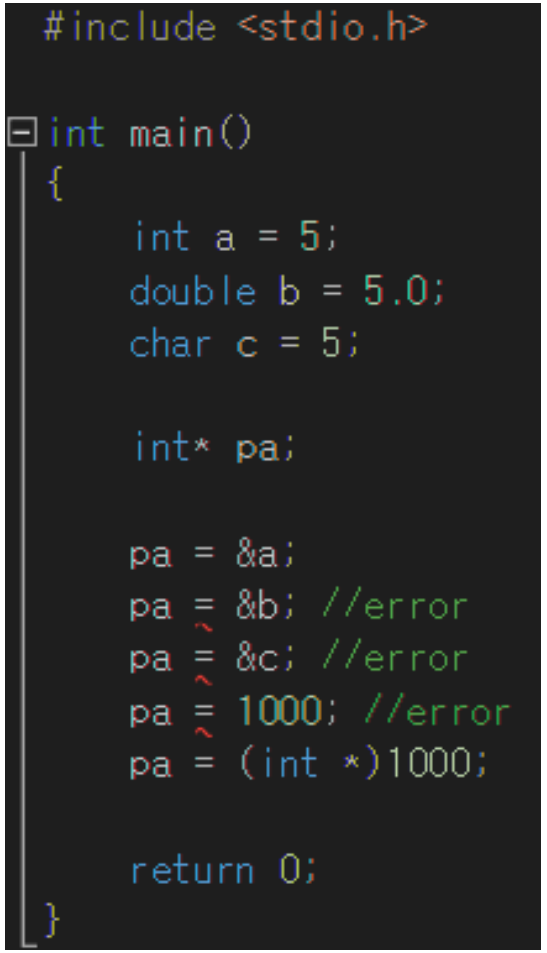
```c
#include <stdio.h>

int main(void)
{
    int a = 10;
    int* pa = &a;

    printf("a 의 값 a : %d₩n", a );
    printf("a 의 주소 값 &a : %p₩n", &a );
    printf("a 의 주소 &a에 저장된 값 *(&a) : %d₩n", *(&a) );

    printf("pa 의 값 pa : %p₩n", pa);
    printf("*pa 의 값 *pa : %d₩n", *pa);

    return 0;
}
```

실습 2



```c
#include <stdio.h>

int main(void)
{
    int a = 10;
    int* pa = &a;

    printf("a 의 값 a : %d₩n", a);
    printf("a 의 주소 값 &a : %p₩n", &a);
    printf("pa 의 값 pa : %p₩n", pa);
    printf("*pa 의 값 *pa : %d₩n₩n", *pa);

    *pa = 20;
    printf("a 의 값 a : %d₩n", a);
    printf("pa 의 값 pa : %p₩n", pa);
    printf("*pa 의 값 *pa : %d₩n₩n", *pa);

    (*pa)++;
    printf("a 의 값 a : %d₩n", a);
    printf("pa 의 값 pa : %p₩n", pa);
    printf("*pa 의 값 *pa : %d₩n₩n", *pa);

    *(pa++);
    printf("a 의 값 a : %d₩n", a);
    printf("pa 의 값 pa : %p₩n", pa);
    printf("*pa 의 값 *pa : %d₩n₩n", *pa);

    return 0;
}
```

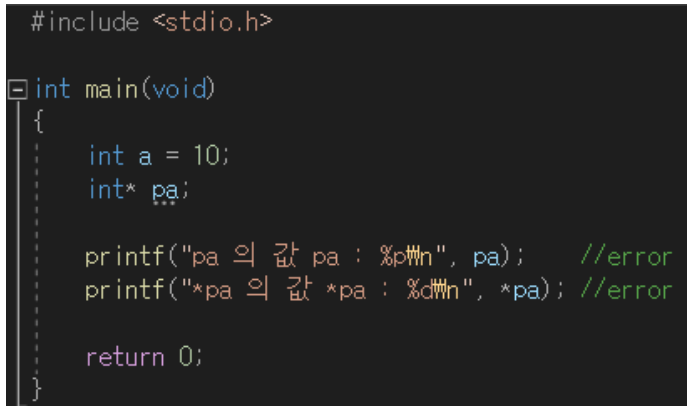
주의사항



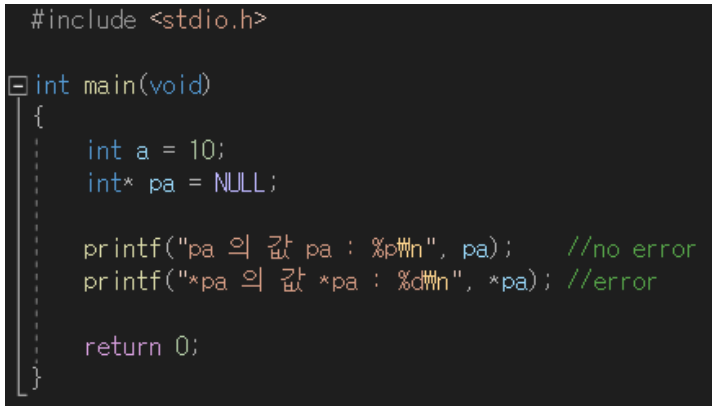
```c
#include <stdio.h>

int main()
{
    int a = 5;
    double b = 5.0;
    char c = 5;

    int* pa;

    pa = &a;
    pa = &b; //error
    pa = &c; //error
    pa = 1000; //error
    pa = (int *)1000;

    return 0;
}
```

- 포인터 변수는 자신의 자료형에 맞는 주소 값 만을 참조할 수 있다.
- '1000' 과 '(int *)1000 은 다르다.
- '1000' 은 값이다.
- '(int *)1000' 은 주소 값이다.

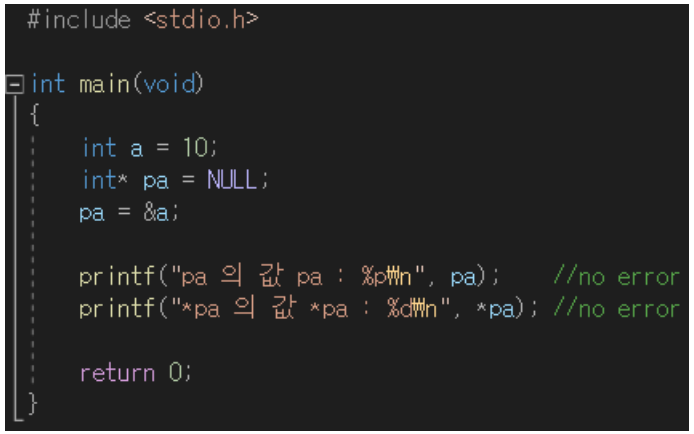
주의사항



```c
#include <stdio.h>

int main(void)
{
    int a = 10;
    int* pa;

    printf("pa 의 값 pa : %p₩n", pa);    //error
    printf("*pa 의 값 *pa : %d₩n", *pa); //error

    return 0;
}
```



```c
#include <stdio.h>

int main(void)
{
    int a = 10;
    int* pa = NULL;

    printf("pa 의 값 pa : %p₩n", pa);    //no error
    printf("*pa 의 값 *pa : %d₩n", *pa); //error

    return 0;
}
```



```c
#include <stdio.h>

int main(void)
{
    int a = 10;
    int* pa = NULL;
    pa = &a;

    printf("pa 의 값 pa : %p₩n", pa);    //no error
    printf("*pa 의 값 *pa : %d₩n", *pa); //no error

    return 0;
}
```

- 초기화 되지 않은 포인터 변수에 접근 할 수 없다.

- 선언시에 변수의 주소를 참조하지 않는다면 적어도 NULL 로 초기화 시켜주자.
  (NULL) = (주소 값) 0
  = 아무것도 가리키지 않은 상태

주의사항

```c
#include<stdio.h>


int main()
{
    int b = 5;
    int* pb = NULL;

    printf("b 의 값: b = %d₩n", b);
    printf("b 의 주소 값: &b = %d₩n", &b);

    pb = &b;

    if (pb != NULL)
    {
        printf("pb 의 값: pb = %d₩n", pb);
        printf("*pb 의 값: *pb = %d₩n", *pb);
    }

    return 0;
}
```

- NULL 로 초기화 된 포인터 변수에 접근 시, 해당 포인터 변수가 NULL 값인지 아닌지 확인 후 접근하자.

Why Pointer?

ROBIT
ROBOT SPORT GAME TEAM

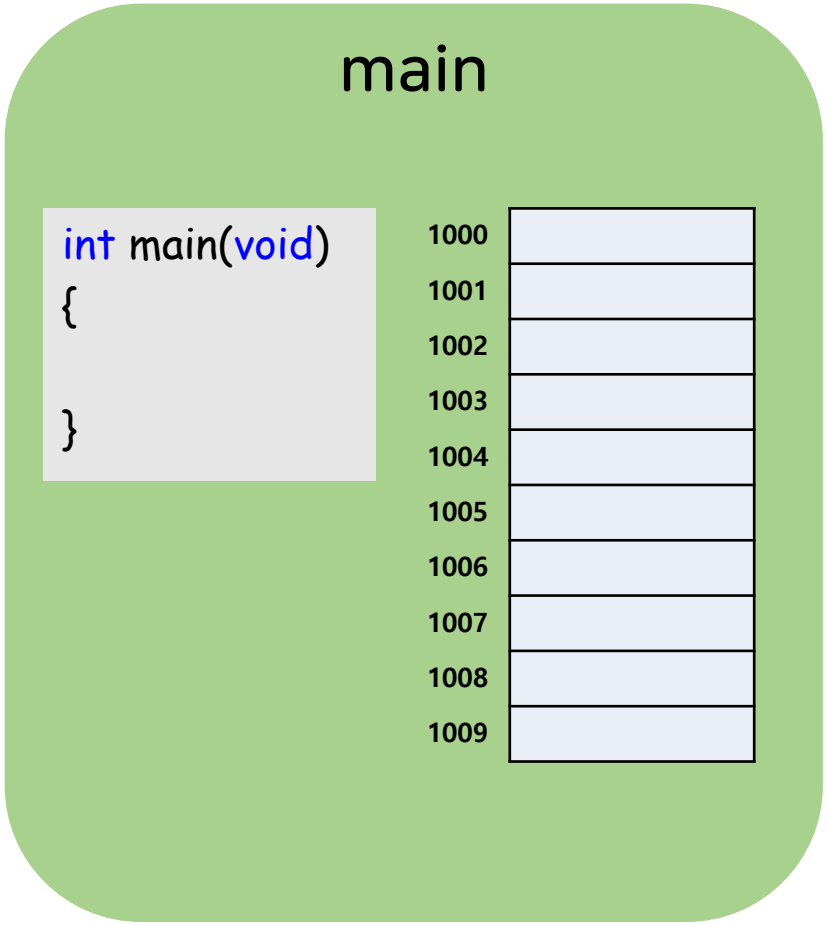
- 데이터 공유, 함수 내부에서 함수 외부의 값을 가져오거나 수정할 수 있다.
- 함수에서 두개 이상의 값을 반환해야 할 때.
- 큰 데이터 구조를 간단한 방법으로 참조할 수 있다.

Why Pointer?

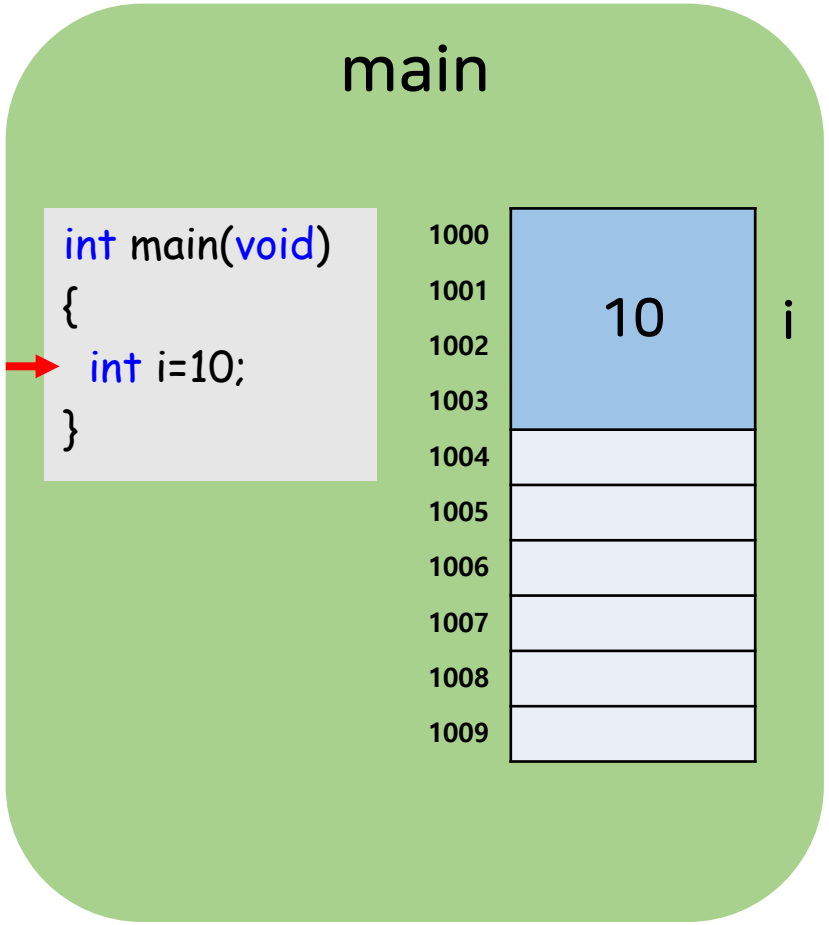
ROBIT
ROBOT SPORT GAME TEAM

- 데이터 공유, 함수 내부에서 함수 외부의 값을 가져오거나 수정할 수 있다.
- 함수에서 두개 이상의 값을 반환해야 할 때.
- 큰 데이터 구조를 간단한 방법으로 참조할 수 있다.

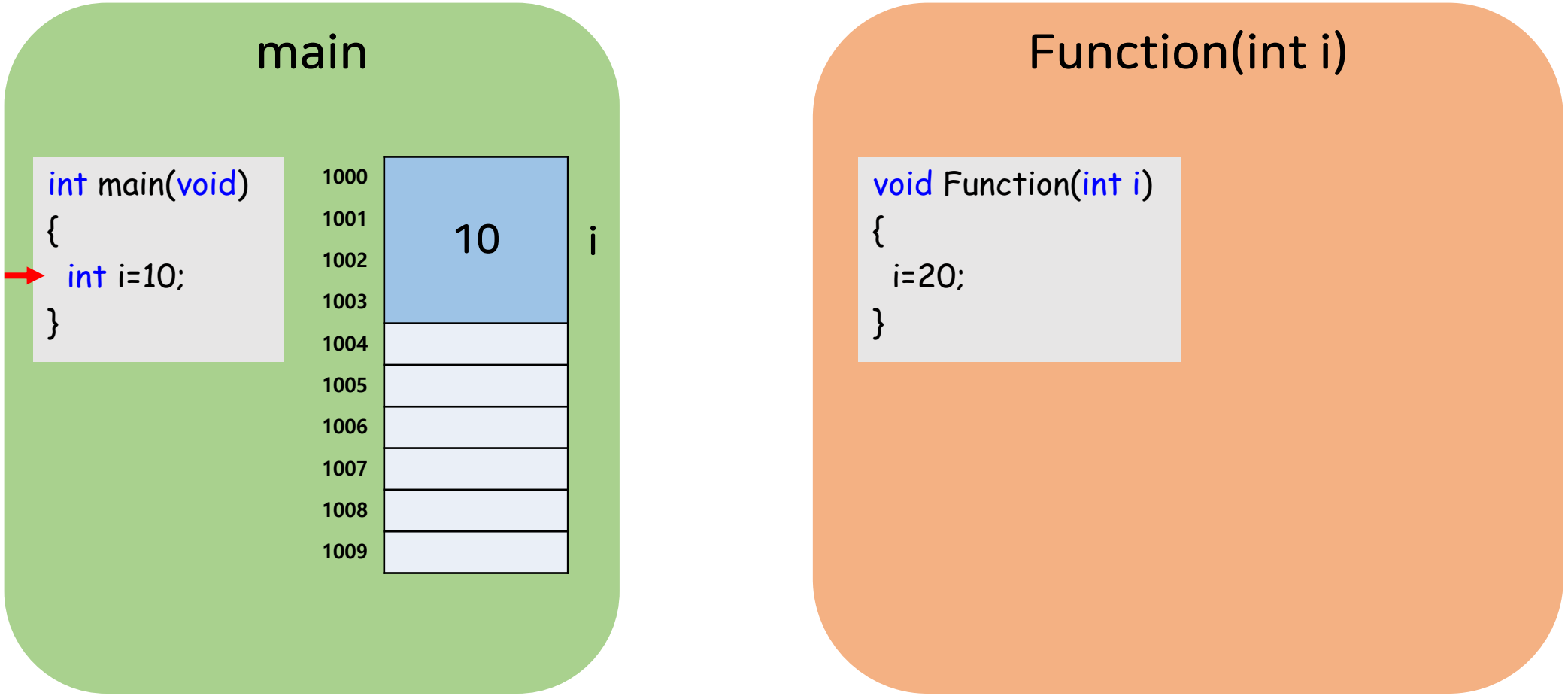
함수와 포인터 -값에 의한 호출



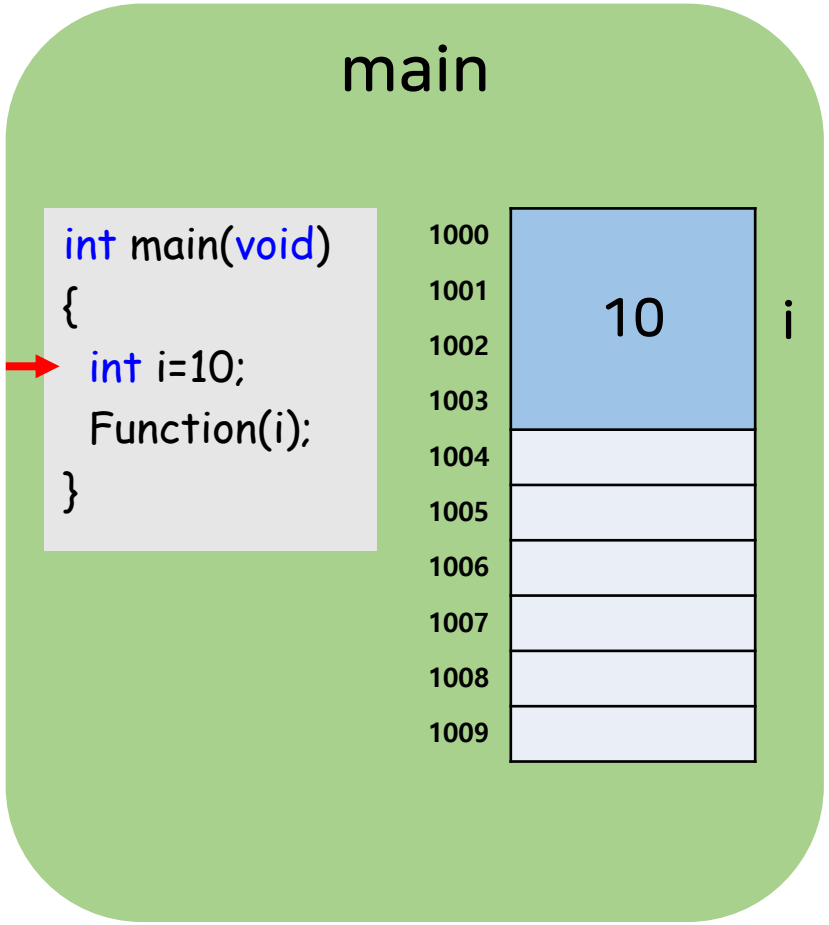
함수와 포인터 -값에 의한 호출

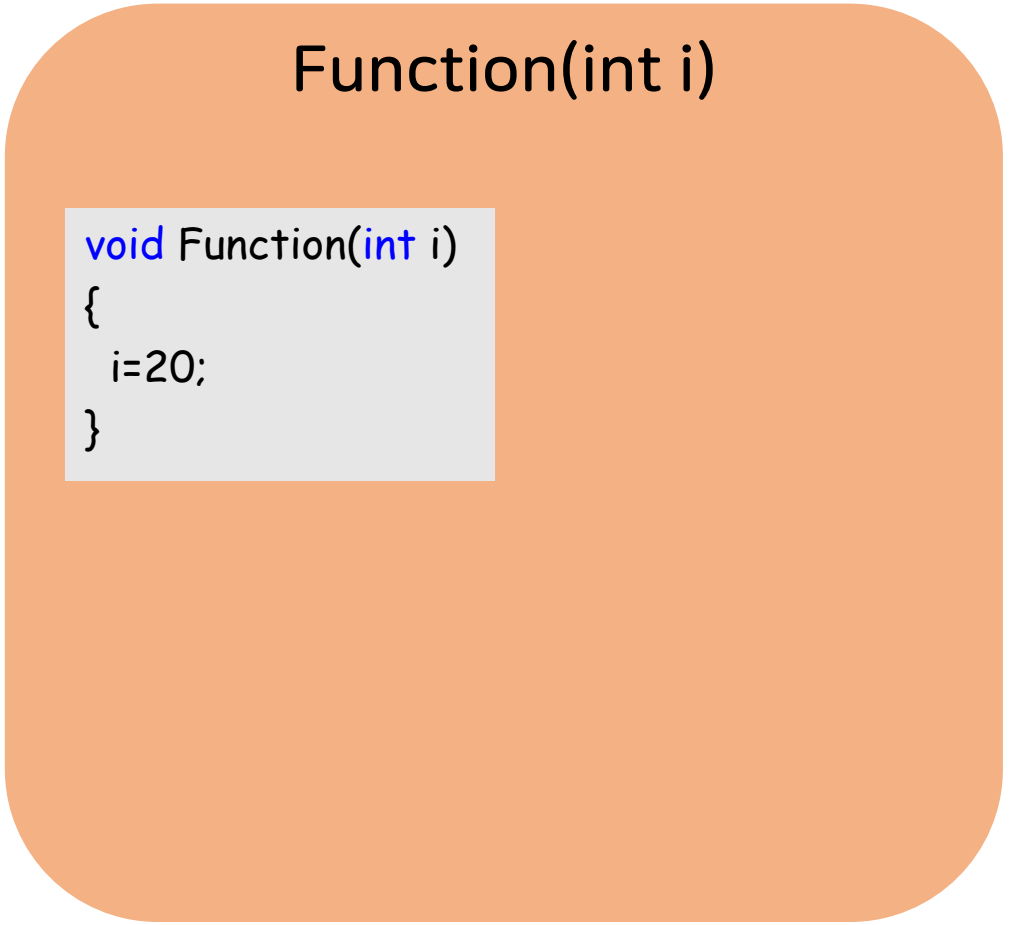


함수와 포인터 -값에 의한 호출

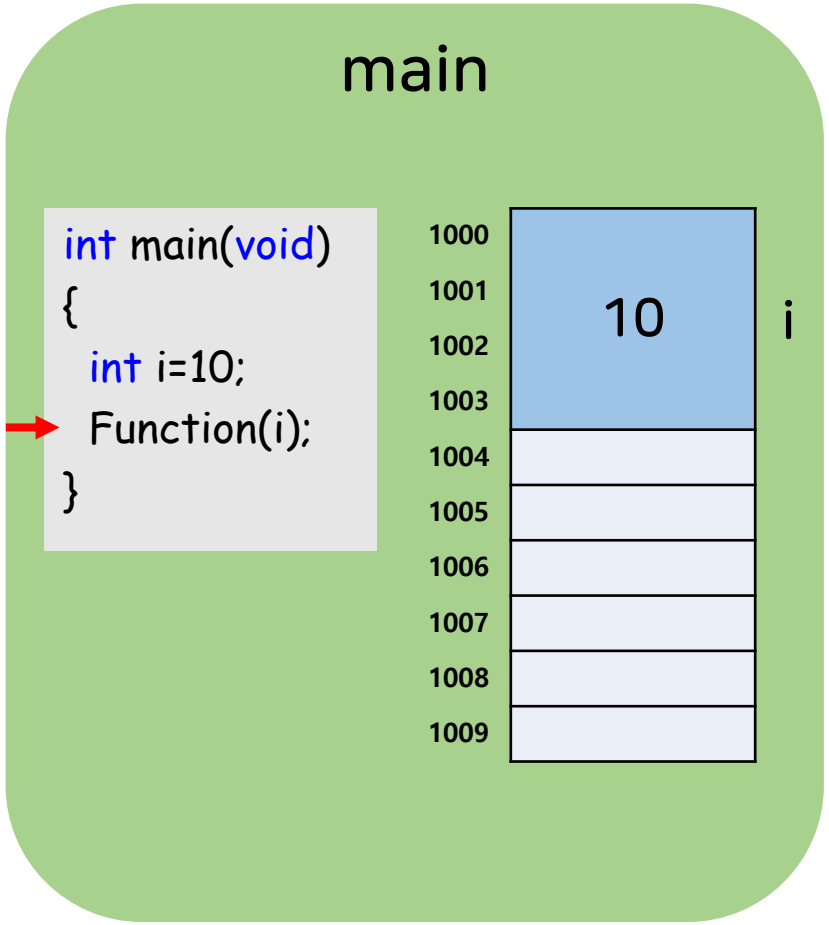


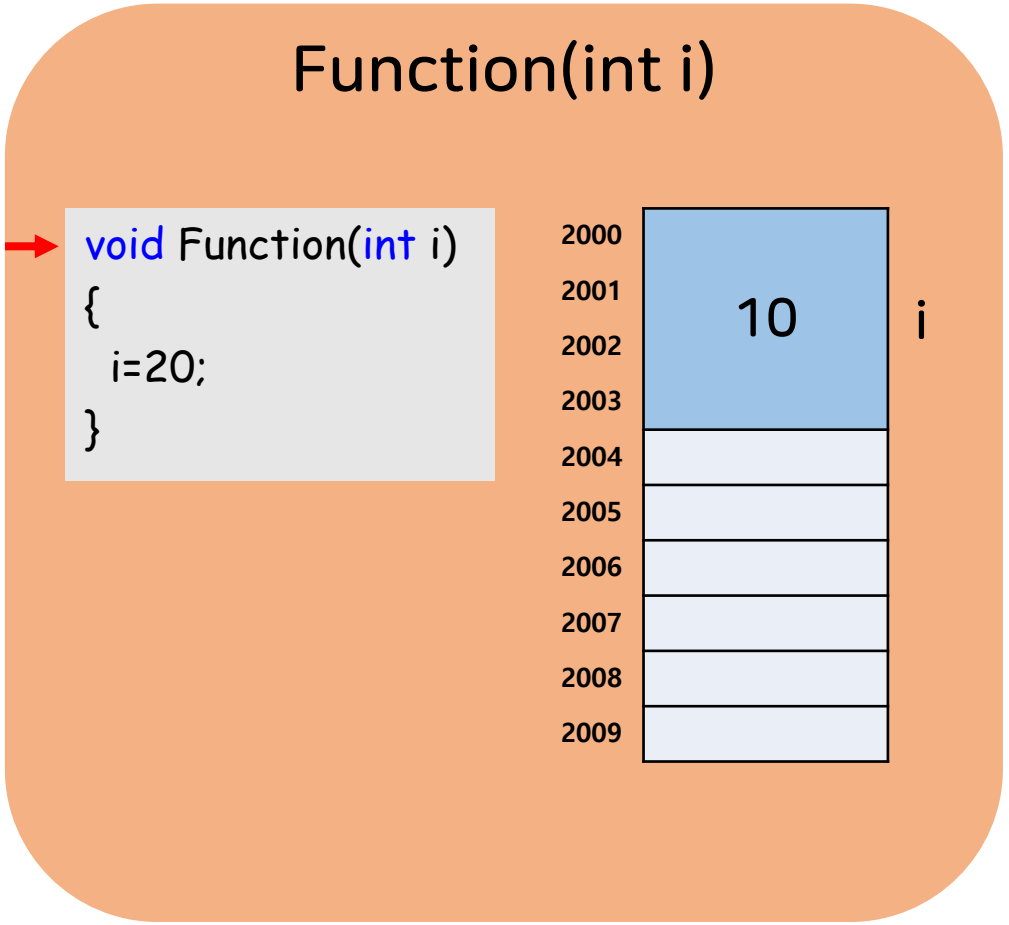
함수와 포인터 -값에 의한 호출



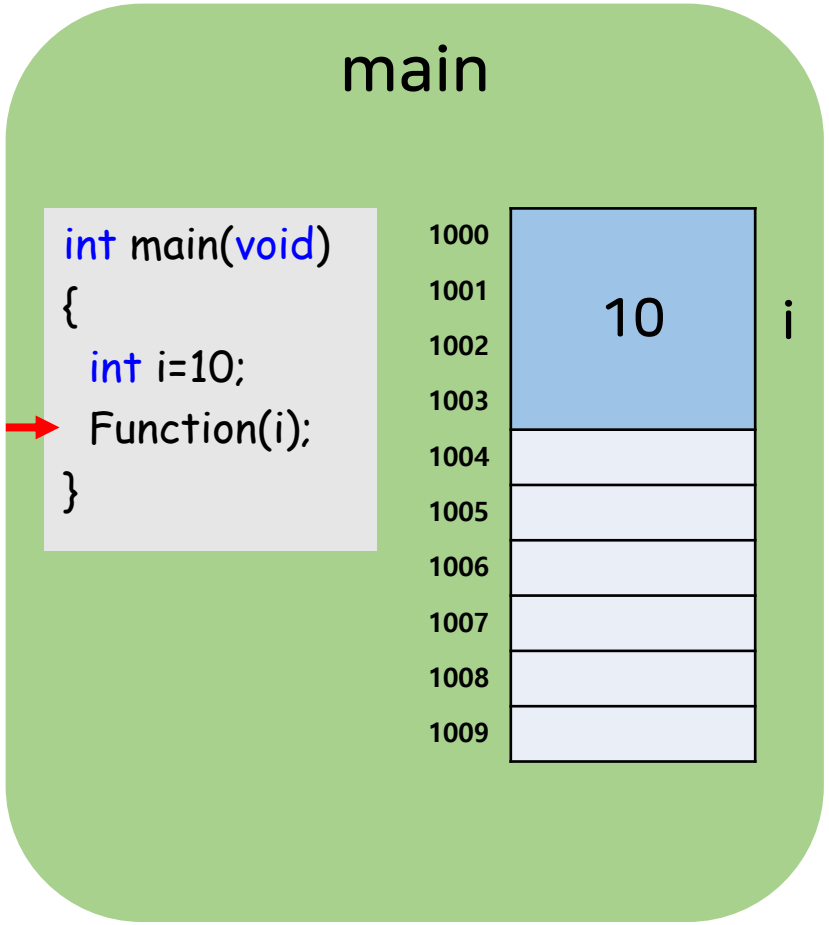


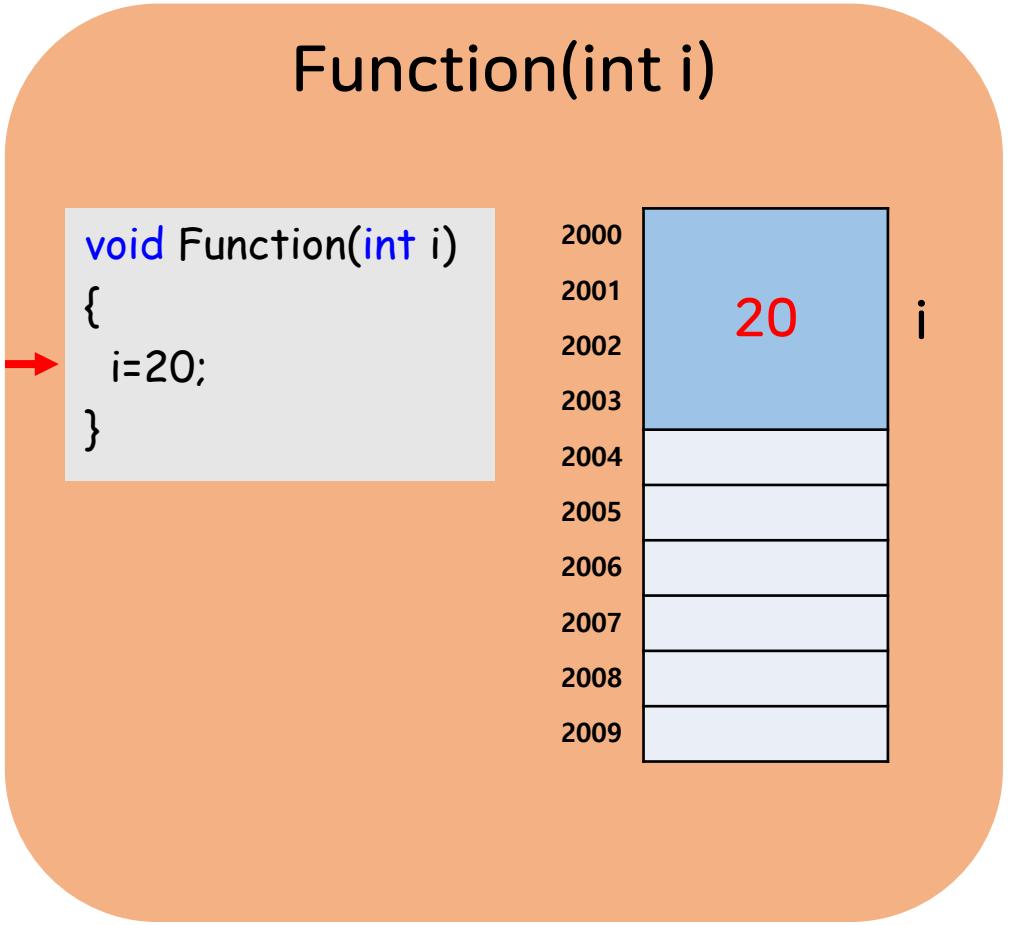
함수와 포인터 -값에 의한 호출



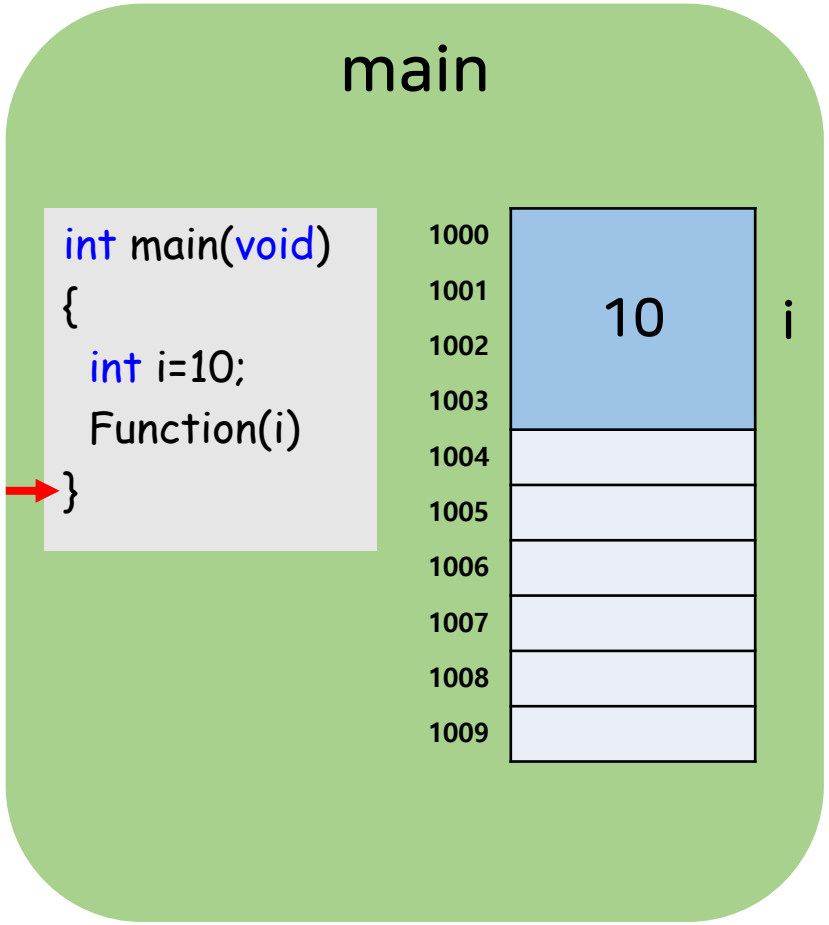


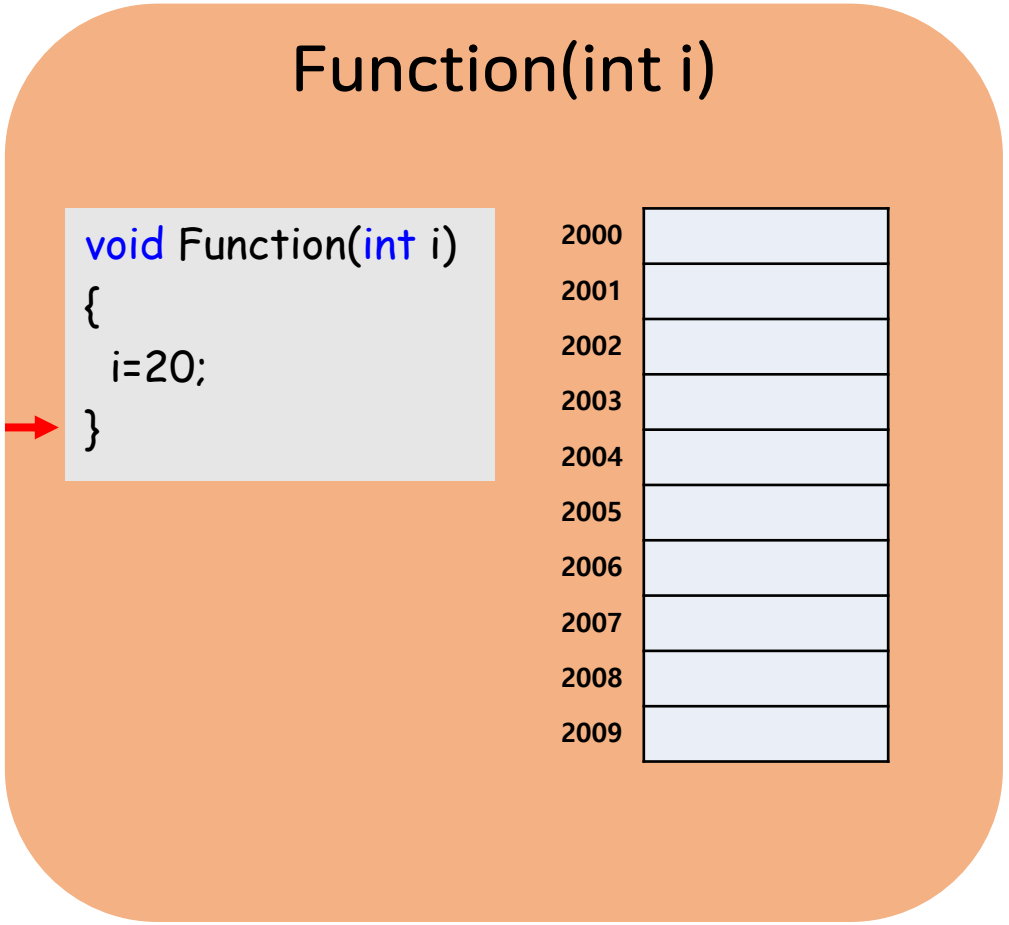
함수와 포인터 -값에 의한 호출



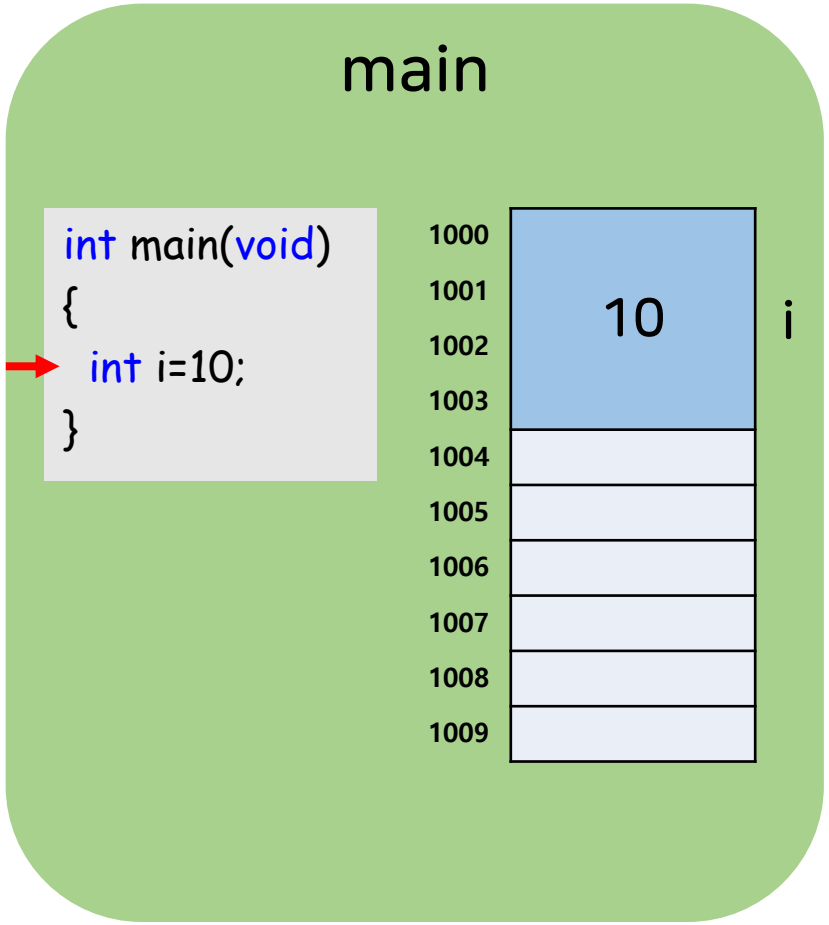


함수와 포인터 -값에 의한 호출

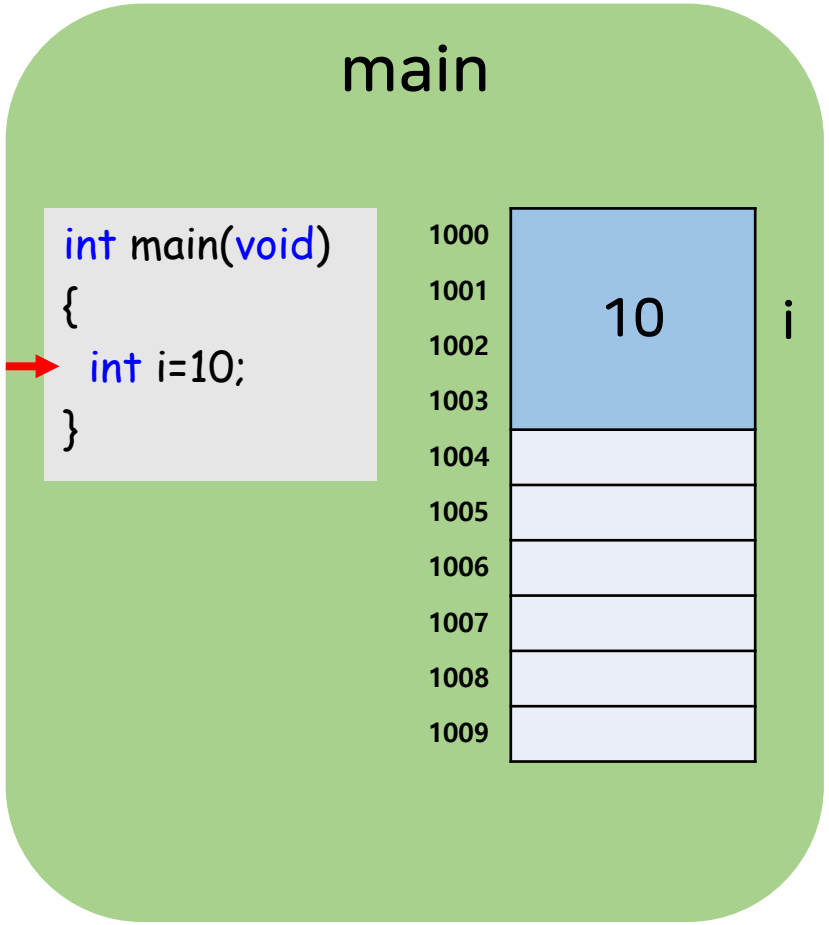


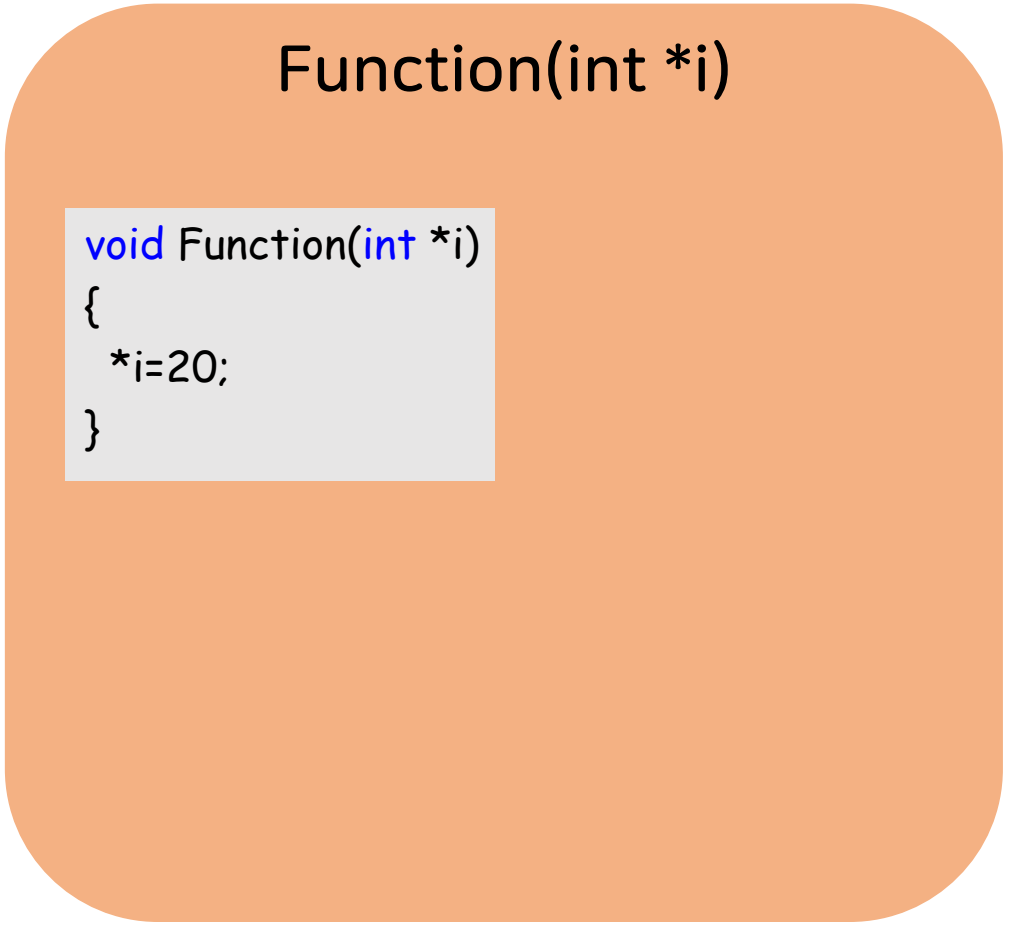


함수와 포인터 -참조에 의한 호출

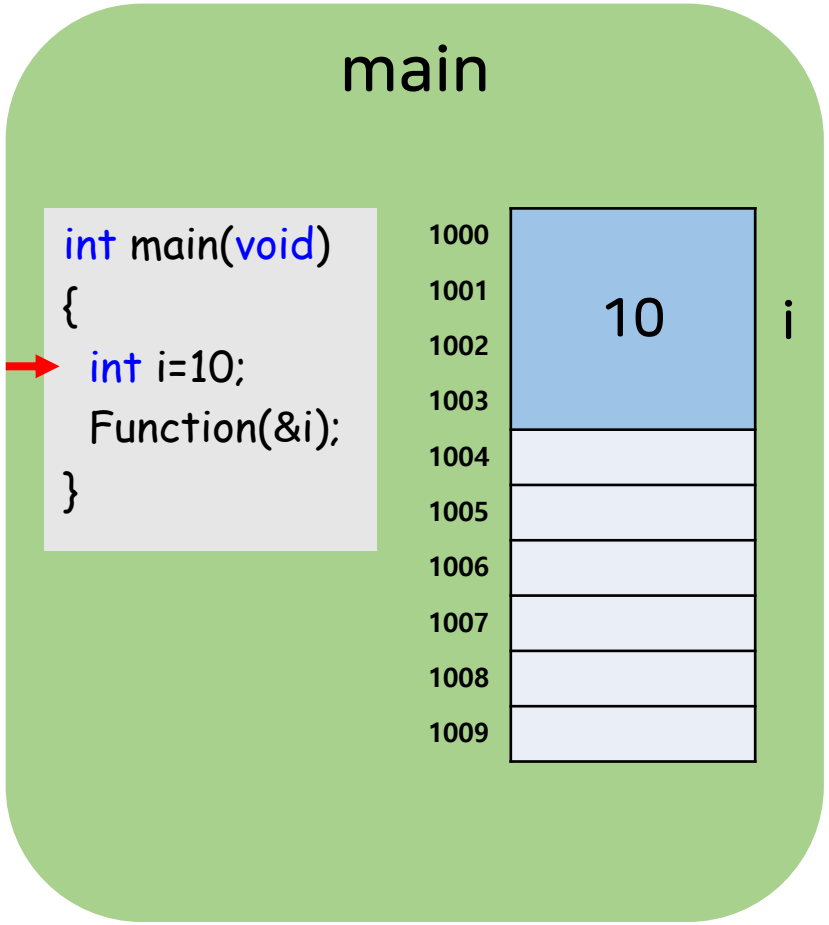


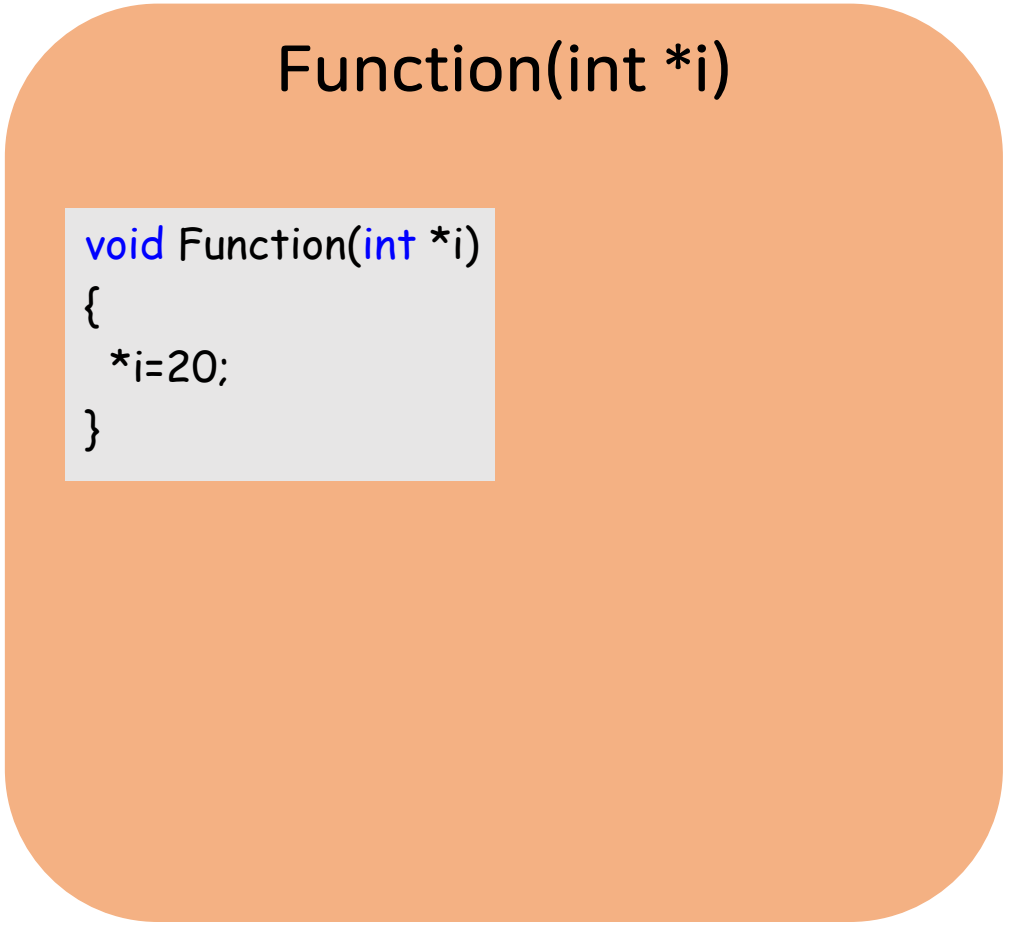
함수와 포인터 -참조에 의한 호출



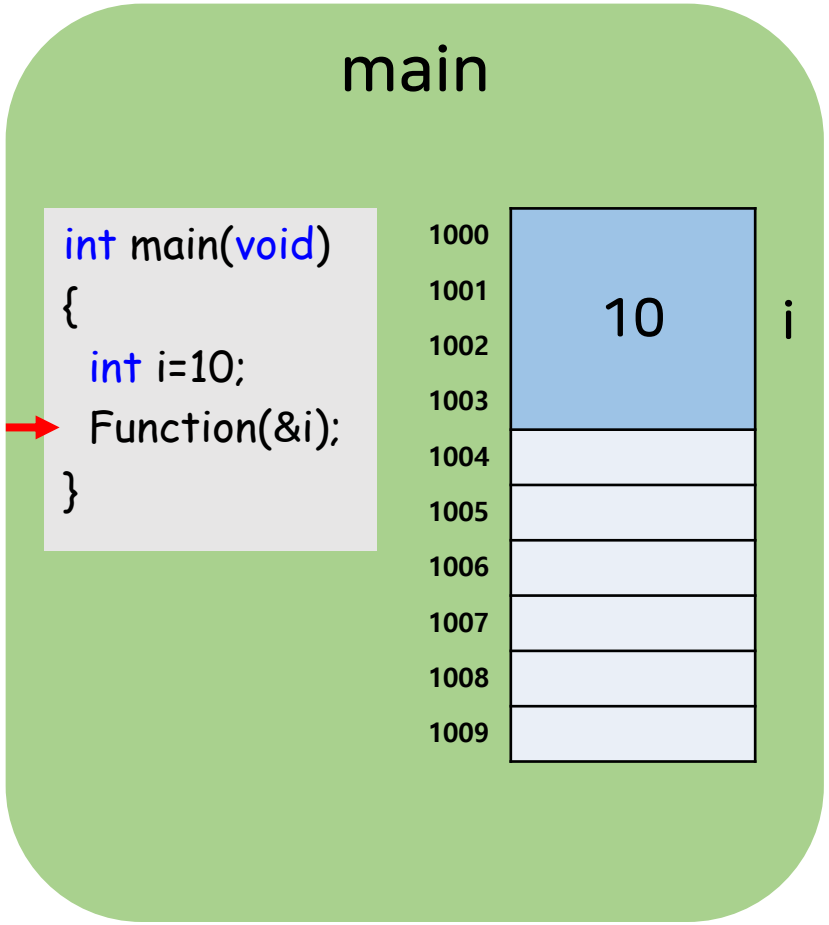


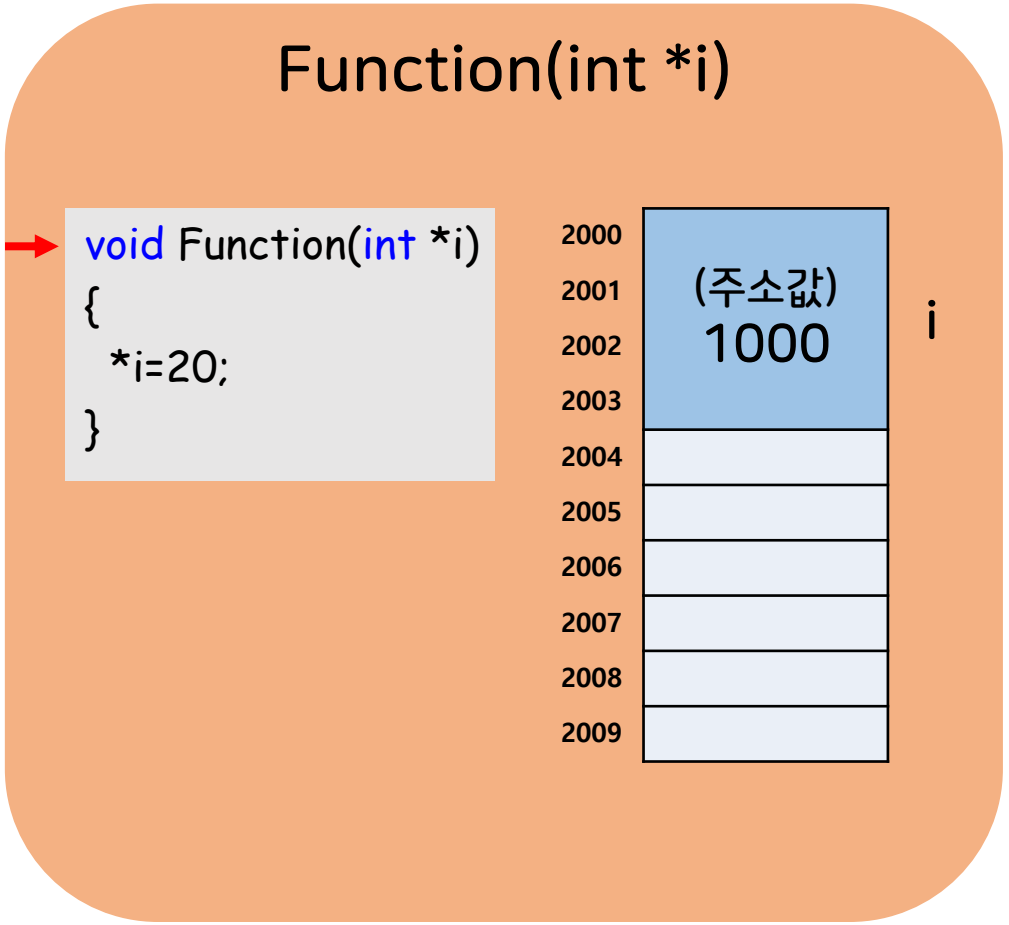
함수와 포인터 -참조에 의한 호출



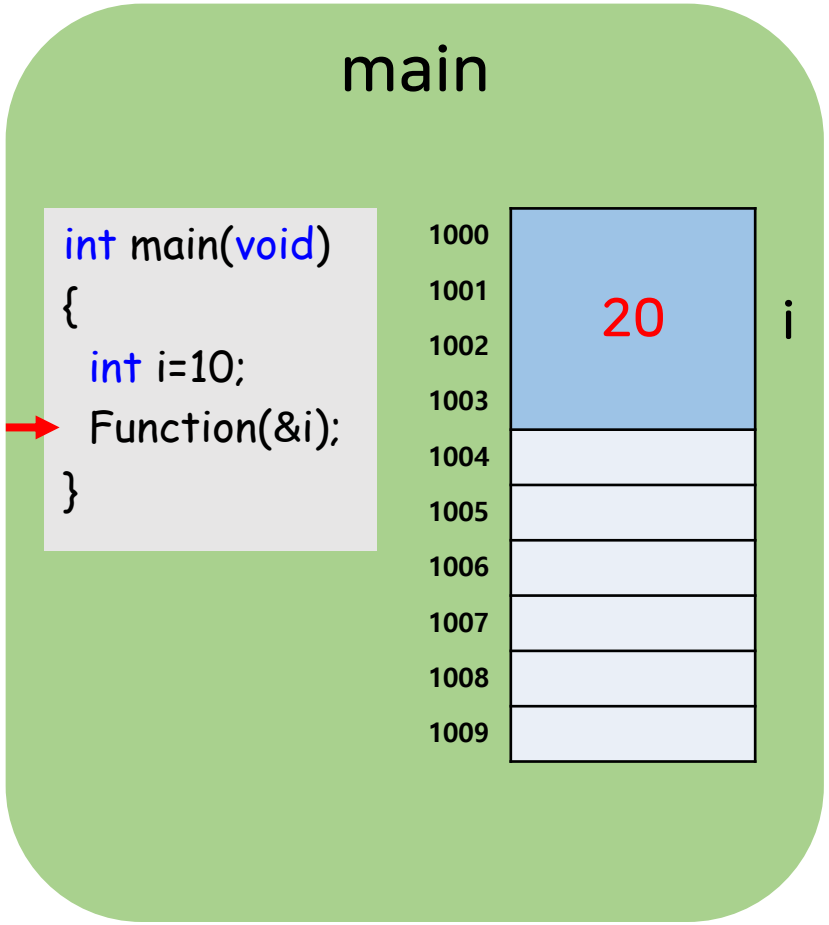


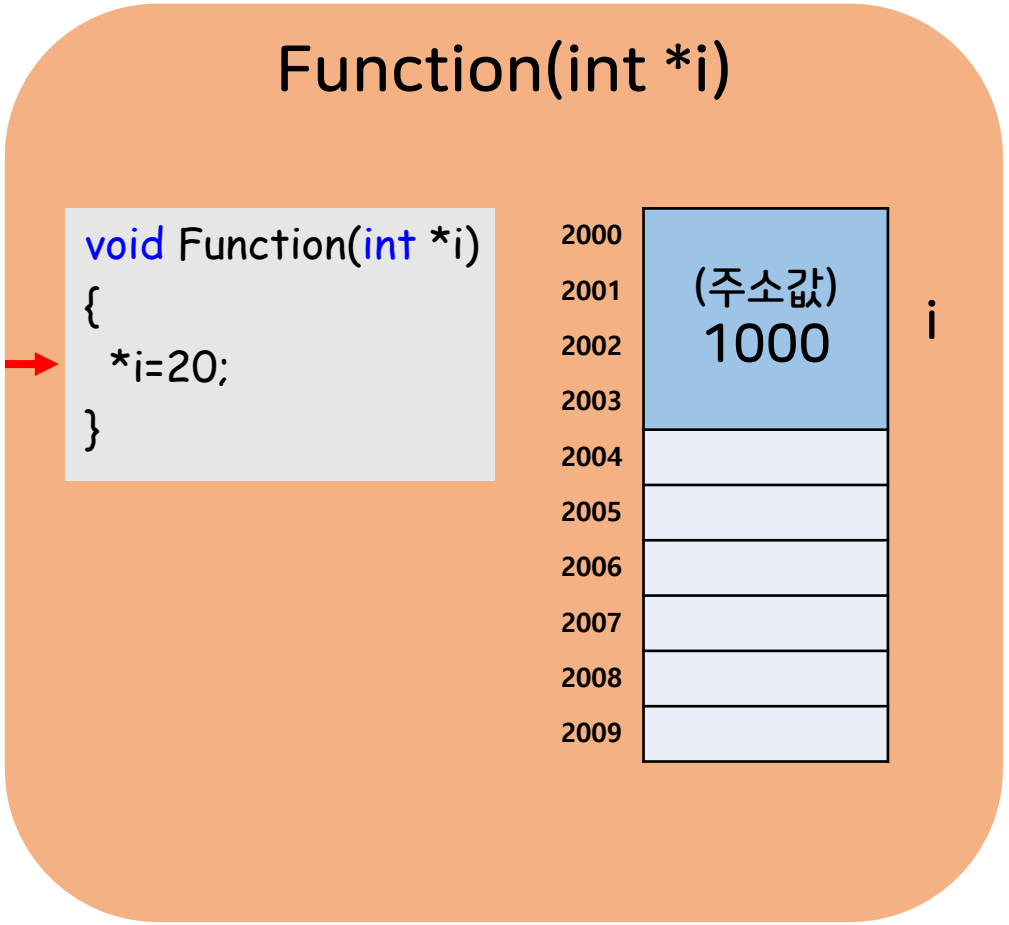
함수와 포인터 -참조에 의한 호출



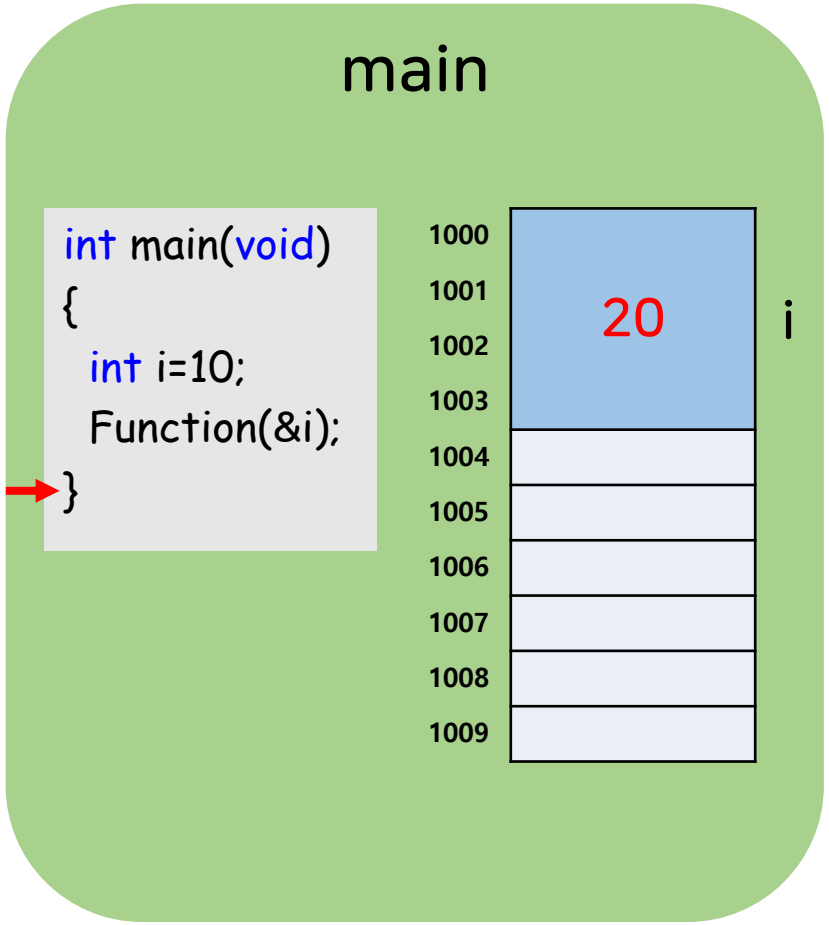


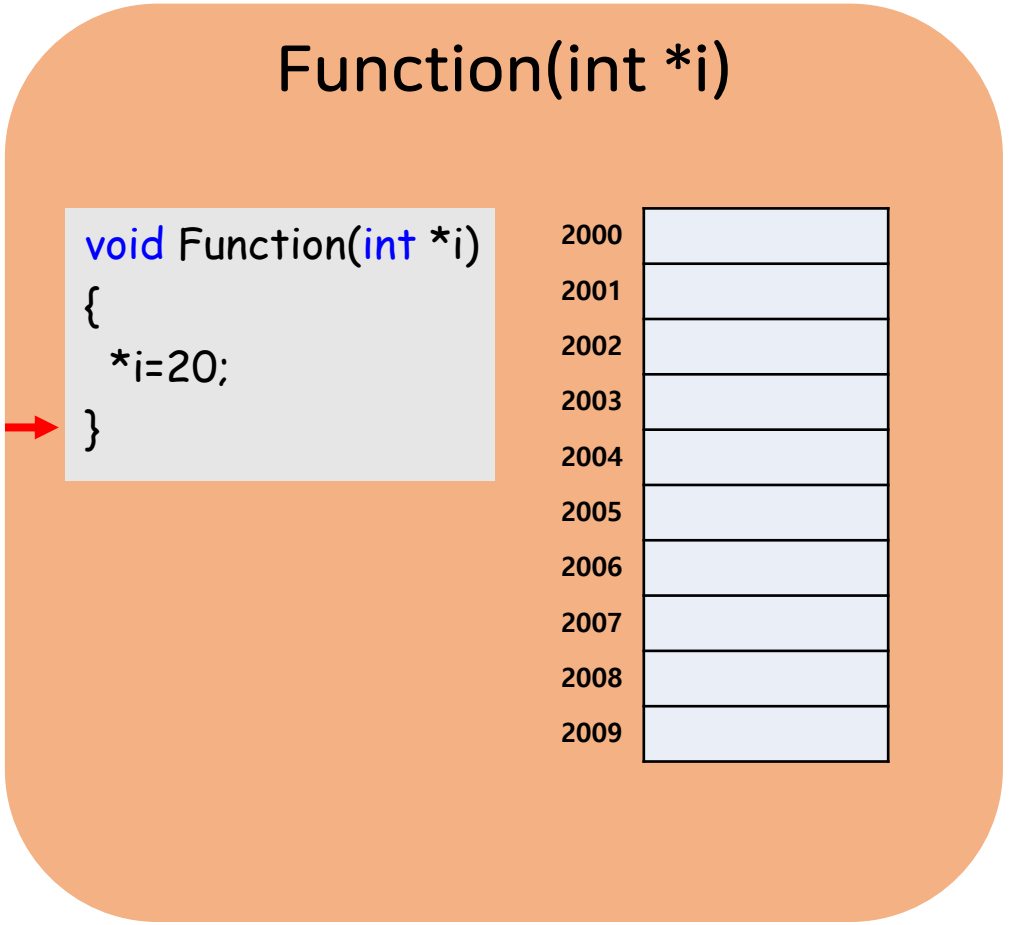
함수와 포인터 -참조에 의한 호출





함수와 포인터 -참조에 의한 호출





Why Pointer?

ROBIT
ROBOT SPORT GAME TEAM

- 데이터 공유, 함수 내부에서 함수 외부의 값을 가져오거나 수정할 수 있다.
- 함수에서 두개 이상의 값을 반환해야 할 때.
- 큰 데이터 구조를 간단한 방법으로 참조할 수 있다.

함수의 반환

- 기본적으로 함수는 하나의 return 값 만을 반환할 수 있다.

```
int Function(int i)
{
  i=20;
  return i;
}
```

```
double Function()
{
  double d=20.0;
  return d;
}
```

함수의 반환

- 하지만 함수의 매개변수의 개수에는 제한이 없기 때문에 함수의 매개변수로 변수의 주소 값을 전달하여 자료형과 개수 상관없이 여러 개의 return을 하는 효과를 가져올 수 있다.

```
int main(void)
{
  int i=10;
  double d=20.0;
  Function(&i, &d);
}
```

```
void Function(int *i, double *d)
{
  *i = 20;
  *d = 30.0;
}
```

```
i : 20, d : 30.000000
```

실습 3

ROBIT
ROBOT SPORT GAME TEAM

값에 의한 호출 VS 참조에 의한 호출
call by value VS call by reference

```c
#include <stdio.h>

void swap_CallByValue(int x, int y);
void swap_CallByReference(int *px, int *py);

int main(void)
{
    int x = 10, y = 20;

    printf("[main] x = %d, y = %d ₩n", x, y);
    swap_CallByValue(x, y);
    printf("[main] x = %d, y = %d ₩n", x, y);
    swap_CallByReference(&x, &y);
    printf("[main] x = %d, y = %d ₩n", x, y);

    return 0;
}
```

```c
void swap_CallByValue(int x, int y)
{
    int temp;
    printf("[Function-V] x = %d, y = %d ₩n", x, y);
    temp = x;
    x = y;
    y = temp;
    printf("[Function-V] x = %d, y = %d ₩n", x, y);
}

void swap_CallByReference(int* px, int* py)
{
    int temp;
    printf("[Function-R] x = %d, y = %d ₩n", *px, *py);
    temp = *px;
    *px = *py;
    *py = temp;
    printf("[Function-R] x = %d, y = %d ₩n", *px, *py);
}
```

```
[main] x = 10, y = 20
[Function-V] x = 10, y = 20
[Function-V] x = 20, y = 10
[main] x = 10, y = 20
[Function-R] x = 10, y = 20
[Function-R] x = 20, y = 10
[main] x = 20, y = 10
```

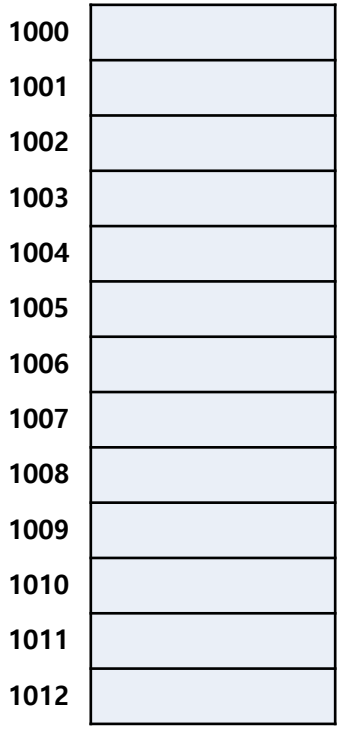
Why Pointer?

ROBIT
ROBOT SPORT GAME TEAM

- 데이터 공유, 함수 내부에서 함수 외부의 값을 가져오거나 수정할 수 있다.
- 함수에서 두개 이상의 값을 반환해야 할 때.
- 큰 데이터 구조를 간단한 방법으로 참조할 수 있다.

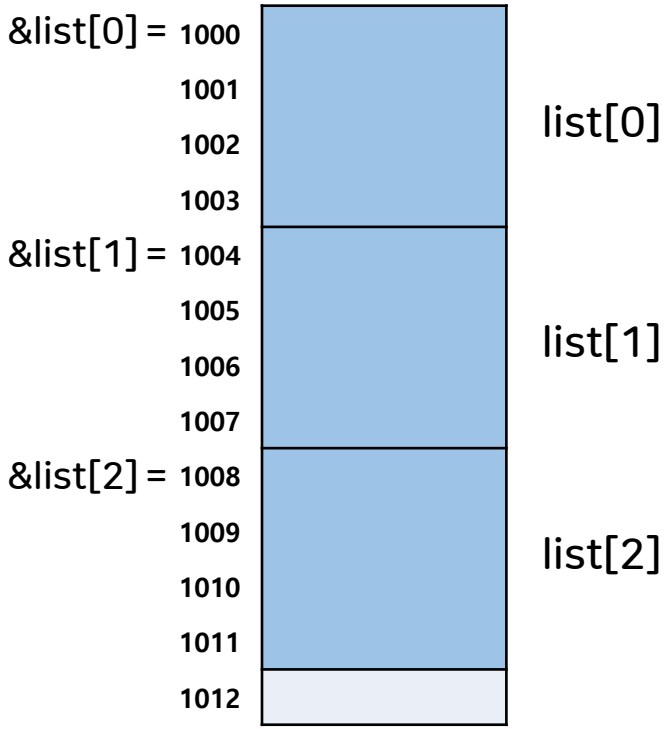
배열과 포인터

```
int main(void)
{
  int i[3];
}
```



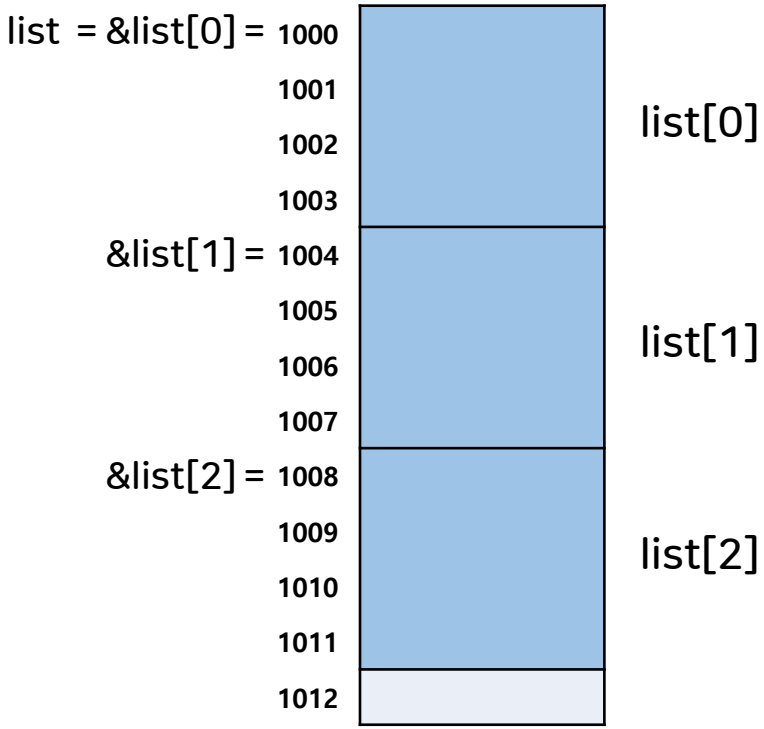
배열과 포인터

```
int main(void)
{
  int list[3];
}
```



배열과 포인터

```
int main(void)
{
  int list[3];
}
```

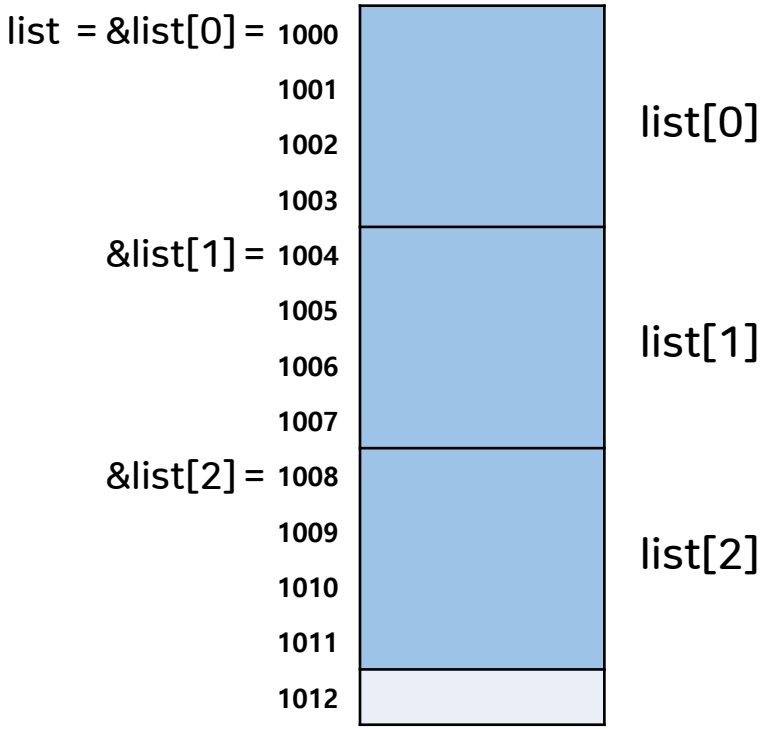


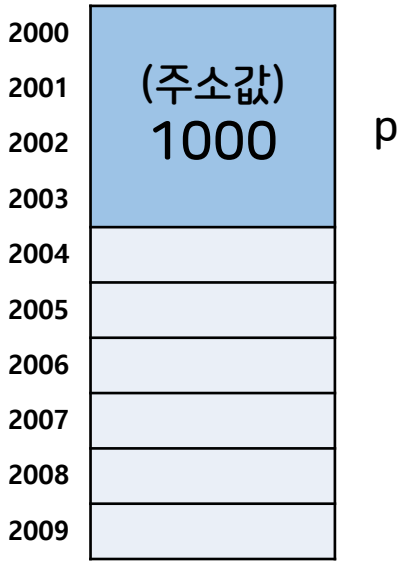

```c
#include <stdio.h>

int main(void)
{
    int list[3];
    printf("list : %p\n", list);
    printf("&list : %p\n", &list);
    printf("&list[0] : %p\n", &list[0]);

    return 0;
}
```

```
list : 000000F42895FCD8
&list : 000000F42895FCD8
&list[0] : 000000F42895FCD8
```

배열과 포인터

```
int main(void)
{
  int list[3];
  int *p = list;
}
```





배열과 포인터

p는 배열 list에서 첫 요소의 포인터、 k = 정수 일 때

p + k = &list[k]

배열과 포인터

ROBIT
ROBOT SPORT GAME TEAM

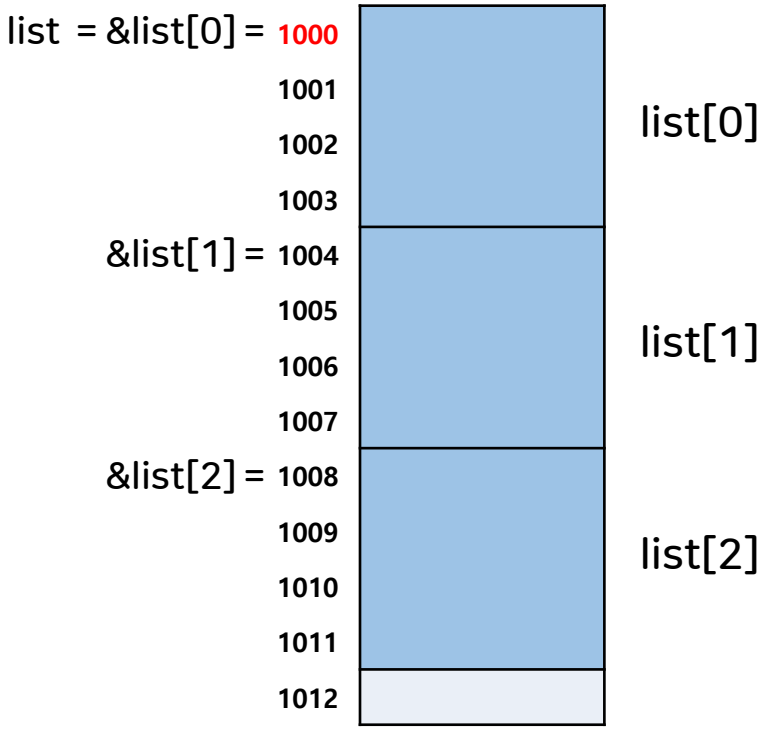
p는 배열 list에서 첫 요소의 포인터、 k = 정수 일 때

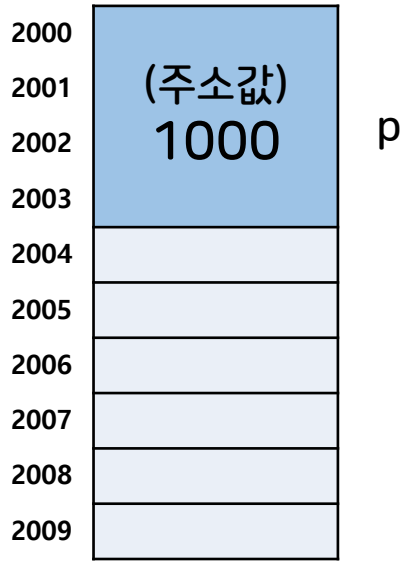
$$p + k = \&list[k]$$

포인터가 배열의 시작 주소를 가리킬 때
이 값에 정수 k를 더한 결과는
배열의 k번째 요소의 주소가 된다.

배열과 포인터

```
int main(void)
{
  int list[3];
  int *p = list;
}
```

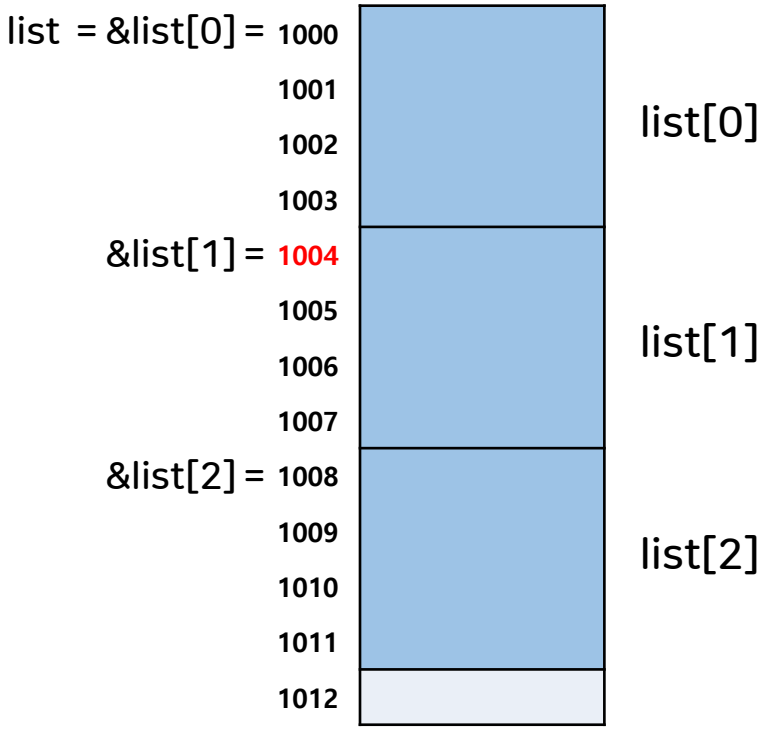


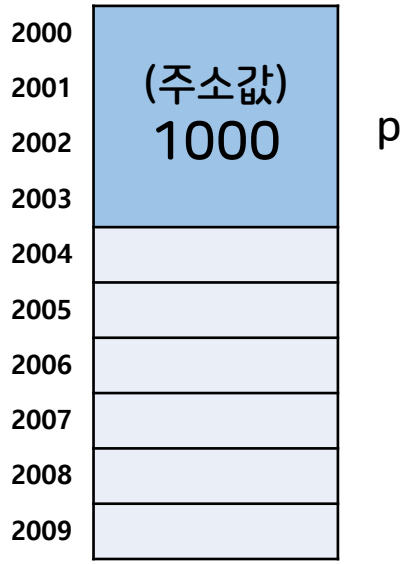


p = 1000
(p+1) =?

배열과 포인터

```
int main(void)
{
  int list[3];
  int *p = list;
}
```

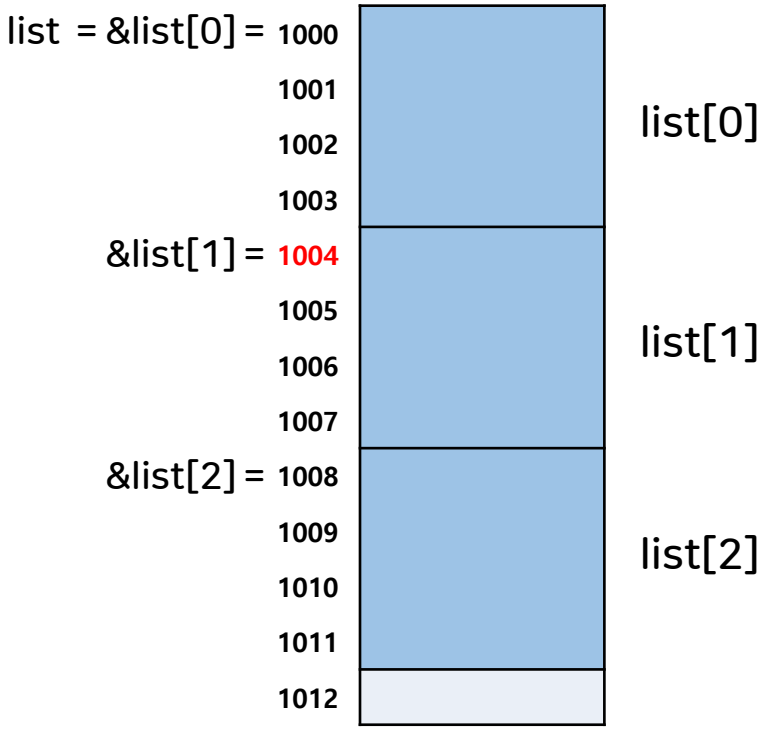


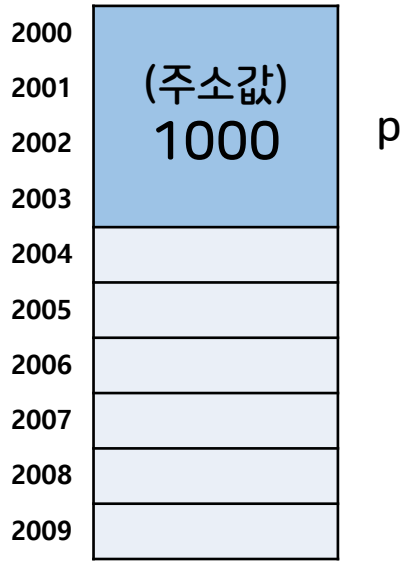


p = 1000
(p+1) =1004

배열과 포인터

```
int main(void)
{
  int list[3];
  int *p = list;
}
```

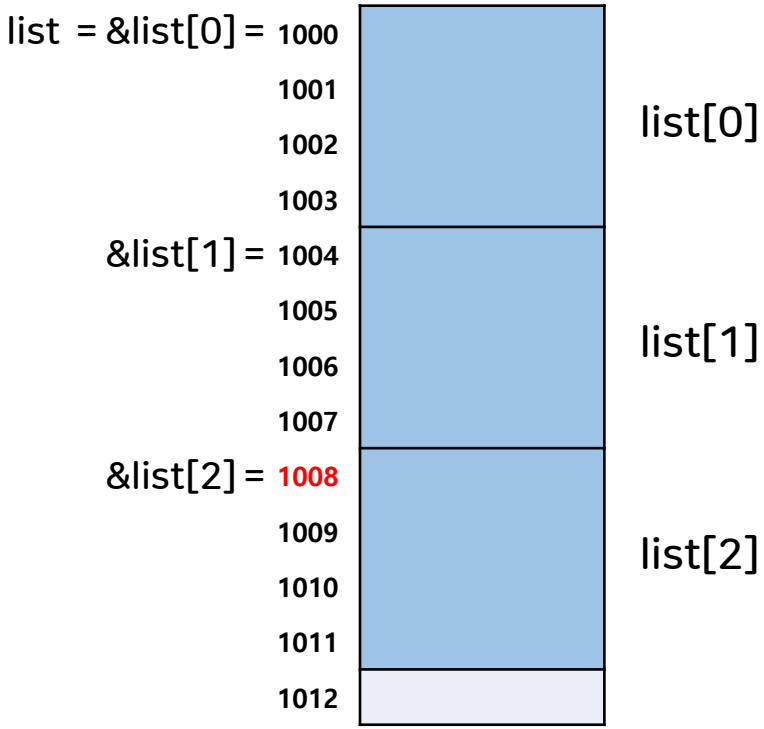


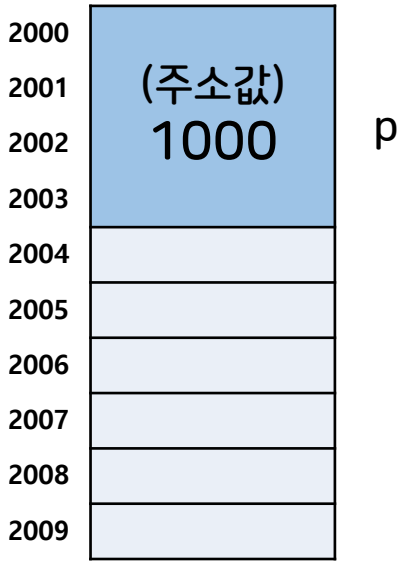


p = 1000
(p+1) =1004
(p+2) =1008

배열과 포인터

```
int main(void)
{
  int list[3];
  int *p = list;
}
```



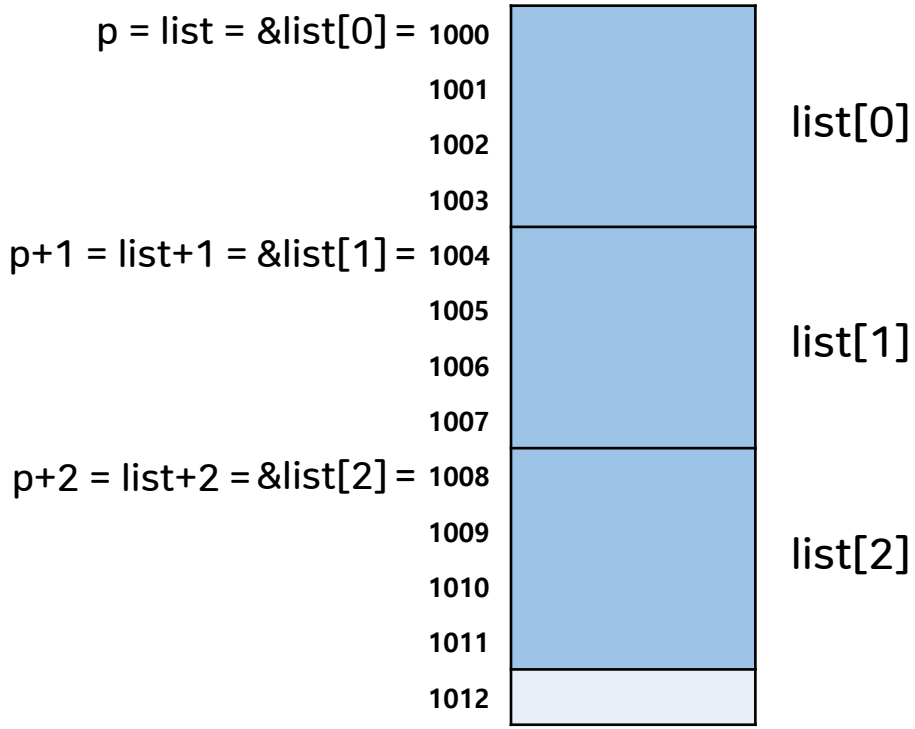


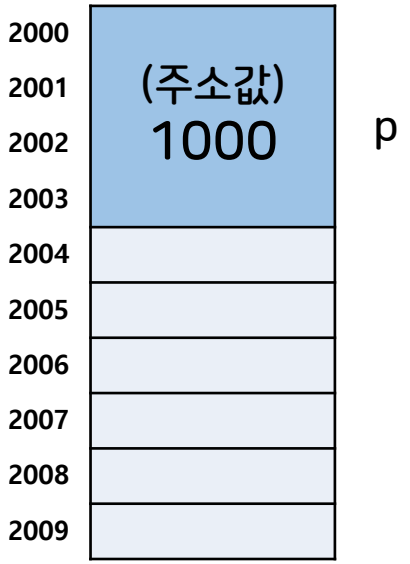
p + k * sizeof(pointer의 base type)

p = 1000
(p+1) =1004
(p+2) =1008

배열과 포인터

```
int main(void)
{
  int list[3];
  int *p = list;
}
```

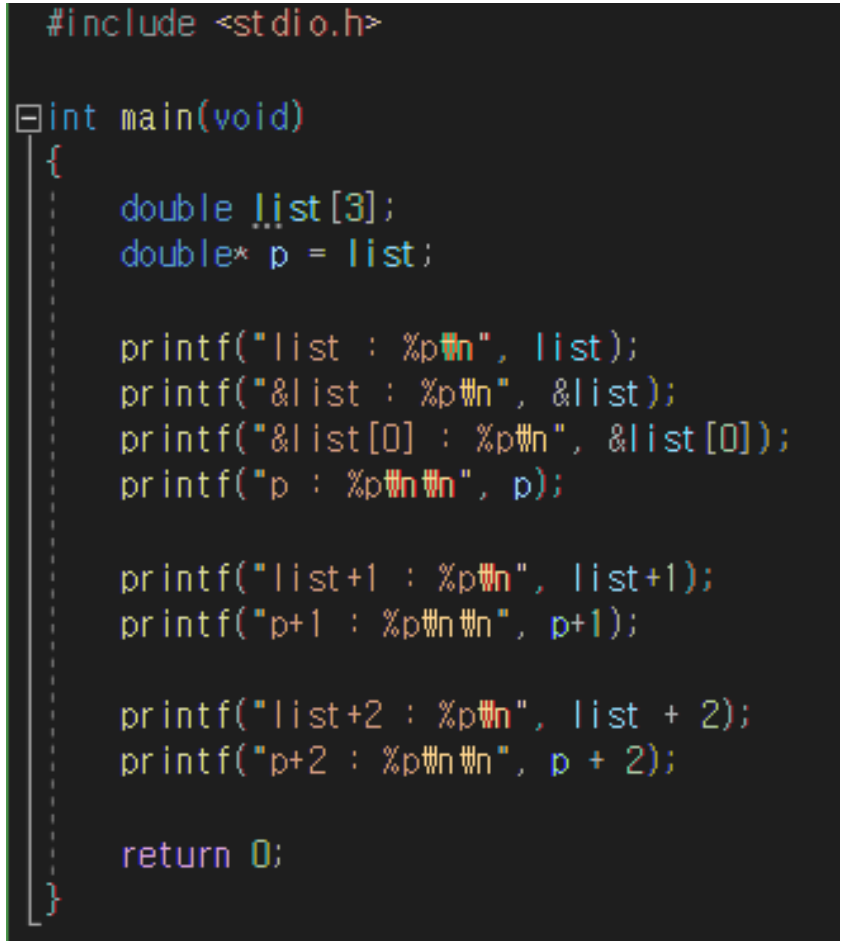




p + k * sizeof(pointer의 base type)

p = 1000
(p+1) =1004
(p+2) =1008

실습 4



```c
#include <stdio.h>

int main(void)
{
    double list[3];
    double* p = list;

    printf("list : %p₩n", list);
    printf("&list : %p₩n", &list);
    printf("&list[0] : %p₩n", &list[0]);
    printf("p : %p₩n₩n", p);

    printf("list+1 : %p₩n", list+1);
    printf("p+1 : %p₩n₩n", p+1);

    printf("list+2 : %p₩n", list + 2);
    printf("p+2 : %p₩n₩n", p + 2);

    return 0;
}
```

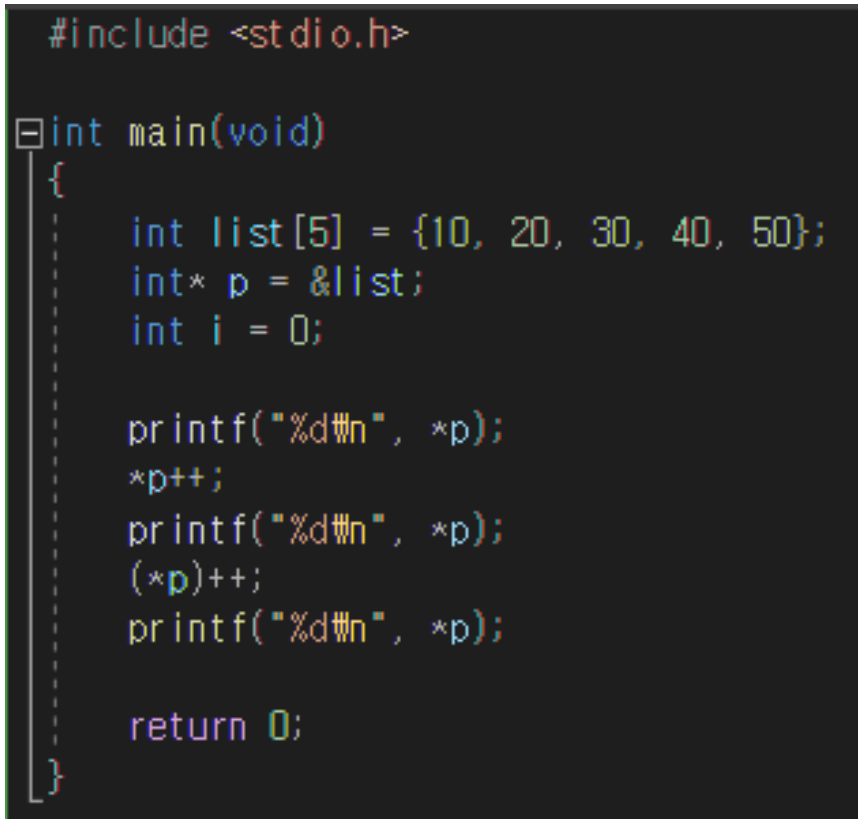
```
list : 0000000F98CFFCD8
&list : 0000000F98CFFCD8
&list[0] : 0000000F98CFFCD8
p : 0000000F98CFFCD8

list+1 : 0000000F98CFFCE0
p+1 : 0000000F98CFFCE0

list+2 : 0000000F98CFFCE8
p+2 : 0000000F98CFFCE8
```

실습 5

```c
#include <stdio.h>

int main(void)
{
    char str[10] = "abc123";
    char* p = str;

    for (int i = 0; i < 10; i++)
    {
        printf("%c", str[i]);
    }
    printf("₩n");

    for (int i = 0; i < 10; i++)
    {
        printf("%c", *(str + i));
    }
    printf("₩n");

    for (int i = 0; i < 10; i++)
    {
        printf("%c", *(p + i));
    }
    printf("₩n");

    return 0;
}
```

```
abc123
abc123
abc123
```

실습 6



```c
#include <stdio.h>

int main(void)
{
    int list[5] = {10, 20, 30, 40, 50};
    int* p = &list;
    int i = 0;

    printf("%d₩n", *p);
    *p++;
    printf("%d₩n", *p);
    (*p)++;
    printf("%d₩n", *p);

    return 0;
}
```

```
10
20
21
```

다차원 배열과 포인터

ROBIT
ROBOT SPORT GAME TEAM

char list[3][4];

list

list[0]	list[0][0] 100	list[0][1] 101	list[0][2] 102	list[0][3] 103
list[1]	list[1][0] 104	list[1][1] 105	list[1][2] 106	list[1][3] 107
list[2]	list[2][0] 108	list[2][1] 109	list[2][2] 110	list[2][3] 111

다차원 배열과 포인터

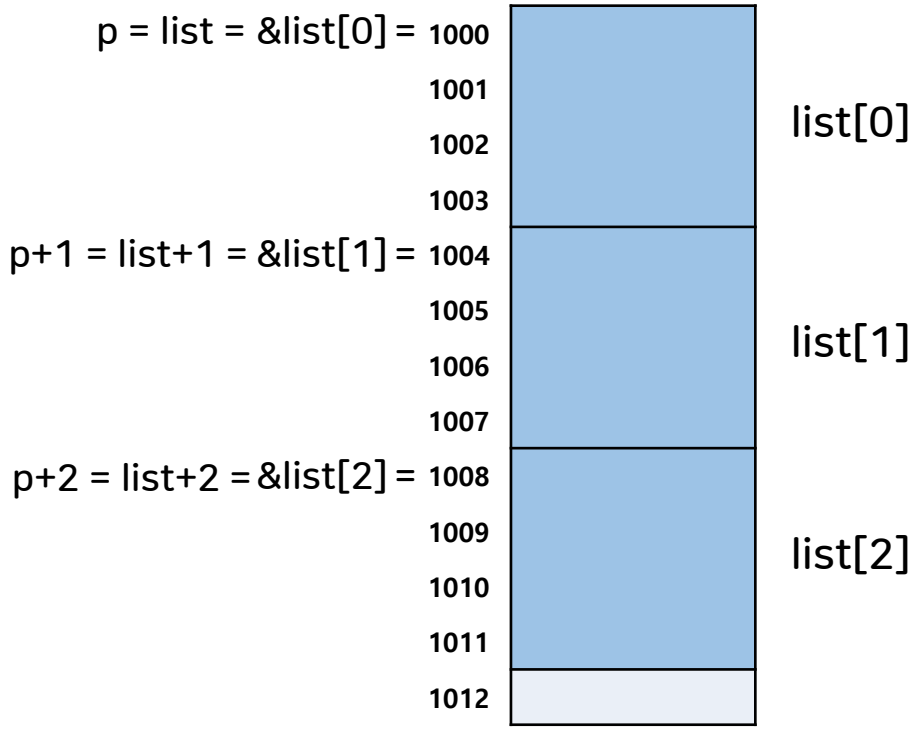
ROBIT
ROBOT SPORT GAME TEAM

char list[3][4];

list=p

list[0]=p	list[0][0] 100	list[0][1] 101	list[0][2] 102	list[0][3] 103
list[1]=p+1	list[1][0] 104	list[1][1] 105	list[1][2] 106	list[1][3] 107
list[2]=p+2	list[2][0] 108	list[2][1] 109	list[2][2] 110	list[2][3] 111
	*(p+n)	*(p+n)+1	*(p+n)+2	*(p+n)+3

다차원 배열과 포인터


p = list = &list[0] =
1000
1001
1002
1003
p+1 = list+1 = &list[1] =
1004
1005
1006
1007
p+2 = list+2 = &list[2] =
1008
1009
1010
1011
1012
list[0]
list[1]
list[2]

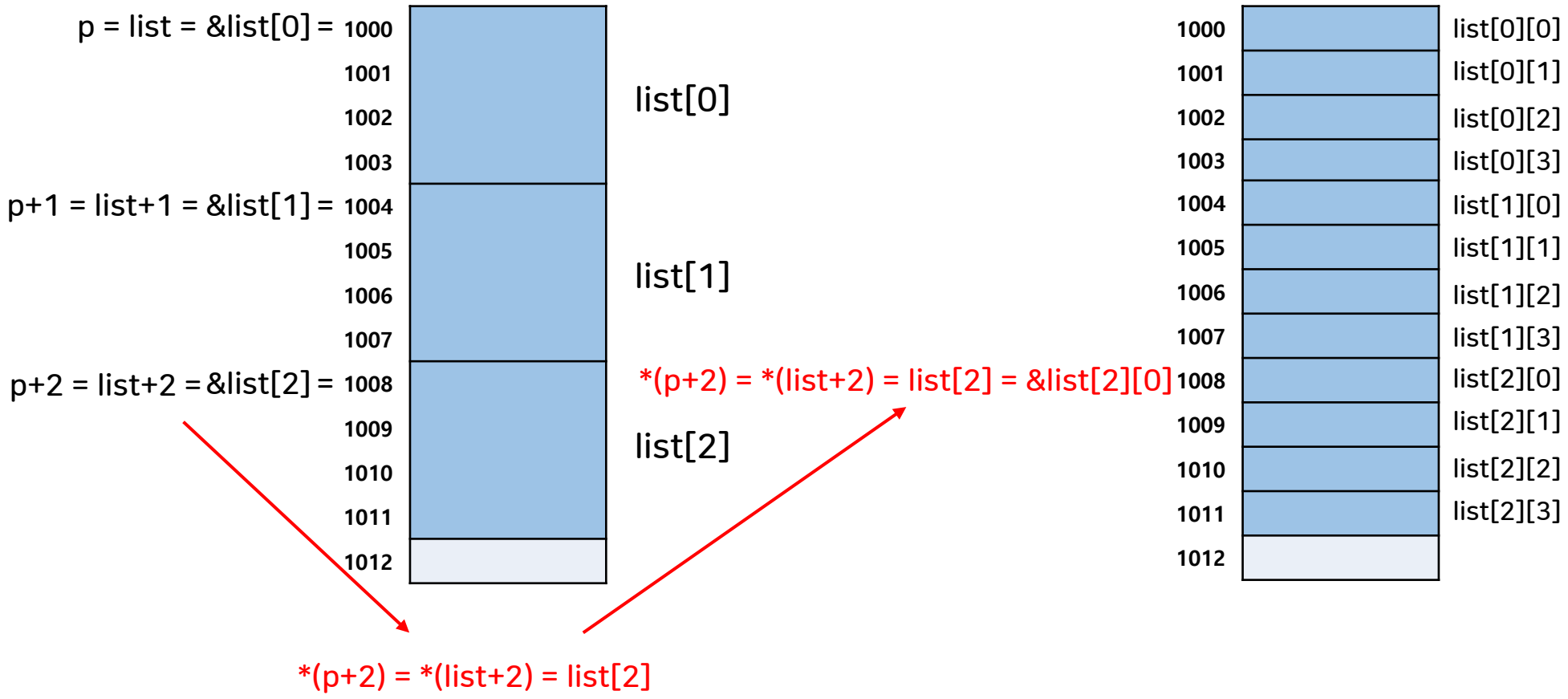
다차원 배열과 포인터


p = list = &list[0] =
1000
1001
1002
1003
p+1 = list+1 = &list[1] =
1004
1005
1006
1007
p+2 = list+2 = &list[2] =
1008
1009
1010
1011
1012
list[0]
list[1]
list[2]

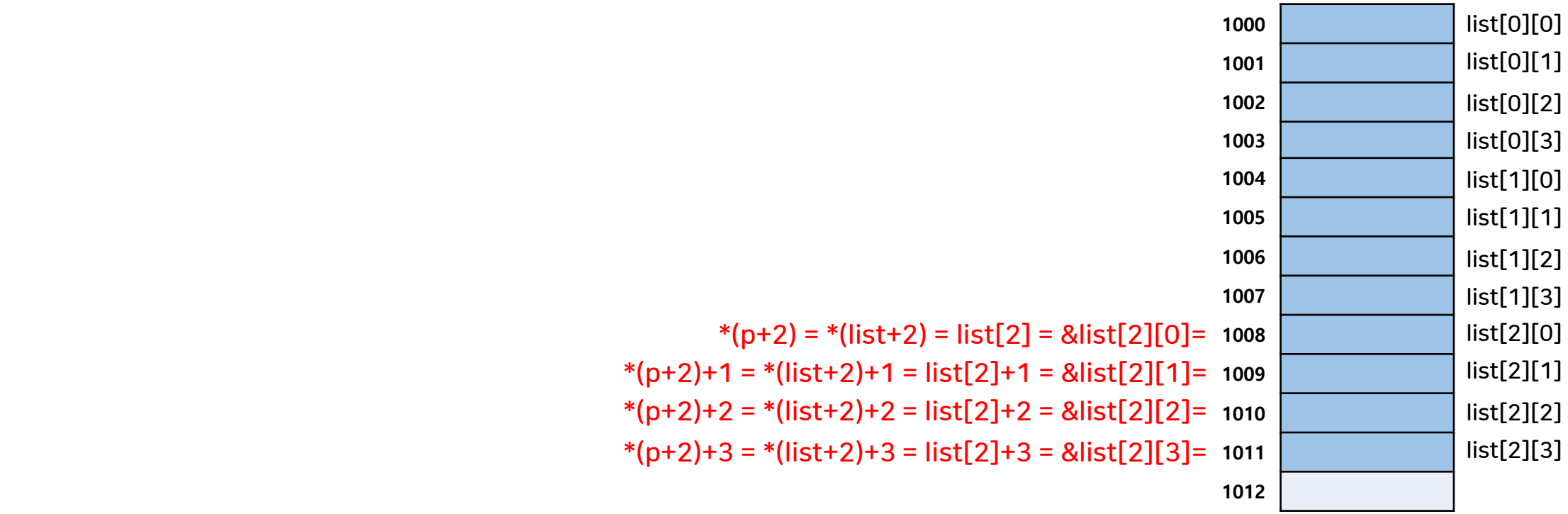
다차원 배열과 포인터

*(p+2) = *(list+2) = list[2]

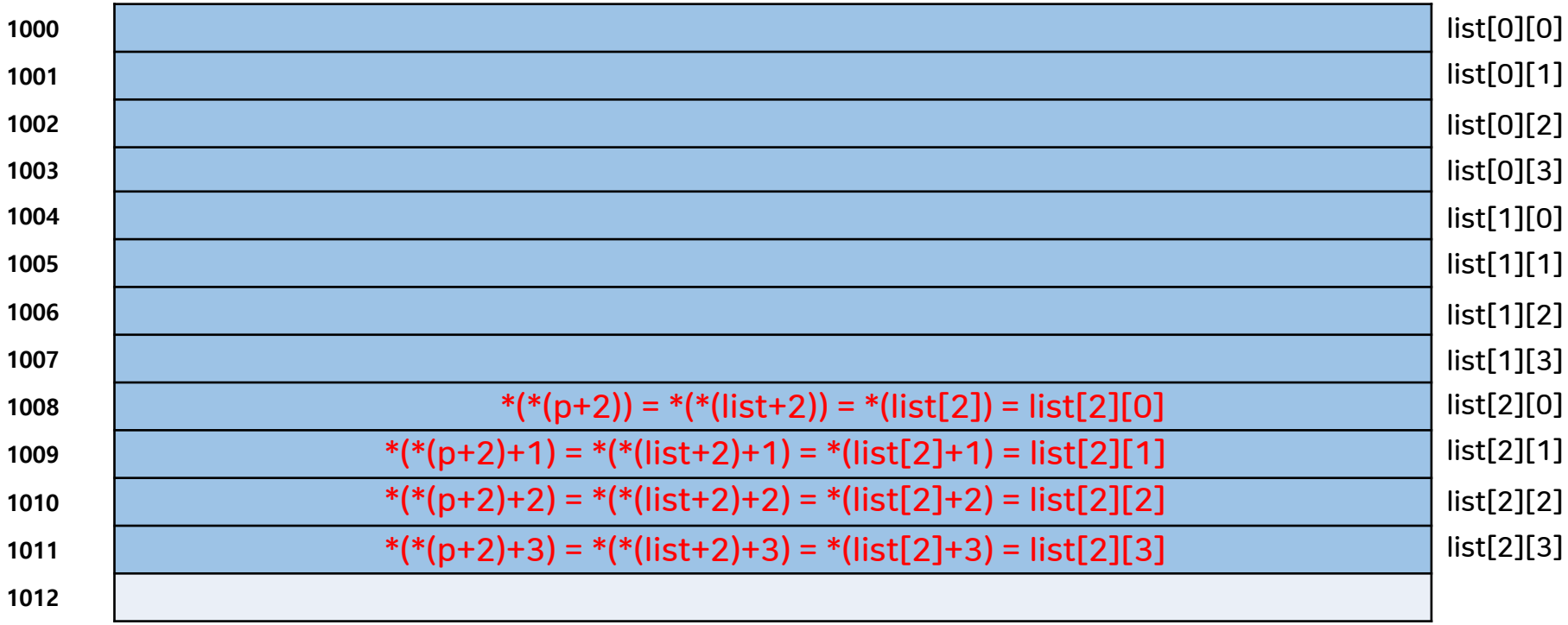
다차원 배열과 포인터


p = list = &list[0] =
1000
1001
1002
1003
p+1 = list+1 = &list[1] =
1004
1005
1006
1007
p+2 = list+2 = &list[2] =
1008
1009
1010
1011
1012
list[0]
list[1]
list[2]
*(p+2) = *(list+2) = list[2] = &list[2][0]
*(p+2) = *(list+2) = list[2]
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
list[0][0]
list[0][1]
list[0][2]
list[0][3]
list[1][0]
list[1][1]
list[1][2]
list[1][3]
list[2][0]
list[2][1]
list[2][2]
list[2][3]

다차원 배열과 포인터


1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
list[0][0]
list[0][1]
list[0][2]
list[0][3]
list[1][0]
list[1][1]
list[1][2]
list[1][3]
list[2][0]
list[2][1]
list[2][2]
list[2][3]
*(p+2) = *(list+2) = list[2] = &list[2][0]=
*(p+2)+1 = *(list+2)+1 = list[2]+1 = &list[2][1]=
*(p+2)+2 = *(list+2)+2 = list[2]+2 = &list[2][2]=
*(p+2)+3 = *(list+2)+3 = list[2]+3 = &list[2][3]=

다차원 배열과 포인터


1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
((p+2)) = *(*(list+2)) = *(list[2]) = list[2][0]
((p+2)+1) = *(*(list+2)+1) = *(list[2]+1) = list[2][1]
((p+2)+2) = *(*(list+2)+2) = *(list[2]+2) = list[2][2]
((p+2)+3) = *(*(list+2)+3) = *(list[2]+3) = list[2][3]
list[0][0]
list[0][1]
list[0][2]
list[0][3]
list[1][0]
list[1][1]
list[1][2]
list[1][3]
list[2][0]
list[2][1]
list[2][2]
list[2][3]

실습 7

```c
#include <stdio.h>

int main(void)
{
    int list[4][3] = {{1,  2,  3},  {4,  5,  6},
                      {7,  8,  9},  {10, 11, 12}};

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 3; j++)
        {
            printf("%d ", *(*(list + i) + j));
        }
        printf(" ");
    }
    printf("₩n");

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 3; j++)
        {
            printf("%d ", list[i][j]);
        }
        printf(" ");
    }
    printf("₩n");
    // *(*(list + i) + j) 는 list[i][j]와 같다.

    printf("list : %d₩n", list);
    printf("list : %d₩n", &list);
    printf("list : %d₩n", &list[0]);
    printf("list : %d₩n", &list[0][0]);

    printf("list : %d₩n", *list);
    printf("list : %d₩n", **list);

    return 0;
}
```

```
1 2 3  4 5 6  7 8 9  10 11 12
1 2 3  4 5 6  7 8 9  10 11 12
list : 561116936
list : 561116936
list : 561116936
list : 561116936
list : 561116936
list : 1
```

Const 포인터

ROBIT
ROBOT SPORT GAME TEAM

상수(Constant)는 변수의 반대(?) 개념으로 생각하면 될듯하다. 즉 변수는 상황에 따라 그 값이 변경가능하지만 상수인 const 로 지정을 하면 그 값을 '절대로' 바꿀수 없다.
또한 변수와는 달리, 처음 상수를 정의할 때 값을 지정해 주어야 한다.

```c
#include <stdio.h>

int main(void)
{
    char ch = 'c';
    char c = 'a';

    char *const ptr = &ch;
    ptr = &c; // 에러 발생!

    return 0;
}
```

const int * p;	• "p" can have another address • The value pointed by "p" cannot be changed
int * const p;	• "p" cannot be changed excepting its initialized address • The value pointed by "p" can be changed
const int * const p;	• "p" cannot be changed excepting its initialized address • The value pointed by "p" cannot be changed

ROBIT
ROBOT SPORT GAME TEAM

과제

과제 공지

ROBIT
ROBOT SPORT GAME TEAM

1. 제출 기한 : 수요일 수업 전까지

2. 제출 형식 : 프로젝트 파일, 보고서(PDF)를 (로빛_18기_수습단원_이름)으로 압축 후
kwrobit2023@gmail.com 으로 제출
(보고서는 코드 설명과 실행 화면 첨부, 메일 제목은 C언어_n일차_이름)

3. 채점 기준:
1) 프로그램의 실행가능 여부
2) 교육하지 않은 C언어 개념 사용시 감점
3) 예외처리
4) 효율적인 코드 작성
5) 제출 형식

과제 1

ROBIT
ROBOT SPORT GAME TEAM

call by value, call by reference에 대한 공부한 후 해당 내용 및 scanf 가 왜 & 를 전달 받는지 이유를 함께 작성 레포트 제출(1page)

과제 1

ROBIT
ROBOT SPORT GAME TEAM

call by value, call by reference에 대한 공부한 후 해당 내용 및 scanf 가 왜 & 를 전달 받는지 이유를 함께 작성 레포트 제출(1page)

과제 2

ROBIT
ROBOT SPORT GAME TEAM

비어있는 공집합 S가 주어졌을 때, 아래 연산을 수행하는 프로그램을 작성하시오.

- add x: S에 x를 추가한다. (1 ≤ x ≤ 20) S에 x가 이미 있는 경우에는 연산을 무시한다.
- remove x: S에서 x를 제거한다. (1 ≤ x ≤ 20) S에 x가 없는 경우에는 연산을 무시한다.
- check x: S에 x가 있으면 1을, 없으면 0을 출력한다.
- toggle x: S에 x가 있으면 x를 제거하고, 없으면 x를 추가한다. (1 ≤ x ≤ 20)
- all: S를 {1, 2, ..., 20} 으로 바꾼다.
- empty: S를 공집합으로 바꾼다.

```
연산을 선택하세요.
add : ~
remove : ~
check : ~
toggle : ~
all : ~
empty : ~

add3
input : add3

집합 : { 3, }

연산을 선택하세요.
add : ~
remove : ~
check : ~
toggle : ~
all : ~
empty : ~

add9
input : add9

집합 : { 3,  9, }

연산을 선택하세요.
add : ~
remove : ~
check : ~
toggle : ~
all : ~
empty : ~
```

조건 :
add,remove,check,toggle,all,empty 모두 포인터 함수로 변환

과제 2

ROBIT
ROBOT SPORT GAME TEAM

과제 2 의 조건

1. 동적 할당으로 최대 데이터 저장 크기 지정
2. 아래 조건 만족

- add x: S에 x(문자열)를 추가한다. (char형 데이터 저장) S에 x가 이미 있는 경우에는 연산을 무시한다.
- remove x: S에서 x(문자열)를 제거한다. S에 x가 없는 경우에는 연산을 무시한다.
- check x: S에 x(문자열)가 있으면 1을, 없으면 0을 출력한다.
- toggle x: S에 x(문자열)가 있으면 x를 제거하고, 없으면 x를 추가한다.
- all: S를 {1, 2, …. N} 으로 바꾼다.
- empty: S를 공집합으로 바꾼다.(메모리 해제 후 다시 최대 데이터 저장 크기 입력을 받아 반복)
- Size : S에 입력된 데이터의 개수 출력.

3. 최대 저장 가능한 배열 크기를 넘은 경우 예외 처리(isFull 함수 만들기)

조건 :
add,remove,check,toggle,all,empty 모두 포인터 함수로 변환

과제 3

```
int main()
{
    int **arr = NULL;
    int row,col,sizeRow,sizeCol;

    printf("열의 수를 입력하세요:");
    scanf("%d",&sizeCol);
    printf("행의 수를 입력하세요:");
    scanf("%d",&sizeRow);

    row = sizeRow;
    col = sizeCol;
```

2차원 동적 메모리 할당 필요

```
    arr_ij(&sizeRow,&sizeCol,arr);

    print(&row,&col,arr);

    for(int i=0; i<row; i++){
        free(arr[i]);
    }

    return 0;
}
```

조건 :
void print(int *row, int *col, int **pArr) 함수 사용
void arr_ij(int *sizeRow, int *sizeCol, int **pArr) 함수 사용
(달팽이 만드는 함수)

```
열의 수를 입력하세요:10
행의 수를 입력하세요:10
  1   2   3   4   5   6   7   8   9  10
 36  37  38  39  40  41  42  43  44  11
 35  64  65  66  67  68  69  70  45  12
 34  63  84  85  86  87  88  71  46  13
 33  62  83  96  97  98  89  72  47  14
 32  61  82  95 100  99  90  73  48  15
 31  60  81  94  93  92  91  74  49  16
 30  59  80  79  78  77  76  75  50  17
 29  58  57  56  55  54  53  52  51  18
 28  27  26  25  24  23  22  21  20  19
```

과제 4

두 문자열에 모두 포함된 가장 긴 공통 부분 문자열찾기.

조건
1. 두 입력은 char 형 데이터이며 동적 할당을 이용하여 크기를 지정해준다.
2. 동적 할당을 이용하여 저장 가능한 데이터의 길이보다 긴 문자열은 저장 가능한 만큼만 저장하고 나머지는 버린다.
3. 전체 시스템은 무한 루프로 구성 (quit을 입력하면 프로그램 종료)
4. 2 개의 배열과 결과를 저장할 배열, 즉 총 3 개의 Input 이 있으며, 반환형은 가장 긴 공통 부분의 문자열 길이를 담을 수 있는 int형인 함수를 사용한다.
5. 결과를 저장할 배열은 최초 2 개의 배열을 동적 할당 하는 과정에서 가장 짧은 길이를 가진 배열의 크기와 똑같은 크기로 동적 할당 한다.
6. 출력은 결과를 저장한 배열의 전체 원소와 함수의 반환으로 받은 문자열의 길이를 출력.

예시
입력1 : ajsdfkqWEefasdfegweoijo
입력2 : fkqWEefaEFAEFBVEWQGF

출력 : fkqWEefa / 8

과제 5

세 문자열에 모두 포함된 가장 긴 공통 부분 문자열찾기.

조건
1. 세 입력은 char 형 데이터이며 동적 할당을 이용하여 크기를 지정해준다.
2. 동적 할당을 이용하여 저장 가능한 데이터의 길이보다 긴 문자열은 저장 가능한 만큼만 저장하고 나머지는 버린다.
3. 전체 시스템은 무한 루프로 구성 (quit을 입력하면 프로그램 종료)
4. 세 개의 배열과 결과를 저장할 배열, 즉 총 4 개의 Input 이 있으며, 반환형은 가장 긴 공통 부분의 문자열 길이를 담을 수 있는 int형인 함수를 사용한다.
5. 결과를 저장할 배열은 최초 3 개의 배열을 동적 할당 하는 과정에서 가장 짧은 길이를 가진 배열의 크기와 똑같은 크기로 동적 할당 한다.
6. 출력은 결과를 저장한 배열의 전체 원소와 함수의 반환으로 받은 문자열의 길이를 출력.

예시
입력1 : ajsdfkqWEefasdfegweoijo
입력2 : fkqWEefaEFAEFBVEWQGF
입력3 : efjqeoijafkqWEeo

출력 : fkqWEe / 6