컴퓨터 프로그래밍 I
(CSE2003-3)

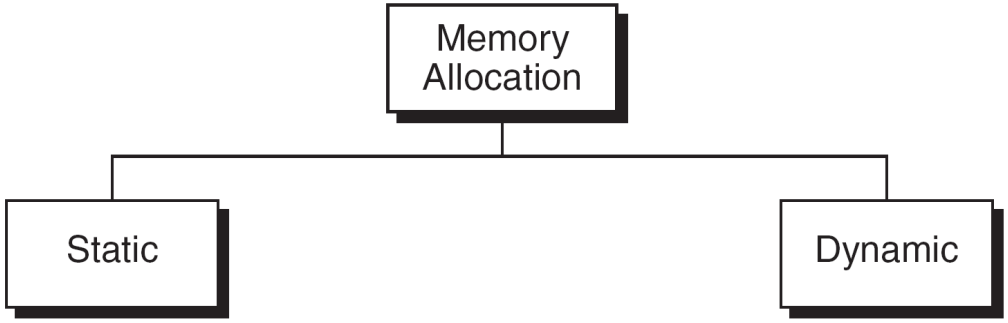
Mon/Wed 16:30-17:45 pm

Lecture 24

Array and Pointers

Memory allocation functions

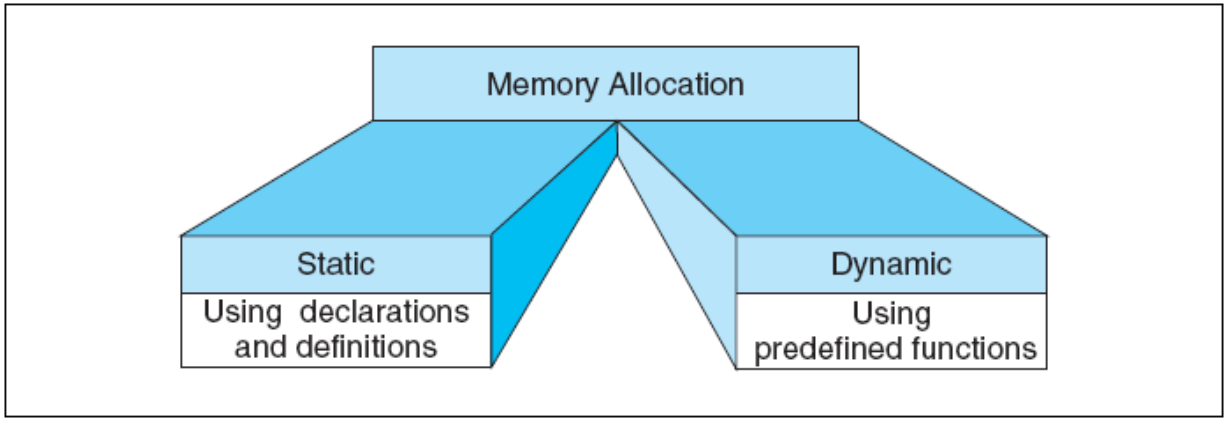
- **C언어를 사용하면서 변수 영역의 메모리를 할당 받는 것으로 다음의 두 가지 방법이 가능하다.**



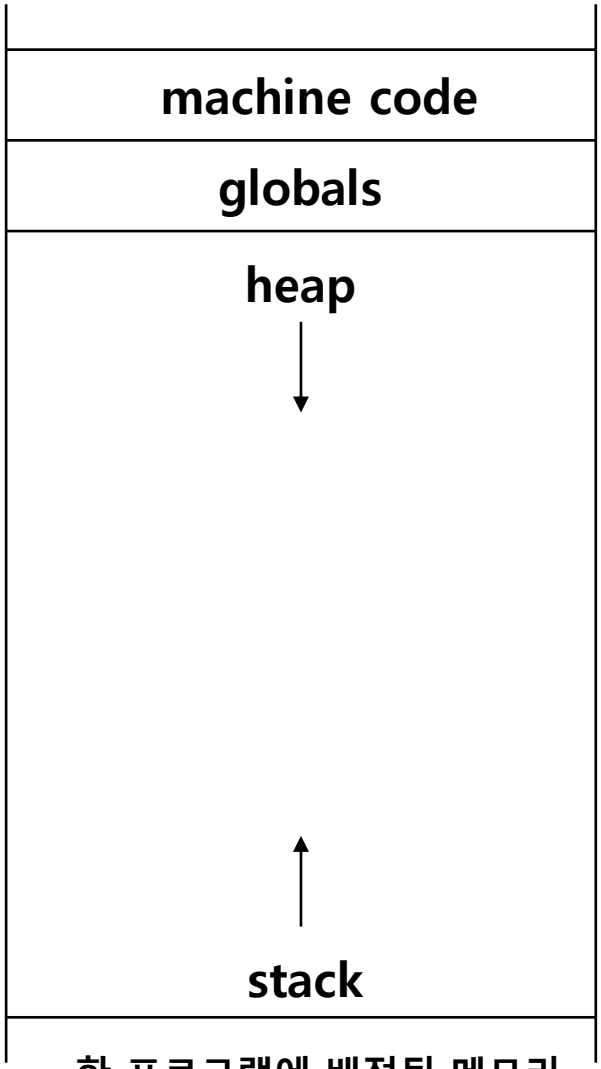
- **Static memory allocation(정적 메모리 할당)**
  - 처리해야 할 데이터의 크기를 미리 다 아는 경우에는 프로그램에 필요한 메모리의 선언과 정의가 완벽하게 다 명시되어 있어야 한다.
    - 변수, 배열 선언 등
    - int i, j, a[100][100];

Memory allocation functions

- **동적 메모리 할당**
  - 얼마나 큰 데이터를 처리해야 하는지 모르는 경우에는 프로그램의 실행 중에 입력되는 데이터에 맞게 기억공간이 할당되어야 한다. ➔ 메모리의 동적 할당(dynamic memory allocation)
  - 메모리의 동적 할당과 해제 기능은 라이브러리로 구현되어 있고, 이를 이용하기 위해서는 반드시 포인터를 이용한다.
    - 동적 할당이란 프로그램 실행 중에 ,즉 run time때 메모리 영역을 할당한다는 뜻이다.



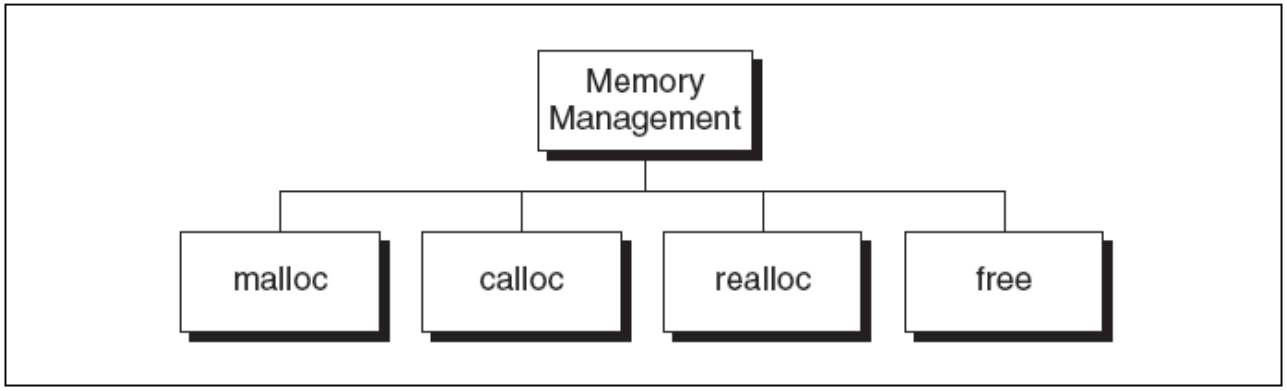
memory 와 프로그램



- machine code: 컴파일된 프로그램이 올라가는 영역
- globals: 글로벌 변수들...
- heap: 동적 할당 메모리 영역
  - malloc()과 같은 동적 메모리 할당 함수가 실행되면 이 영역의 메모리를 가져와서 그것의 주소를 return 한다
- stack은 정적으로 선언된 변수들이 run-time에 배정되는 위치

Memory allocation functions

- **Memory management functions**



- 메모리의 동적 할당에 사용되는 메모리 할당 함수들
  - malloc(), calloc(), realloc()은 메모리를 할당하기 위한 함수
  - free()는 더 이상 사용하지 않는 메모리를 리턴하는 함수

- 함수를 사용하기 위해서는 "stdlib.h" 헤더파일을 포함해야 한다.
  - #include<stdlib.h>

malloc()

- **malloc() 함수는 동적으로 메모리를 할당하기 위한 함수이다.**
  - malloc(n)과 같이 호출하면 n byte의 메모리를 가지고 와서 그것의 시작 주소(pointer)를 return 한다.
  - malloc() 함수는 할당한 메모리의 시작주소를 void 포인터로써 return한다. 할당에 실패한 경우 NULL을 return 한다.

  - malloc() 함수의 정의는 다음과 같다

```
void *malloc (size_t size);
```

    - 항상 void 포인터를 return하고 파라메터로는 unsigned integer 형태로 받는다.

Dynamic memory allocation

```
// statically obtain an integer
int x;

// dynamically obtain an integer
int *px = malloc(4);

// dynamically obtain an integer
int *py = malloc(sizeof(int));
```

Dynamic memory allocation

```
// get an integer from the user
int x;
scanf("%u", &x);

// array of floats on the heap
float* heap_array = malloc(x * sizeof(float));
```

Dynamic memory allocation

- 문제점:
  - 동적으로 할당 받은 메모리는 사용한 함수가 종료되어도 자동으로 시스템에 반환되지 않는다!!!
  - 동적으로 할당된 메모리를 다시 시스템이 회수하지 못하는 현상을 memory leak라고 하는데, 이렇게 지속적으로 메모리를 회수하지 못할 경우 시스템의 성능이 심각할 정도로 문제가 된다.
  - 동적으로 할당된 메모리를 사용할 때, 반드시 해당 메모리를 free() 시켜 주는 것이 중요하다!!!

Dynamic memory 반환

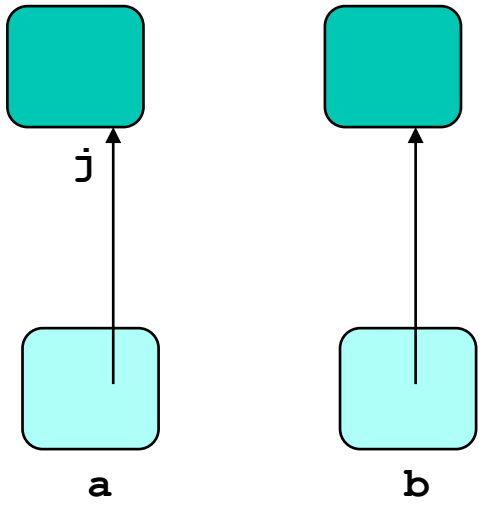
```
char* word = malloc(50 * sizeof(char));
// word를 가지고 작업 시작

// word를 가지고 작업 완료
free(word)
```

Three golden rules

1. malloc()으로 할당 받은 메모리는 반드시 나중에 free() 시켜주어야 한다.
2. malloc()으로 받은 메모리만 free() 시켜주면 된다
3. free()는 꼭 한 번만 시행한다.
   - 같은 메모리를 두 번 이상 free() 시키면 에러가 남

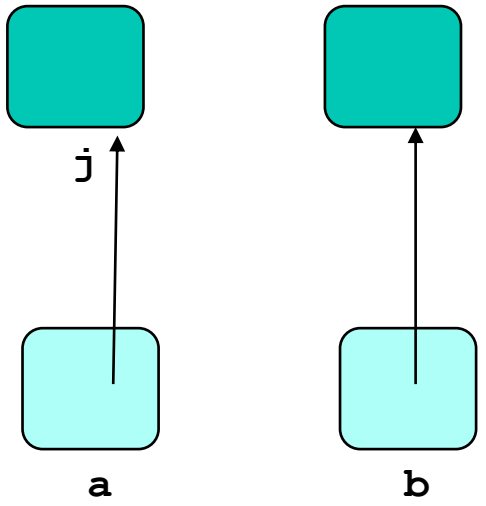
Dynamic memory allocation

```
int  j;
int* a;
int* b = malloc(sizeof(int))
a = &j;
```



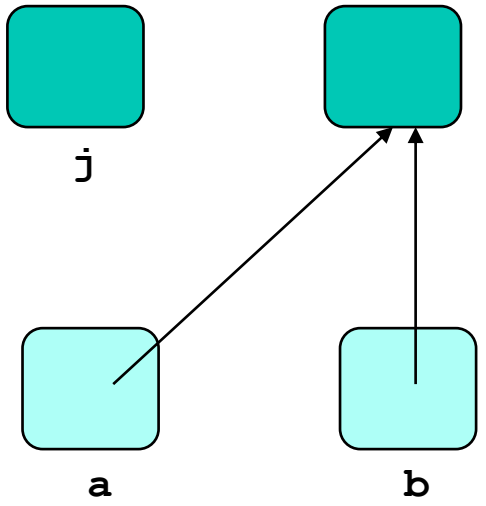
Dynamic memory allocation

```
int  j;
int* a;
int* b = malloc(sizeof(int))
a = &j;
a = b;
```



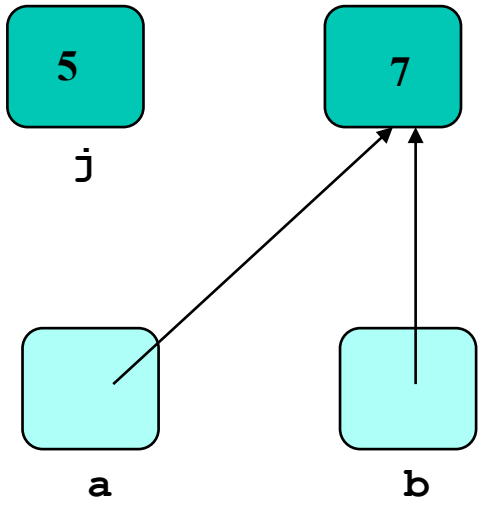
Dynamic memory allocation

```
int  j;
int* a;
int* b = malloc(sizeof(int))
a = &j;
a = b;
```



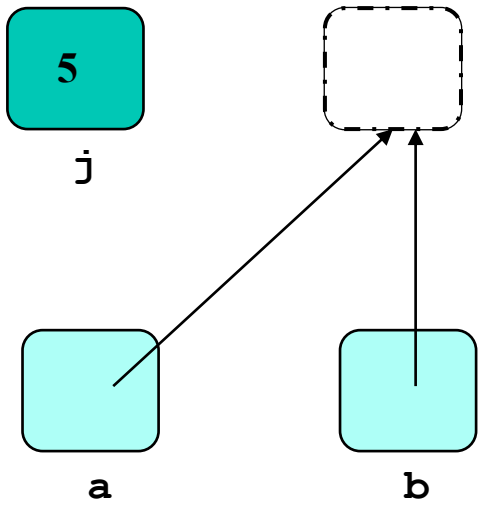
Dynamic memory allocation

```
int  j;
int* a;
int* b = malloc(sizeof(int))
a = &j;
a = b;
j = 5;
*b = j + 2;
```



Dynamic memory allocation

```
int  j;
int* a;
int* b = malloc(sizeof(int))
a = &j;
a = b;
j = 5;
*b = j + 2;
free(b);
*a = 11;
```



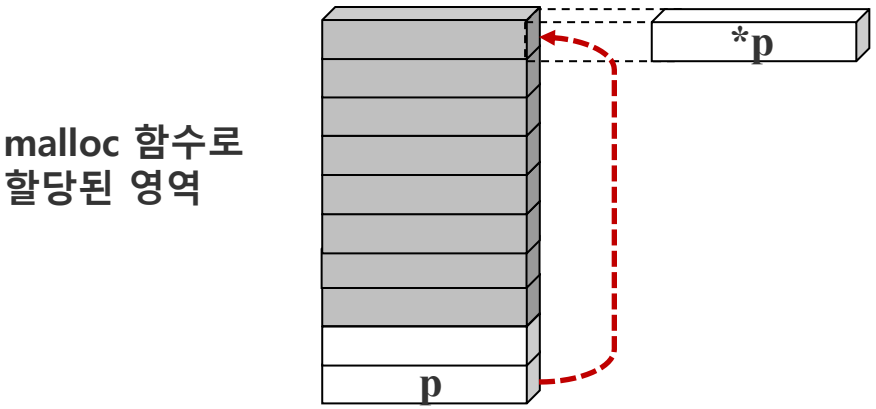
Unpredictable result since it's gone…
May cause segmentation fault, or what…

Dynamic allocation of array

Dynamic allocation of array

```
int *p;
p = (int *)malloc(8 * sizeof(int));
```

- malloc은 8개의 int를 위한 기억 장소를 할당해 그 시작 주소값을 되돌려 준다.



- 포인터 변수가 있으면 포인터가 가리키는 영역을 배열처럼 사용가능
- 포인터 변수 p가 가리키는 원소의 i번째 원소를 나타내는 표현식
  - *(p+i) , p[i]

malloc()

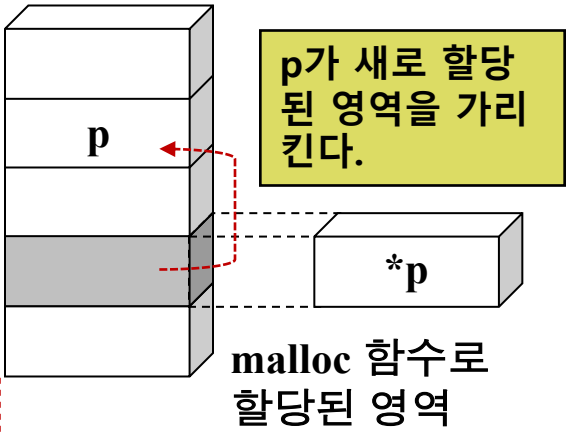
- 예제프로그램 - malloc을 이용한 변수 할당

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        int *p;

        p=(int*)malloc(1*sizeof(int));

        if(p==NULL){
                printf("not allocated\n");
                return 1;
        }

        *p=1;

        printf("%d\n", *p);
        return 0;
}
```

```
[root@mclab chap10]# ./chap10-5
1
[root@mclab chap10]#
```

int형 1개 크기의 메모리 할당하고 그 주소값을 p에 넘겨준다.

새로 할당된 메모리영역에 접근



malloc()

- 예제프로그램 – 메모리를 동적으로 할당 받아 배열로 사용하는 프로그램

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        int *ip;
        int i, sum = 0;

        ip = (int *)malloc(5*sizeof(int));
        if(ip==0) {
                printf("short memory\n");
                return 1;
        }
        printf("Input the age of 5 people : ");
        for(i=0; i<5; i++) {
                scanf("%d", ip+i);
                sum+=ip[i];
        }
        printf("Average : %.1lf\n", sum/5.0);
        free(ip);
        return 0;
}
```

20바이트의 기억공간 할당

```
[root@mclab chap10]# vi chap10-6.c
[root@mclab chap10]# gcc -o chap10-6 chap10-6.c
[root@mclab chap10]# ./chap10-6
Input the age of 5 people : 21 27 24 22 35
Average : 25.8
[root@mclab chap10]#
```

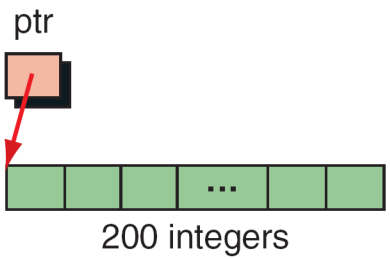
할당 받은 메모리를 배열처럼 사용하여 값을 입력 받는 부분

Contiguous Memory Allocation - calloc()

- 두 번째 메모리 관리 함수는 calloc() 이다.
  - calloc() 함수의 프로토타입은 다음과 같다.

```
void *calloc (size_t element-count,
              size_t element_size);
```

  - 할당 받은 메모리를 사용할 하나의 원소 크기와 전체 원소의 개수를 개별적인 인자로 받는다 .
  - calloc() 함수는 메모리를 할당하면서 해당 메모리를 0으로 초기화 시킨다.




```
if (!(ptr = (int*)calloc (200, sizeof(int)))
    // No memory available
    exit (100) ;

// Memory available
…
```

malloc()과 calloc()의 차이점

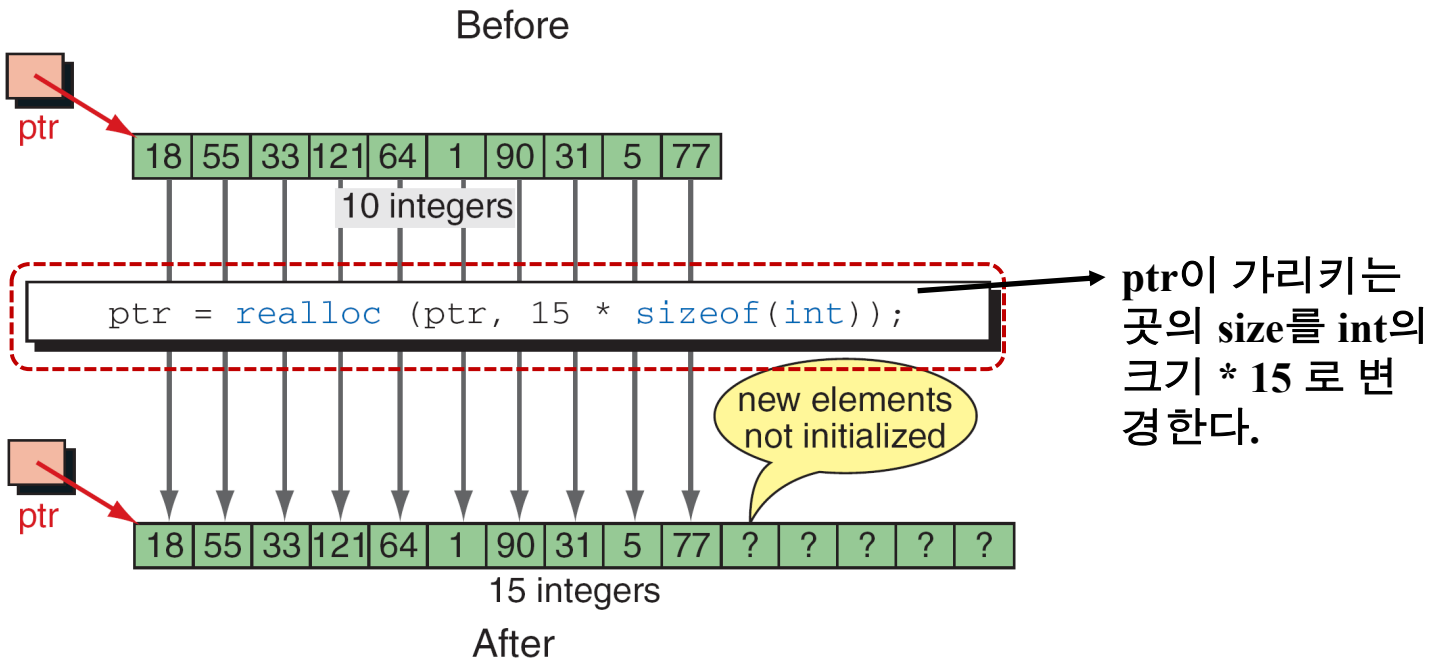
<calloc함수 호출 예>

Reallocation of Memory - realloc ()

- realloc() 함수는 변수의 메모리를 동적으로 변경하기 위하여 사용한다.
- 함수의 프로토타입은 다음과 같다.

```
void *realloc (void *ptr, size_t newSize);
```

- 즉, ptr이 현재 할당하고 있는 메모리의 크기를 newSize로 변경한다.



calloc() & realloc()

➢ 예제 프로그램 - realloc함수를 이용한 양수값 입력 프로그램

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        int *ip;
        int size = 5;
        int cnt = 0;
        int num;
        int i;

        ip=(int *)calloc(size, sizeof(int));
        while(1){
                printf("Input the positive integer : ");
                scanf("%d", &num);
                if(num<=0) break;
                if(cnt<size) {
                        ip[cnt++]=num;
                }
                else {
                        size += 5;
                        ip=(int *)realloc(ip, size*sizeof(int));
                        ip[cnt++]=num;
                }
        }
        for(i=0; i<cnt; i++) {
                printf("%5d", ip[i]);
        }
        printf("\n");
        free(ip);
        return 0;
}
```

일정한 크기의 기억공간을 할당(calloc) 받고 나서, 입력되는 데이터가 양수일 경우에 기억공간을 재할당(realloc)한다.

할당 받은 기억공간이 남아있으면 데이터를 저장한다.

기억공간이 부족하면 크기를 늘려서 재할당 받는다.

```
[root@mclab chap10]# vi chap10-7.c
[root@mclab chap10]# gcc -o chap10-7 chap10-7.c
[root@mclab chap10]# ./chap10-7
Input the positive integer : 36
Input the positive integer : 28
Input the positive integer : 14
Input the positive integer : 0
   36   28   14
[root@mclab chap10]#
```

Releasing Memory - free ()

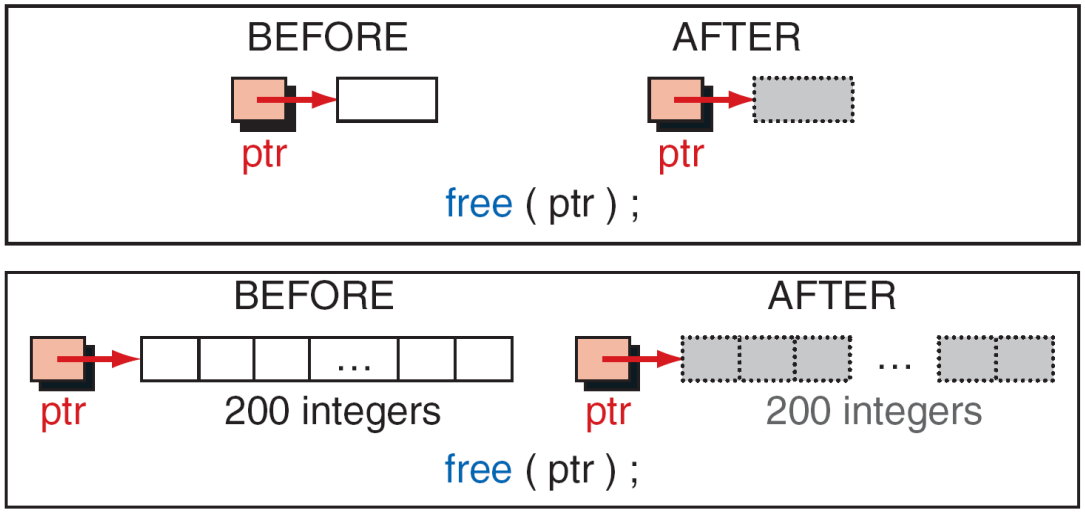
- 동적으로 할당 받은 메모리 영역에 대하여 반환하는 함수
  - free() 함수의 프로토타입은 다음과 같다.

```
void free (void *ptr);
```

  - 사용하지 않는 메모리에 대한 계속적인 할당은 메모리 사용의 낭비의 단점이 있기 때문에 동적으로 할당 받은 필요 없는 메모리 영역에 대해서는 free()를 통해서 메모리를 반환하도록 한다.
  - free() 는 ptr이 가리키는 기억 장소를 해제한다.
  - 단, ptr이 NULL일 때는 아무 일도 하지 않는다.
  - ptr은 malloc(), calloc(), realloc()에 의하여 이전에 할당되었던 기억 장소의 포인터이어야 한다.

Releasing Memory - free ()

- 다음은 free()의 동작을 보여주는 그림이다.



- **Caution!**

동적으로 할당한 기억 장소 영역은 책임지고 해제하지 않으면 안 된다.
영역을 해제할 때까지는 반드시 포인터를 기억해 두어야 한다.

Memory leak

- 더 이상 사용되지 않는 메모리 공간의 반환
  - 동적으로 할당 받은 메모리 공간은 더 이상 해당 공간을 참조하는 포인터가 없는 경우에도 heap영역에 자리를 차지하고 있다.
  - 동적 할당으로 생성된 메모리를 적절한 타이밍에 해제하지 않으면 해당 메모리는 생성된 만큼의 공간을 차지하면서도 정작 사용할 수 없는 상태가 된다

```
int *intArr;
intArr=(int *)malloc(sizeof(int)*5);
intArr=(int *)malloc(sizeof(int)*10);
free(intArr);
```

프로그램이 종료될 때 까지 남아있으나 사용할 수는 없는 메모리

  - 메모리 누수가 반복적으로 일어날 경우 시간이 지나면서 사용할 수 있는 메모리를 모두 사용하여 프로그램이 더 이상 올바르게 기능하지 못하게 된다.
  - 이와 같은 문제를 메모리 누수(memory leak)라고 한다.

Dangling pointer

- **Dangling pointer**
  - 메모리가 삭제되거나 재 할당 될 때 유효하지 않은 메모리 공간을 참조하는 포인터.
  - 해제된 공간을 참조하고 있거나 지역변수를 참조할 때 발생하기 쉽다.

```
int main( ){
    int *intPointer;
    dangle(&intPointer);
    printf("%d", *intPointer);
    return 0;
}
void dangle(int **intPtr){
    int a=1;
    *intPtr=&a;
}
```

함수의 지역변수를 참조하고 있지만 함수 수행 종료 후 해당 변수는 stack에서 제거되기 때문에 dangling pointer가 된다

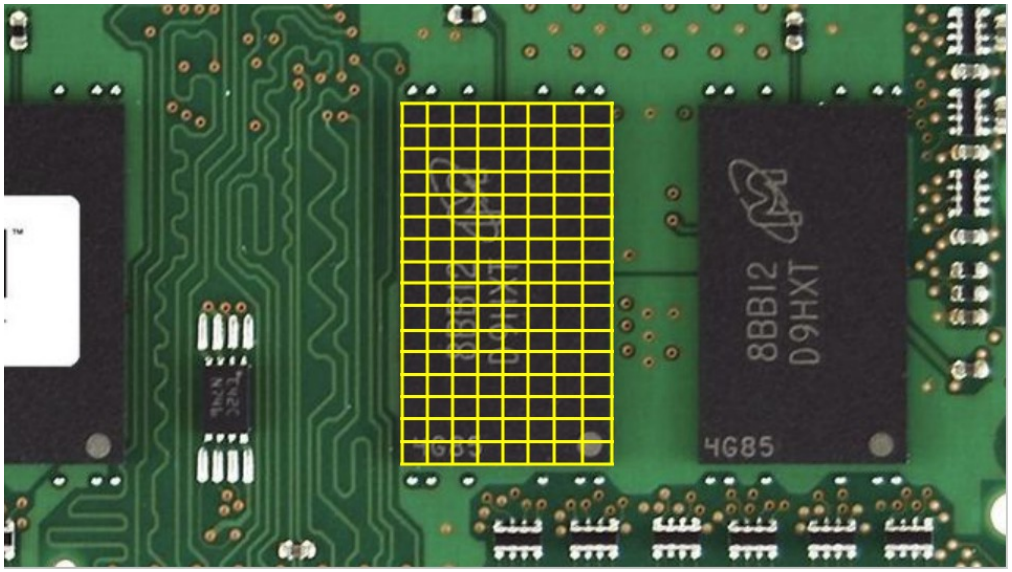
Dangling pointer

```
int main(){
    int *intPointer;
    intPointer=(int*)malloc(sizeof(int)*5);

    for(int i=0;i<5;i++)
        intPointer[i]=i;

    dangle(intPointer);

    printf("%d ", intPointer[1]);

    return 0;

}
void dangle(int *intPtr){
    free(intPtr);
}
```

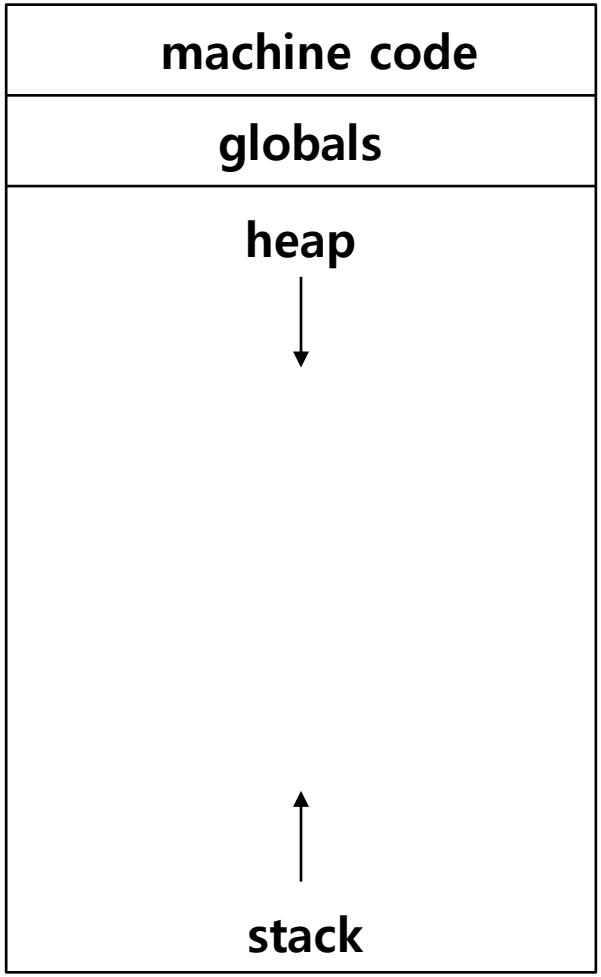
이미 해제된 메모리 공간의 주소를 그대로 참조하고 사용하게 되면 프로그램이 오작동 하게 된다.

- Dangling pointer에 대한 접근은 프로그램의 오작동이나 중단을 유발하지만 컴파일 단계에서 검출되지 않기 때문에 주의가 필요하다.

부록: memory 와 프로그램

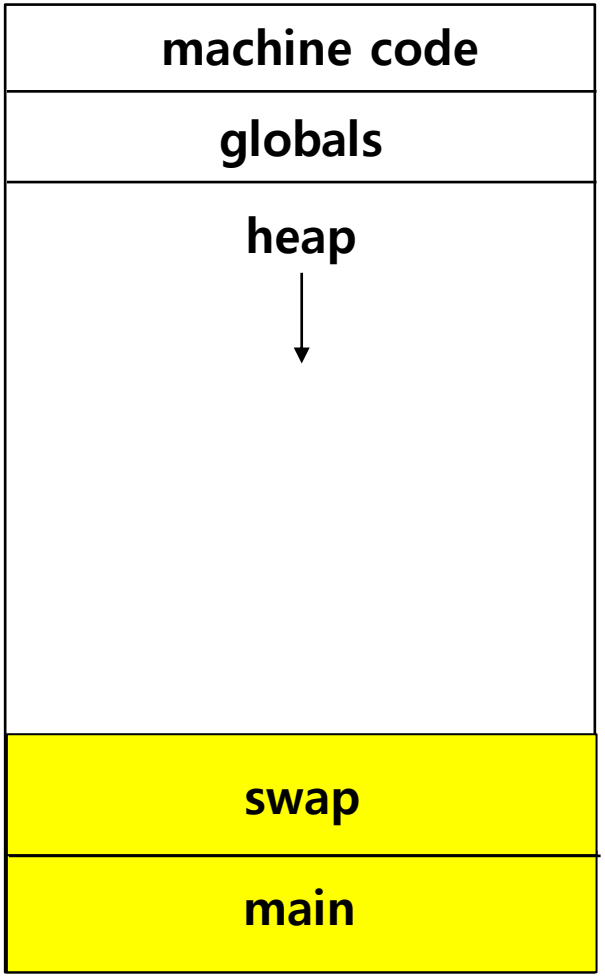


memory 와 프로그램



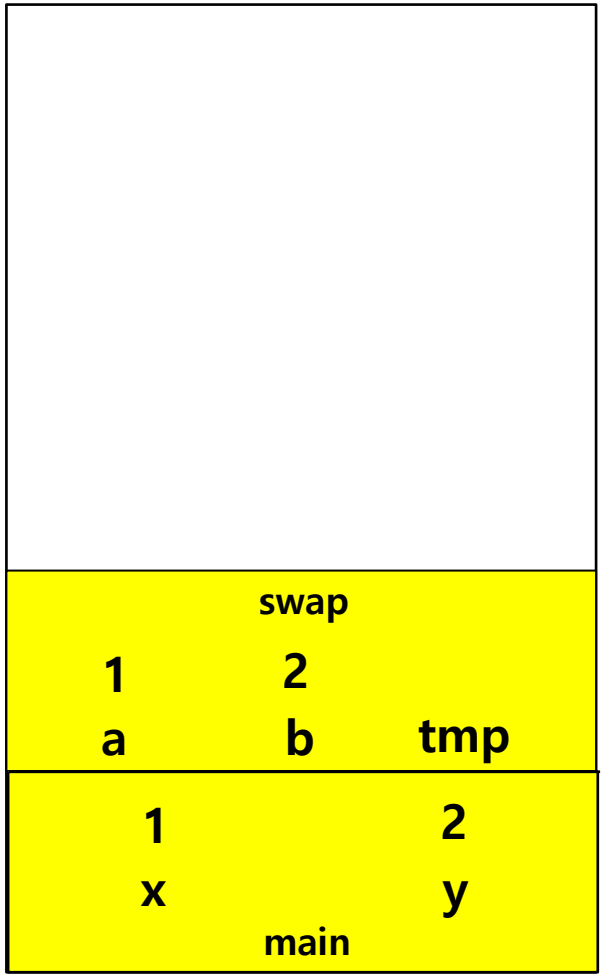

```
void main()
{
    int x = 1;
    int y = 2;
    swap(x, y)
    printf("x = %i, y = %i\n", x, y);
}

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

memory 와 프로그램



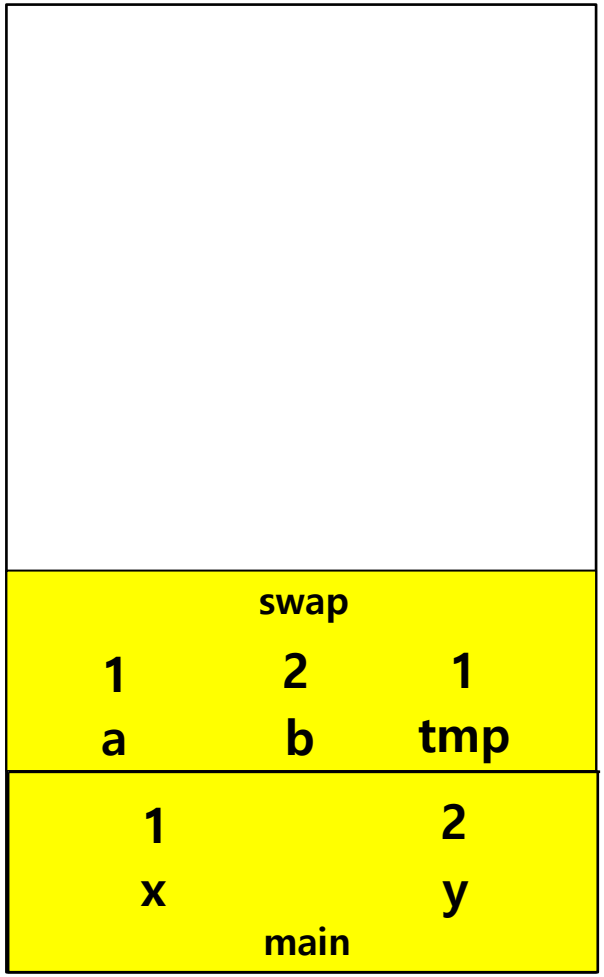

```
void main()
{
    int x = 1;
    int y = 2;
    swap(x, y)
    printf("x = %d, y = %d\n", x, y);
}

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

memory 와 프로그램



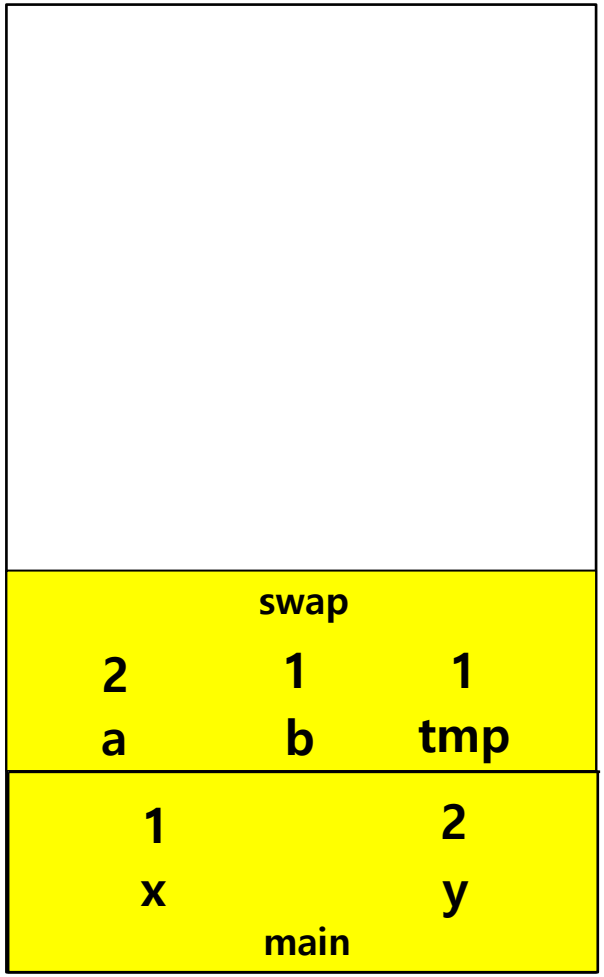

```
void main()
{
    int x = 1;
    int y = 2;
    swap(x, y)
    printf("x = %d, y = %d\n", x, y);
}

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

memory 와 프로그램



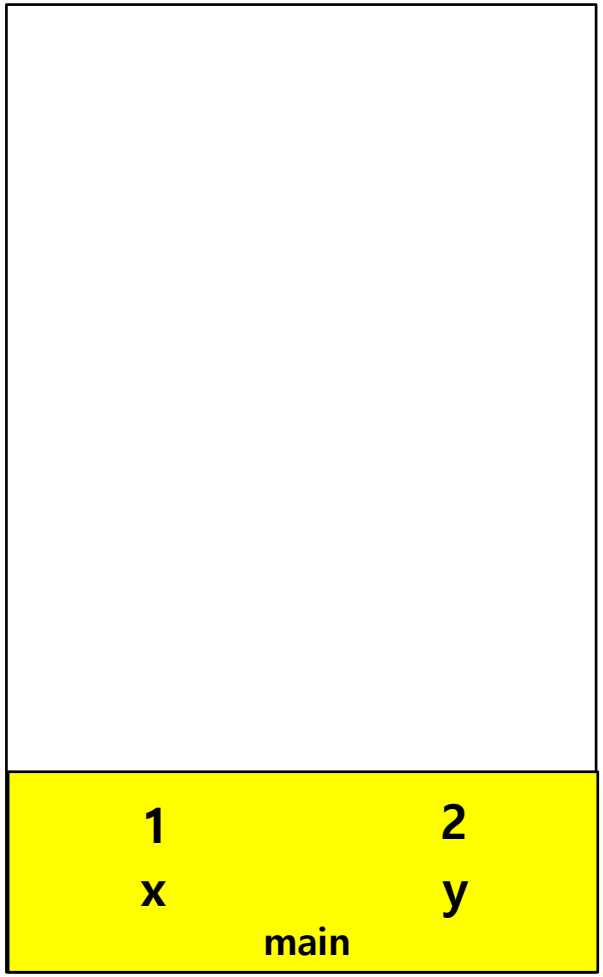

```
void main()
{
    int x = 1;
    int y = 2;
    swap(x, y)
    printf("x = %d, y = %d\n", x, y);
}

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

memory 와 프로그램




```
void main()
{
    int x = 1;
    int y = 2;
    swap(x, y)
    printf("x = %d, y = %d\n", x, y);
}

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

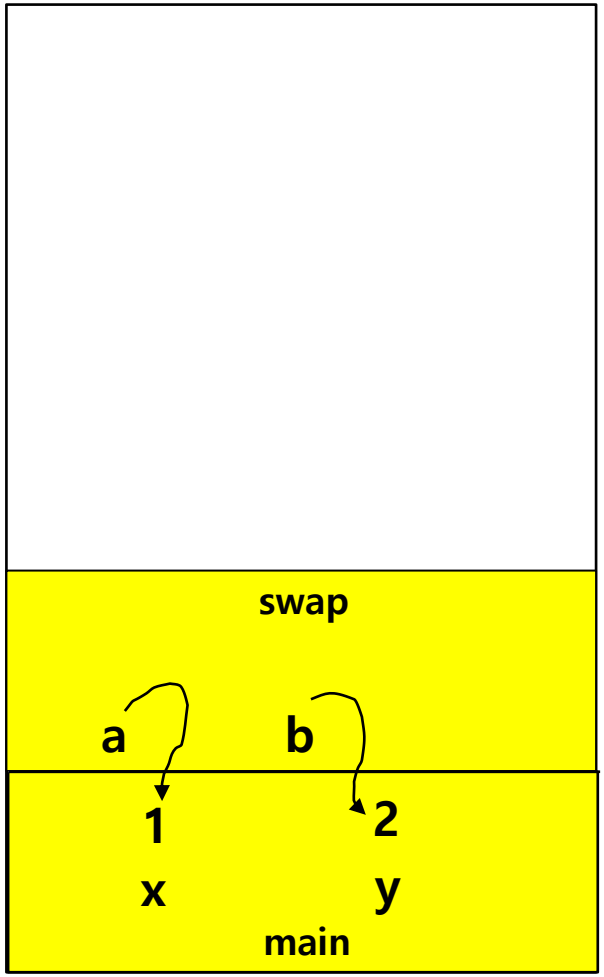
memory 와 프로그램




```
void main()
{
    int x = 1;
    int y = 2;
    swap(x, y)
    printf("x = %d, y = %d\n", x, y);
}

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

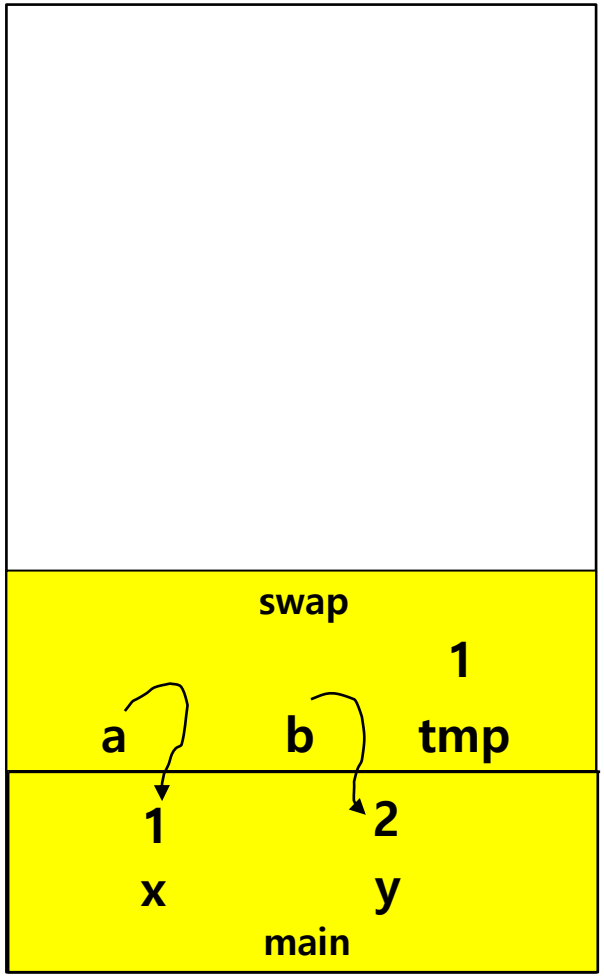
출력 결과: x = 1, y = 2

memory 와 프로그램



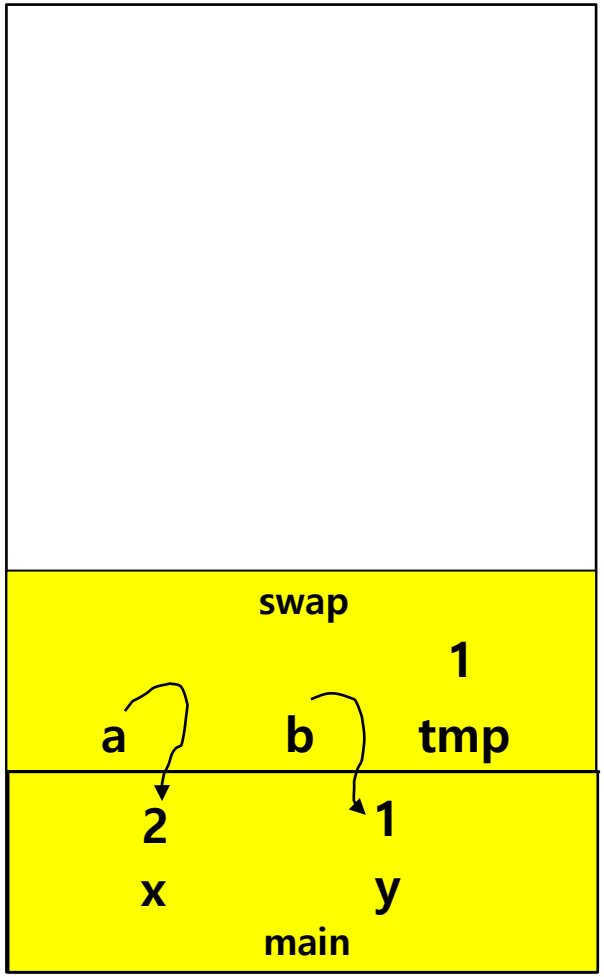

```
void main()
{
    int x = 1;
    int y = 2;
    swap(&x, &y)
    printf("x = %d, y = %d\n", x, y);
}

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

memory 와 프로그램



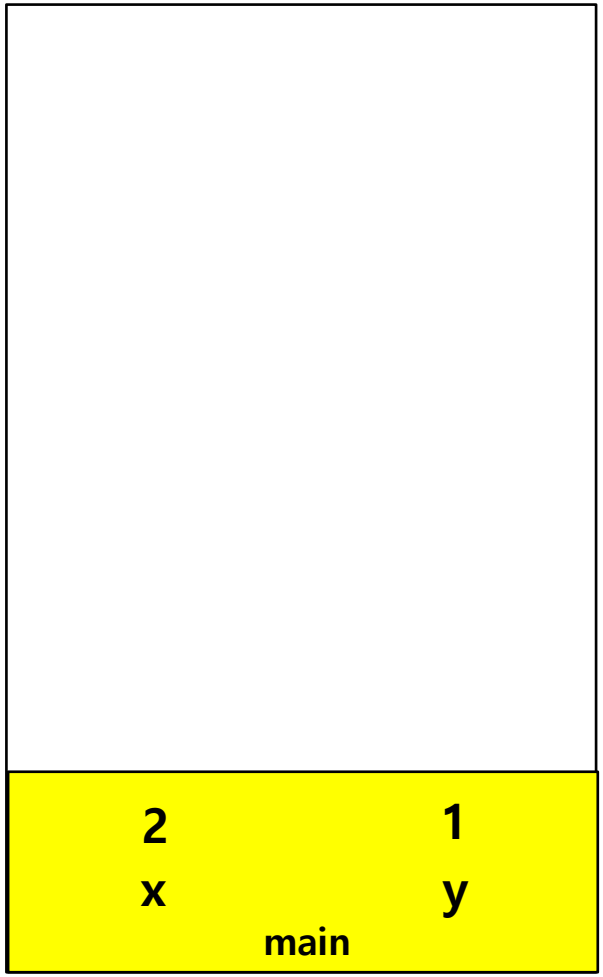

```
void main()
{
    int x = 1;
    int y = 2;
    swap(&x, &y)
    printf("x = %d, y = %d\n", x, y);
}

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

memory 와 프로그램




```
void main()
{
    int x = 1;
    int y = 2;
    swap(&x, &y)
    printf("x = %d, y = %d\n", x, y);
}

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

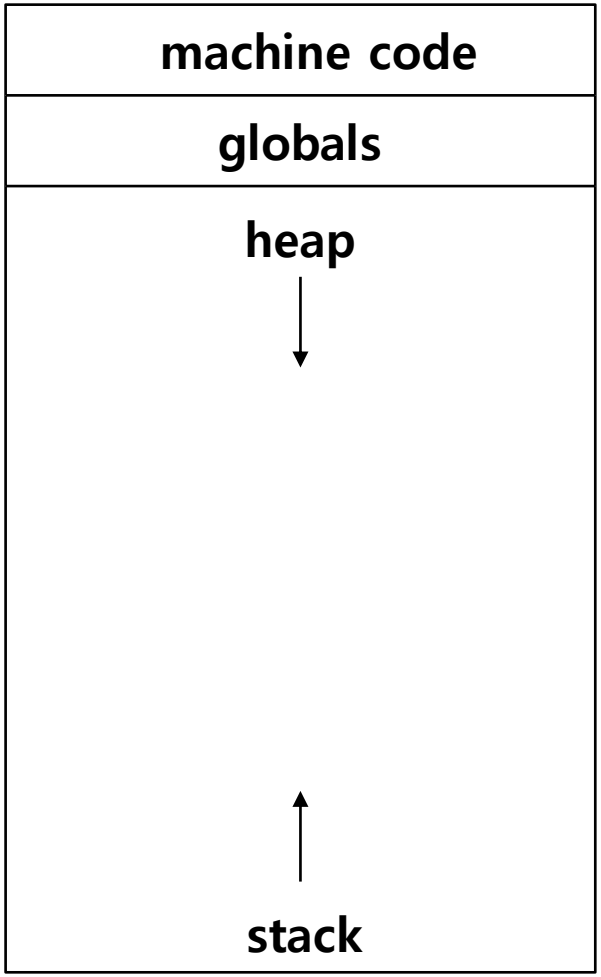
memory 와 프로그램




```
void main()
{
    int x = 1;
    int y = 2;
    swap(&x, &y)
    printf("x = %d, y = %d\n", x, y);
}

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

출력 결과: x = 2, y = 1

memory 와 프로그램



- heap overflow
- stack overflow