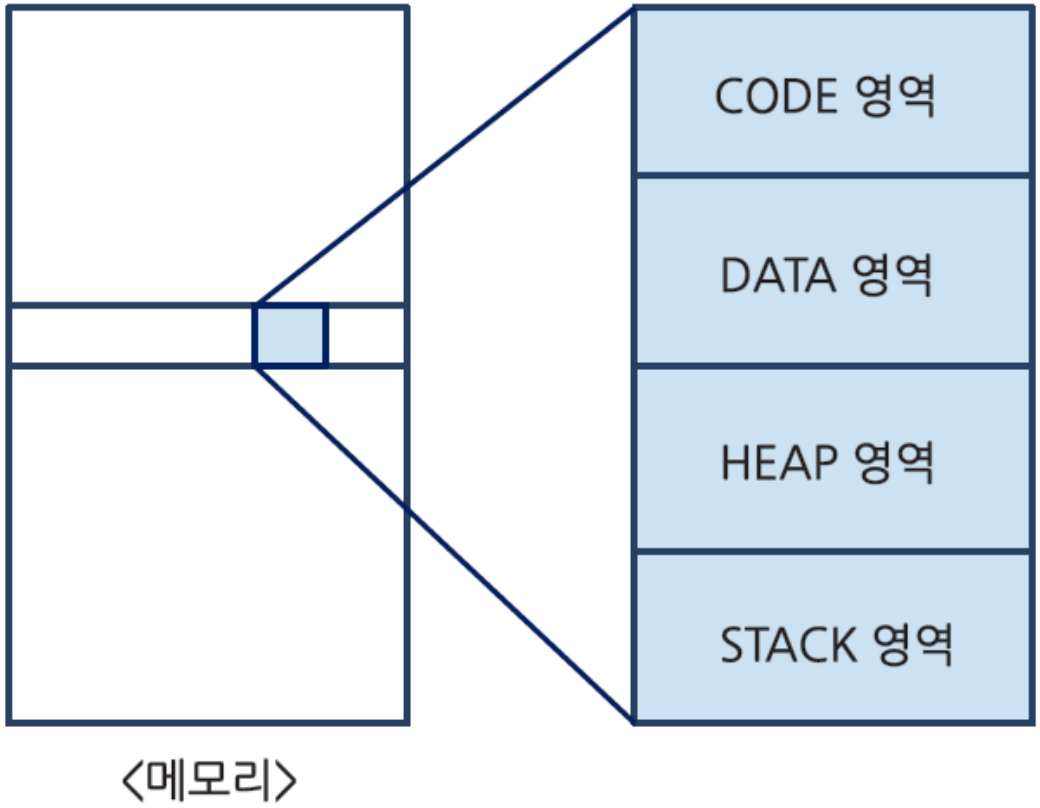
KG 아이티뱅크

C언어

VISUAL STUDIO

동적할당

동적할당


CODE 영역
DATA 영역
HEAP 영역
STACK 영역
실행할 프로그램의 코드
전역변수, static변수
동적할당
지역변수, 매개변수
<메모리>

동적할당

정적할당

- Compile(컴파일) 시 할당될 메모리 크기가 결정
- Stack과 Data영역에 할당되는 메모리

동적할당

- 실행중에 메모리 할당
- 동적할당된 메모리는 Heap영역에 할당

동적할당 사용 예

- 특정 영역에서 한번만 사용하는 변수를 정적할당하게 되면 함수가 만들어질 때 할당되고 함수가 끝날 때 해제 되기 때문에 불필요한 메모리 공간을 차지한다
- 사용할 변수 공간의 크기를 컴파일 하는 시점에 정할 수 없는 경우

지역변수

<파일이름 : 01.지역변수.c>

```
#include<stdio.h>
void func_A(void);
int func_B(void);

void main(void)
{
	int aa = 10;
	printf("함수 호출 전 main()함수의 aa 값 : %d₩n", aa);
	func_A();
	printf("func_A() 함수 호출 후 main()함수의 aa 값 : %d₩n", aa);		//aa는 메인함수의 지역변수
	printf("func_B() 함수 호출 후 main()함수의 aa 값 : %d₩n", func_B());//func_B()함수의 리턴 값 aa
}
void func_A()
{
	int aa = 20;
	int bb = 30;
	printf("func_A()함수의 aa값 : %d₩n", aa);
	printf("func_A()함수의 bb값 : %d₩n", bb);
}
int func_B()
{
	int aa = 35;		//func_B()함수의 지역변수 aa 선언
	return aa;		//aa는 35로 리턴
}
```

지역변수 : 각 함수가 끝나면 사라지는 변수

지역변수

<파일이름 : 02.지역변수.c>

```
#include<stdio.h>

void main()
{
	for (int i = 1; i < 3; i++)//for반복문 내부의 지역변수 total선언
	{
		int total = 0;
		total += i;
		printf("for의 total값은 %d입니다.₩n", total);
	}
	if (total < 10)
	{
		printf("if의 total값은 %d입니다.₩n", total);
	}
}
```

for함수 안에서 total을 선언하면 for문이 끝나는 순간 없어짐

전역변수

<파일이름 : 03.전역변수.c>

```
#include<stdio.h>
int num;  //전역변수 선언, 초기화하지 않아도 0이 된다.
void grow();

void main()
{
	printf("함수 호출 전 num의 값 : %d₩n", num);
	grow();
	printf("함수 호출 후 num의 값 : %d₩n", num);
}
void grow()
{
	num = 60;
	printf("grow 함수 내부 num의 값 : %d₩n", num);
}
```

전역변수 선언 시 전 지역에서 사용 가능

정적변수

<파일이름 : 04.정적변수.c>

```
#include<stdio.h>

void sv() {
	static int a = 10;//정적변수, 함수 실행 시 단 한번만 초기화(계속 축적)
	int b = 10;		//지역변수, 함수 실행 시 매번 초기화
	a++;
	b++;
	printf("static 변수 a의 값 : %d, 지역변수 b의값 : %d₩n", a, b);

}

void main()
{
	int i;
	for (i = 0; i < 5; i++) {
		sv();
	}
	//printf("%d", a);//정적변수는 함수내에서만 사용가능
}
```

정적변수는 단 한번만 초기화가 되고 함수 내에서는 안 사라짐
하지만 전역변수처럼 다른 함수에도 남아있는게 아니라 그 함수안에서만 존재

정적변수

<파일이름 : 05.정적변수.c>

```
#include<stdio.h>

void count();
void main()
{
	int num = 0;
	while (num < 3)
	{
		count();
		num++;
	}
}
void count(void)
{
	static int x = 0;
	int y = 0;
	x += 1;
	y += 1;
	printf("x값 : %d, y값 : %d₩n", x, y);
}
```

동적할당

malloc 함수란?

- heap 영역에 동적으로 공간을 할당 해 주는 함수
- Header File : <stdlib.h>

형식

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

인자의 크기만큼(size_t size) 메모리를 할당하고 할당된 메모리의 시작주소를 return 한다
리턴되는 주소는 void 포인터형이기 때문에 반드시 형변환해서 사용해야 한다

malloc 함수의 반환형이 void인 이유

- malloc 함수는 인자로 단순히 숫자하나를 전달받기 때문에 할당하는 메모리의 자료형을 알지 못한다. 따라서 반환할 주소의 데이터자료형을 알지 못하기 때문에 형 변환을 통해 할당된 메모리 주소값을 저장하도록 한다

동적할당

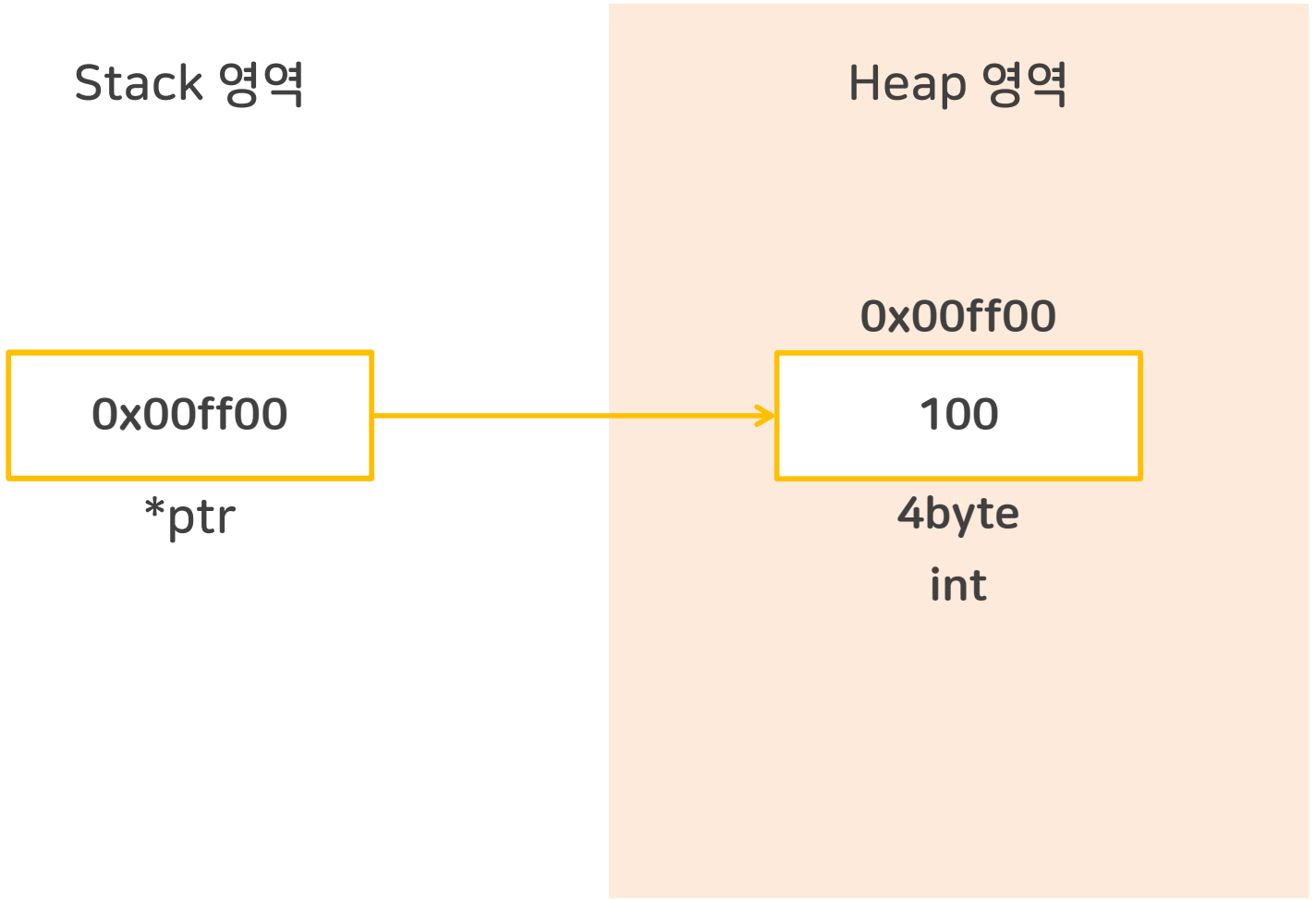
<파일이름 : 06.동적할당.c>

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

void main() {
	int* ptr = (int *)malloc(sizeof(int));
	*ptr = 100;

	printf("ptr이 가르키는 값 : %d₩n", *ptr);
	printf("ptr이 가르키는 공간의 크기 : %d₩n", sizeof(*ptr));
}
```

malloc(sizeof(int))는 heap영역에 int형 크기의 공간을 만들고 그 주소를 반환 -> 주소를 반환하기 때문에 반환형을 int*로 해줘야함

동적할당

<파일이름 : 06.동적할당.c>



동적할당

free 함수란?

- heap 영역에 동적으로 할당된 공간을 해제해 주는 함수
- Header File : <stdlib.h>

형식

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

- heap영역의 공간은 함수가 끝나도 사라지지 않는다
- malloc 함수를 통해 heap 영역의 공간을 할당해 사용한 후 사용이 끝나면 공간을 반드시 해제시켜 줘야 한다

동적할당

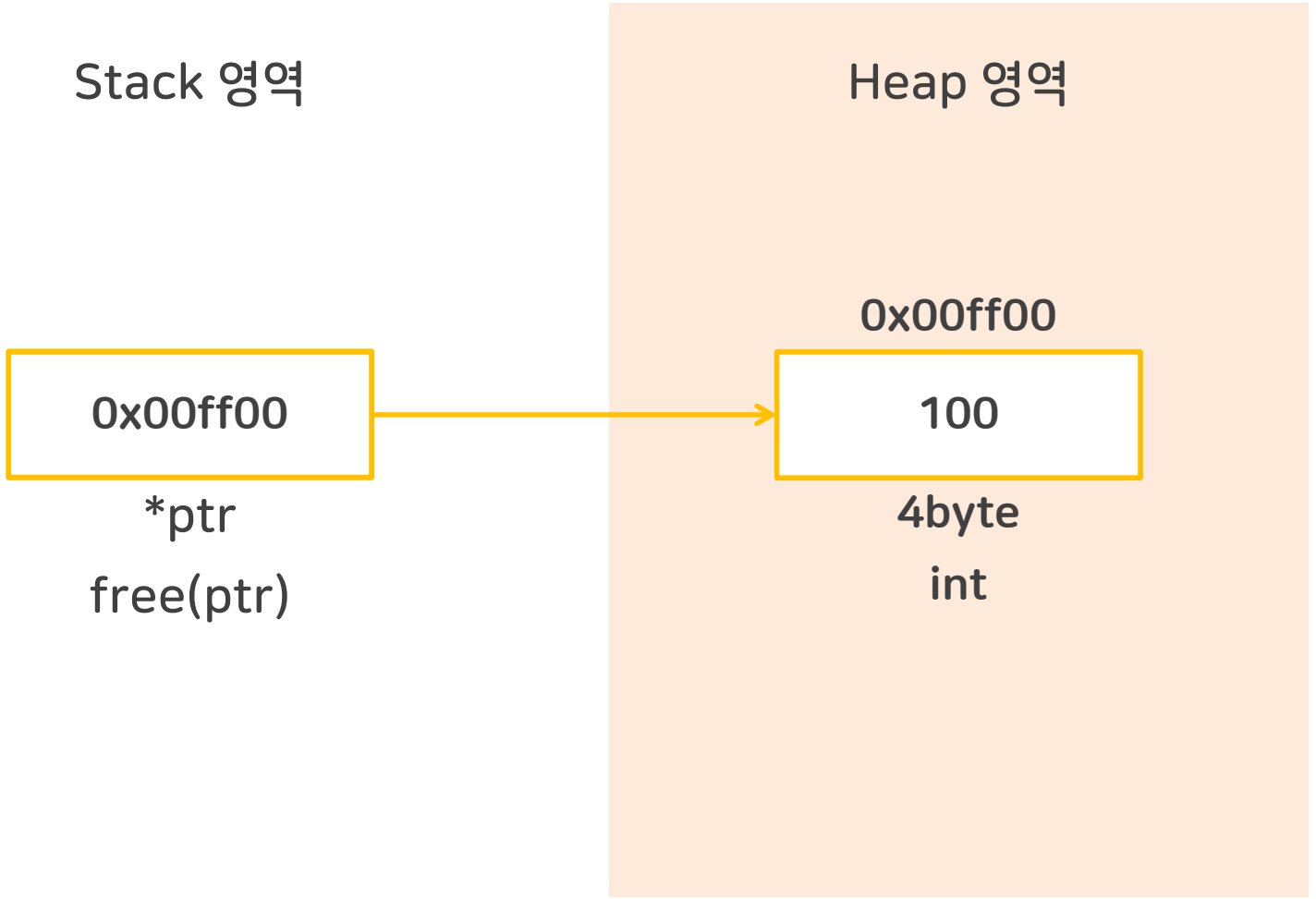
<파일이름 : 07.동적할당.c>

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

void main() {
	int* ptr = (int *)malloc(sizeof(int));
	*ptr = 100;

	printf("ptr이 가르키는 값 : %d₩n", *ptr);
	printf("ptr이 가르키는 공간의 크기 : %d₩n", sizeof(*ptr));

	free(ptr);
	printf("ptr이 가르키는 값 : %d₩n", *ptr);
	printf("ptr이 가르키는 공간의 크기 : %d₩n", sizeof(*ptr));
}
```

free로 공간을 해제하면 heap영역에 할당한 공간이 사라짐

동적할당

<파일이름 : 07.동적할당.c>



동적할당

<파일이름 : 08.동적할당.c>

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

void main()
{
	char* ptr;
	int count, i;

	printf("몇 개의 공간을 만들까요? ");
	scanf("%d", &count);

	ptr = (char*)malloc(sizeof(char)*count);

	for (i = 0; i<count; i++) {
		ptr[i] = i + 1;
		printf("%d₩n", ptr[i]);
	}
	free(ptr);
}
```

동적할당 할 사이즈에 원하는 공간의 개수를 곱하면 그 만큼 할당