

충북대학교 SW중심대학산업단

C언어 프로그래밍 기초 및 실습

박상수 강사

2020년 8월 14일

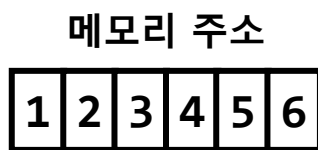
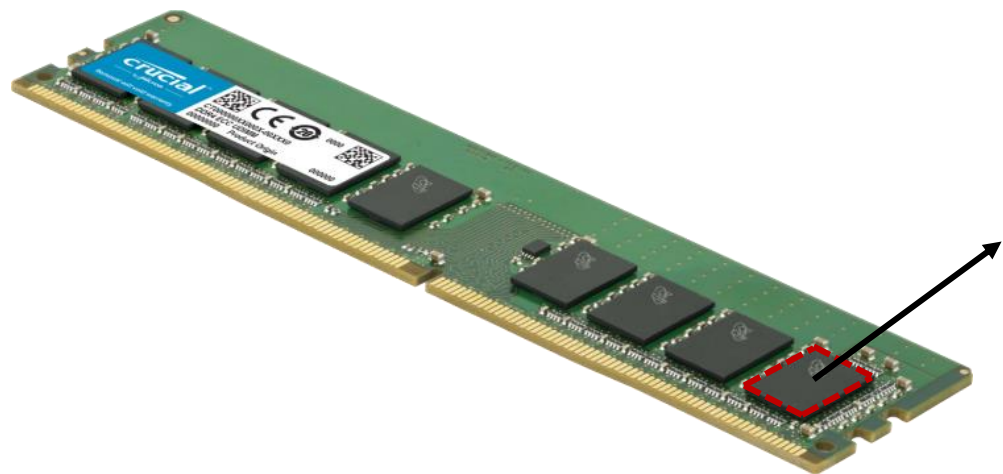
```
data[src_idx];  
data[src_idx + 1];  
data[src_idx + 2];  
dst[dst_idx + 0] = charset[(s0 & 0xfc) >> 2];  
dst[dst_idx + 1] = charset[((s0 & 0x03) << 4) | ((s1 & 0xf0) >> 4)];  
dst[dst_idx + 2] = charset[((s1 & 0x0f) << 2) | ((s2 & 0xf0) >> 4)];  
dst[dst_idx + 3] = charset[(s2 & 0x3f)];  
(src_idx < length)  
uint8_t s0 = data[src_idx];  
uint8_t s1 = (src_idx + 1 < length) ? data[src_idx + 1] : 0;
```

강의 목차

- 포인터
 - 포인터 개념/덧셈, 다차원 배열을 위한 포인터
- 동적 메모리 할당
 - 메모리 할당 및 초기화
- 연습문제
 - Call-by-reference를 사용한 프로그램

변수가 컴퓨터 메모리에 저장되는 방법

- 모든 데이터들은 메모리 상에 특정한 공간에 저장
 - 특정한 공간은 고유의 주소 (address)가 존재
 - 변수는 사용자가 특정한 공간의 주소를 모르더라도 쉽게 사용 가능
 - 컴파일러에 의해 특정 주소에 있는 데이터를 변경
 - **컴파일러**: 사람이 작성한 프로그래밍 언어를 기계가 이해할 수 있는 언어로 변경해 주는 프로그램



```
#include <stdio.h>
int main() {
    int a;
    a = 10; // 0x152839 위치의 데이터를 10으로 변경
    printf("a 의 값은 : %d \n", a);
    return 0;
}
```

변수가 컴퓨터에서 처리되는 과정

포인터 #1

■ 메모리 상에 위치한 특정 데이터가 저장된 주소 값을 보관하는 변수

- 포인터: 메모리 상에 위치한 특정한 데이터의 시작 주소 값을 보관하는 변수

(포인터에 주소값이 저장되는 데이터의 형) *(포인터의 이름);

- 포인터를 사용하여 **다른 객체에 접근하여 읽고 쓰거나 함수를 실행**
- 변수를 정의 할 때 데이터 형 (type)이 있던 것처럼 포인터에도 존재
- 포인터를 사용하기 위해서는 포인터 변수에 참조할 변수의 주소 값을 불러오는 과정 필요

```
/* 포인터의 시작 */
#include <stdio.h>
int main() {
    int *p; // int 형의 포인터 변수
    int a;  // int 형의 변수

    p = &a; // 포인터 변수 p에 변수 a의 주소 값 저장, & (단항 연산자)는 주소 값을 불러오는 기능

    printf("포인터 p 에 들어 있는 값 : %p \n", p); // %p (16진수 형태로 출력하는 포맷)
    printf("int 변수 a 가 저장된 주소 : %p \n", &a);

    return 0;
}
```

포인터 p 에 들어 있는 값 : 0x7fffe11ee5cc
int 변수 a 가 저장된 주소 : 0x7fffe11ee5cc

포인터 사용의 예: 주소 확인

포인터 #2

■ 포인터: 특정 데이터의 주소 값을 보관

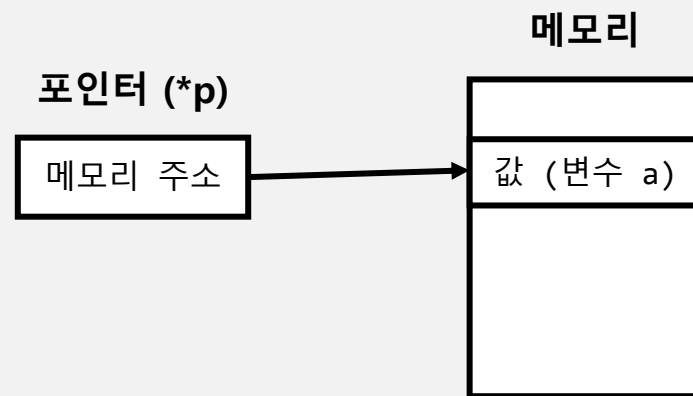
- 주소 값을 보관하는 데이터 형에 *을 붙임으로써 정의
 - & 연산자로 특정한 데이터의 메모리 상의 주소 값을 알 수 있음
 - 어떠한 데이터의 주소 값을 알아냈다면, 주소 값에 대응되는 데이터를 가져오는 연산자 필요 (*)

```
/* * 연산자의 이용 */
#include <stdio.h>
int main() {
    int *p;
    int a;

    p = &a; // 포인터 변수에 a 변수의 주소 값 저장
    a = 2;  // a 변수에 값 저장

    printf("a 의 값 : %d \n", a);
    printf("*p 의 값 : %d \n", *p);

    return 0;
}
```



a 의 값 : 2
*p 의 값 : 2

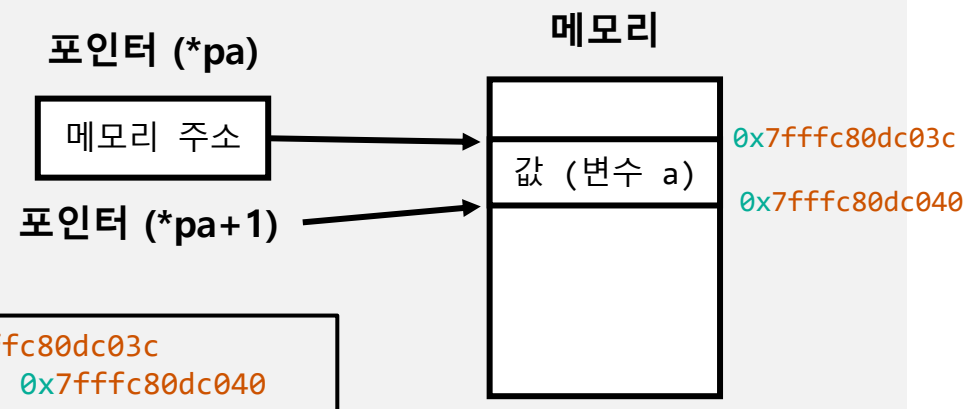
포인터 사용의 예: 포인터가 가리키는 변수의 값 확인

포인터 #3

■ 포인터 덧셈과 뺄셈

- 포인터 변수에 덧셈/뺄셈 연산을 통해 주소 값 변경 가능
 - pa와 pa+1의 값 차이는 4 (byte)
 - 포인터 변수에 1을 덧셈, 변수의 데이터 타입이 **int** (4byte)이기 때문에 4byte 증가

```
/* 포인터의 덧셈 */  
#include <stdio.h>  
int main() {  
    int a;  
    int* pa;  
    pa = &a;  
  
    printf("pa 의 값 : %p \n", pa);  
    printf("(pa + 1) 의 값 : %p \n", pa + 1);  
  
    return 0;  
}
```



포인터 덧셈의 예: 덧셈 연산을 통해 포인터가 가리키는 주소 변경

포인터 실습 #1

■ 다양한 데이터 포맷의 포인터 덧셈/뺄셈의 결과는 ?

- **int** (정수형 데이터, 4byte), **char** (문자 데이터, 1byte), **double** (소수점 데이터, 8byte)
 - 덧셈은 주소가 데이터 타입의 크기 만큼 증가, 그렇다면 뺄셈은 데이터 타입 만큼 감소 ?

```
#include <stdio.h>
int main() {
    int a;
    char b;
    double c;
    int* pa = &a;
    char* pb = &b;
    double* pc = &c;

    printf("pa 의 값 : %p \n", pa);
    printf("(pa + 1) 의 값 : %p \n", pa + 1);
    printf("pb 의 값 : %p \n", pb);
    printf("(pb + 1) 의 값 : %p \n", pb + 1);
    printf("pc 의 값 : %p \n", pc);
    printf("(pc - 1) 의 값 : %p \n", pc - 1);

    return 0;
}
```

```
pa 의 값 : 0x7ffca01c9494
(pa + 1) 의 값 : 0x7ffca01c9498
pb 의 값 : 0x7ffca01c9493
(pb + 1) 의 값 : 0x7ffca01c9494
pc 의 값 : 0x7ffca01c9498
(pc - 1) 의 값 : 0x7ffca01c9490
```

포인터 덧셈/뺄셈의 예: 다양한 데이터 타입

포인터 #4

■ 배열과 포인터

- 배열: 많은 변수를 하나의 집합처럼 사용하는 방법
 - 배열들의 각 원소는 메모리 상에 연속된 주소에 위치
 - `arr[0]`과 `arr[1]`의 주소 값 차이는 4 (byte)
 - 포인터 변수를 사용하여 쉽게 배열의 각 원소의 값을 수정 가능 !

```
#include <stdio.h>
int main() {
    int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int* parr;
    int i;
    parr = &arr[0];

    for (i = 0; i < 10; i++) {
        printf("arr[%d] 의 주소값 : %p ", i, &arr[i]);
        printf("(parr + %d) 의 값 : %p ", i, (parr + i));

        if (&arr[i] == (parr + i)) {
            /* 만일 (parr + i) 가 성공적으로 arr[i] 를 가리킨다면 */
            printf(" --> 일치 \n");
        } else {
            printf(" --> 불일치\n");
        }
    }
    return 0;
}
```

arr[0] 의 주소값 : 0x7fff28d0f720	(parr + 0) 의 값 : 0x7fff28d0f720	--> 일치
arr[1] 의 주소값 : 0x7fff28d0f724	(parr + 1) 의 값 : 0x7fff28d0f724	--> 일치
arr[2] 의 주소값 : 0x7fff28d0f728	(parr + 2) 의 값 : 0x7fff28d0f728	--> 일치
arr[3] 의 주소값 : 0x7fff28d0f72c	(parr + 3) 의 값 : 0x7fff28d0f72c	--> 일치
arr[4] 의 주소값 : 0x7fff28d0f730	(parr + 4) 의 값 : 0x7fff28d0f730	--> 일치
arr[5] 의 주소값 : 0x7fff28d0f734	(parr + 5) 의 값 : 0x7fff28d0f734	--> 일치
arr[6] 의 주소값 : 0x7fff28d0f738	(parr + 6) 의 값 : 0x7fff28d0f738	--> 일치
arr[7] 의 주소값 : 0x7fff28d0f73c	(parr + 7) 의 값 : 0x7fff28d0f73c	--> 일치
arr[8] 의 주소값 : 0x7fff28d0f740	(parr + 8) 의 값 : 0x7fff28d0f740	--> 일치
arr[9] 의 주소값 : 0x7fff28d0f744	(parr + 9) 의 값 : 0x7fff28d0f744	--> 일치

포인터를 사용한 행렬의 값 제어

포인터 실습 #2

- 다음의 소스 코드를 완성하여 각 포인터가 가리키는 주소 값의 $8_{(16)}$ 차이가 되도록 하시오
 - `int` (정수형 데이터, 4byte)을 고려하여 작성

```
#include <stdio.h>

int main()
{
    int array[10]={1,2,3,4,5,6,7,8,9,10};
    int* numPtrA = &array;

    printf("org is %p\n", numPtrA);
    printf("mod1 is %p\n", numPtrA+2);
    printf("mod2 is %p\n", _____);

    return 0;
}
```

```
org is 0x7fff5a616900
mod1 is 0x7fff5a616908
mod2 is 0x7fff5a616914
```

포인터 덧셈/뺄셈 실습

포인터 #5

■ 포인터를 사용해 1차원 배열 가리키기

- `parr = arr`
 - `arr`은 배열의 첫번째 원소를 가리키는 포인터로 변환
 - `parr[n]` 또는 `*(parr+n)`을 사용하여 포인터가 가리키는 주소의 값 확인 가능 !

```
#include <stdio.h>
int main() {
    int arr[3] = {1, 2, 3};
    int *parr;

    parr = arr;
    /* parr = &arr[0]; 동일! */

    printf("arr[1] : %d \n", arr[1]);
    printf("parr[1] : %d \n", parr[1]); // *(parr+1)과 동일한 결과
    return 0;
}
```

<code>arr[1]</code>	:	<code>2</code>
<code>parr[1]</code>	:	<code>2</code>

1차원 배열과 포인터

포인터 #5

■ 포인터를 사용해 1차원 배열 가리키기

- `parr = arr`
 - `arr`은 배열의 첫번째 원소를 가리키는 포인터로 변환
 - `parr[n]` 또는 `*(parr+n)`을 사용하여 포인터가 가리키는 주소의 값 확인 가능 !

```
#include <stdio.h>
int main() {
    int arr[3] = {1, 2, 3};
    int *parr;

    parr = arr;
    /* parr = &arr[0]; 동일! */

    printf("arr[1] : %d \n", arr[1]);
    printf("parr[1] : %d \n", parr[1]); // *(parr+1)과 동일한 결과
    return 0;
}
```

<code>arr[1]</code>	:	<code>2</code>
<code>parr[1]</code>	:	<code>2</code>


1차원 배열과 포인터

포인터 #5

■ 2차원 배열 포인터 메모리 주소

- 컴퓨터 메모리 구조는 1차원, 따라서 2차원 배열을 1차원 구조로 메모리 상에서 동작

• `arr[0][0]: 0x7ffebc238060`, `arr[0][2]: 0x7ffebc238068`, `arr[1][0]: 0x7ffebc23806c`



```
#include <stdio.h>
int main() {
    int arr[2][3];

    printf("arr[0] : %p \n", arr[0]);
    printf("&arr[0][0] : %p \n", &arr[0][0]);

    printf("arr[1] : %p \n", arr[1]);
    printf("&arr[1][0] : %p \n", &arr[1][0]);

    return 0;
}
```

			8068
8060	arr[0][0]	arr[0][1]	arr[0][2]
806c	arr[1][0]	arr[1][1]	arr[1][2]
	arr[2][0]	arr[2][1]	arr[2][2]

<code>arr[0]</code>	<code>: 0x7ffebc238060</code>
<code>&arr[0][0]</code>	<code>: 0x7ffebc238060</code>
<code>arr[1]</code>	<code>: 0x7ffebc23806c</code>
<code>&arr[1][0]</code>	<code>: 0x7ffebc23806c</code>

1차원 메모리에서 동작하는 2차원 배열

포인터 #5

■ 2차원 배열을 위한 포인터

- 2차원 배열을 포인터에서 사용하기 위해서는 **이중 포인터** 사용

자료형 (*포인터이름)[가로크기]

- 포인터를 선언할 때 *과 포인터 이름을 괄호로 묶어준 뒤 []에 가로 크기를 지정
- `int (*numPtr)[4];`
 - 가로 크기가 4인 배열을 가리키는 포인터

`int (*numPtr)[4]`



arr[0][0]	arr[0][1]	arr[0][2]	arr[0][3]
arr[1][0]	arr[1][1]	arr[1][2]	arr[1][3]
arr[2][0]	arr[2][1]	arr[2][2]	arr[2][3]

이중 포인터의 예

포인터 #6

■ 2차원 배열을 위한 포인터

■ `int (*numPtr)[4]`

자료형 (*포인터이름)[가로크기]

- 배열의 데이터에 접근하기 위해서는 **1차원 배열의 시작 주소**와 1차원 배열 중에서 **어떤 원소에 접근** 할 것인지 정보 필요
 - `numArr[1][2]` 주소의 값에 접근하기 위해서 `*(numPtr+2))[2]` 사용

```
#include <stdio.h>

int main()
{
    int numArr[3][4] = { // 세로 3, 가로 4 크기의 int형 2차원 배열 선언
        { 11, 22, 33, 44 },
        { 55, 66, 77, 88 },
        { 99, 110, 121, 132 }
    };

    int (*numPtr)[4] = numArr;

    printf("org is %d\n", numPtr[2][1]); // 110: 2차원 배열을 포인터로 접근하는 경우 직접 인덱스를 부여하거나
    printf("ptr is %d\n", (*(numPtr+2))[1]); // 110: 2차원 배열을 포인터를 사용하는 방식으로 접근 가능

    printf("org is %d\n", numPtr[1][3]); // 88: 2차원 배열을 포인터로 접근하는 경우 직접 인덱스를 부여하거나
    printf("ptr is %d\n", (*(numPtr+1))[3]); // 88: 2차원 배열을 포인터를 사용하는 방식으로 접근 가능

    return 0;
}
```

이중 포인터를 사용하여 2차원 행렬의 값을 출력하는 프로그램

포인터 연습문제

■ Call by value와 Call by reference

- 함수에서 인자를 전달받는 방법은 Call-by-value와 Call-by-reference 두 종류 존재
 - Call-by-value: 인자의 값을 직접 전달받는 방법 (값을 복사하여 처리)
 - Call-by-reference: 인자의 값을 주소로 전달받는 방법 (전달 받은 값이 위치한 주소에 접근)
 - 함수를 정의할 때, 인자 값에 *표시
 - 함수를 호출할 때, 인자 값에 주소 (&)로 전달해야 함

```
#include <stdio.h>

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

int main()
{
    int a = 20, b = 60;
    swap(a, b);
    printf("a: %d, b: %d", a, b);
}
```

a: 20, b: 60

```
#include <stdio.h>

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

void main()
{
    int a = 20, b = 60;
    swap(&a, &b);
    printf("a: %d, b: %d", a, b);
}
```

a: 60, b: 20

Call-by-value와 Call-by-reference의 차이

포인터 연습문제

■ Call by reference를 사용하여 최대값을 찾는 함수를 설계하시오

- 모든 인자를 Call-by-reference를 사용
- num1과 num2의 값을 비교하여 (If-else)
 - num1이 num2보다 크면 num3에 num1의 값을
 - 그렇지 않은 경우는 num3에 num2의 값을 저장

```
#include <stdio.h>

void find_max(int *num1, int *num2, int *num3)
{

}

void main()
{
    int a = 20, b = 60, c = 0;
    find_max(&a, &b);
    printf("a: %d, b: %d, c: %d ", a, b, c);
}
```

a: 60, b: 20, c: 60

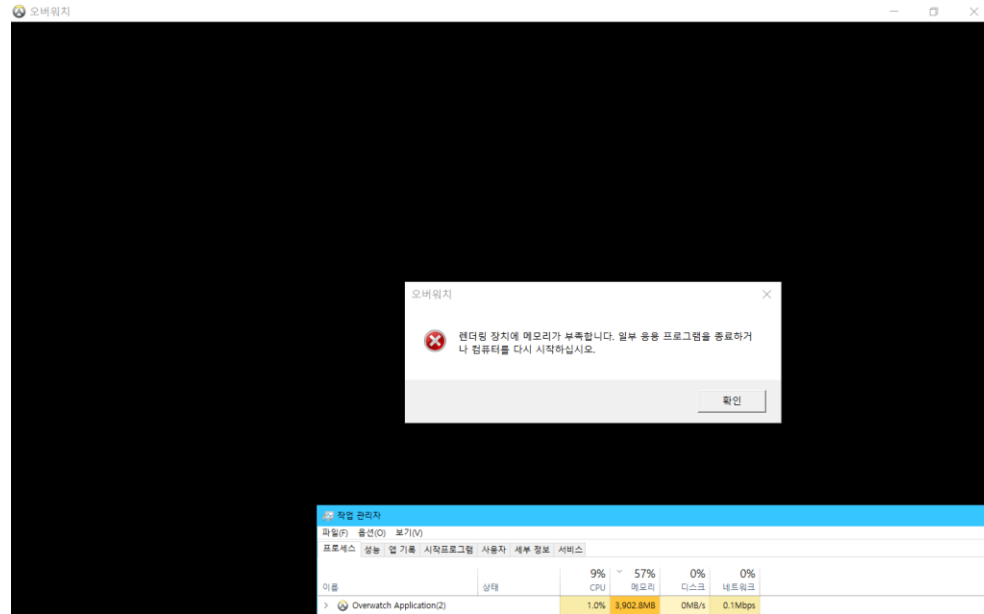
Call-by-reference를 사용하는 최대값을 찾는 프로그램

강의 목차

- 포인터
 - 포인터 개념/덧셈, 다차원 배열을 위한 포인터
- 동적 메모리 할당
 - 메모리 할당 및 초기화
- 연습문제
 - Call-by-reference를 사용한 프로그램

배열의 크기

- 배열의 크기는 컴파일 시간에 확정되어야 함
 - 컴파일러가 배열의 크기를 추측할 필요없이 명확하게 명시
 - 학생의 성적을 처리하는 프로그램을 설계할 때 ?
 - 학생과 과목의 숫자는 명확하게 알 수가 없음
 - 배열의 크기를 충분히 크게 잡게 되면 메모리 낭비 발생



컴퓨터에서 메모리가 부족한 경우

동적 메모리 할당 #1

■ 가변적으로 메모리의 크기를 변경 할 수 있는 동적 할당

■ malloc 함수

- <stdlib.h>에 정의되어 있기 때문에 추가 필요
- 인자로 전달된 크기의 바이트 수만큼 메모리 공간을 할당
 - sizeof(int): int 형 데이터의 크기, SizeOfArray: 할당한 데이터에 있는 원소의 개수
 - 자신이 할당한 메모리의 시작 주소를 리턴, 이때 (int *)형으로 변환하여 arr에 저장
 - 할당 받은 메모리를 사용하고 다시 컴퓨터에게 돌려주는 과정 필요 (free)

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

int main(int argc, char **argv) {
    int SizeOfArray;
    int *arr;

    printf("만들고 싶은 배열의 원소의 수 : ");
    scanf("%d", &SizeOfArray);

    arr = (int *)malloc(sizeof(int) * SizeOfArray);
    // int arr[SizeOfArray] 와 동일한 작업을 한 크기의 배열 생성

    free(arr);

    return 0;
}
```

동적 메모리 할당의 예: malloc

동적 메모리 할당 #2

■ 메모리의 내용을 원하는 크기만큼 특정 값을 설정하는 방법

- `void *memset(void *_Dst, int _Val, size_t _Size);`
 - `<string.h>`에 정의되어 있기 때문에 추가 필요
 - `_Dst`: 할당된 메모리 변수
 - `_Val`: 할당된 메모리 변수를 채울 값
 - `_Size`: 메모리 변수의 크기

```
#include <stdio.h>
#include <stdlib.h> // malloc, free 함수가 선언된 헤더 파일
#include <string.h> // memset 함수가 선언된 헤더 파일

int main()
{
    int *numPtr = malloc(sizeof(int)); // int의 크기 4바이트만큼 동적 메모리 할당
    memset(numPtr, 0x27, 4); // numPtr이 가리키는 메모리를 4바이트만큼 0x27로 설정

    printf("0x%llx\n", *numPtr); // 0x272727: 27이 4개 들어가 있음
    free(numPtr); // 동적으로 할당한 메모리 해제

    return 0;
}
```

동적 메모리 할당의 예: memset

동적 메모리 실습

■ 다음 소스 코드를 완성하여 9999 99999가 출력되도록 하세요

■ `void *malloc(size_t _Size)`

- `_Size`: 메모리 할당의 크기

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

int main()
{
    int *numPtr1 = _____;
    short *numPtr2 = _____;

    *numPtr1 = INT_MAX;
    *numPtr2 = LLONG_MAX;

    printf("%d %d\n", *numPtr1, *numPtr2);

    free(numPtr1);
    free(numPtr2);

    return 0;
}
```

동적 메모리 할당 실습문제

강의 목차

- 포인터
 - 포인터 개념/덧셈, 다차원 배열을 위한 포인터
- 동적 메모리 할당
 - 메모리 할당 및 초기화
- 연습문제
 - Call-by-reference를 사용한 프로그램

연습문제

- 2개 정수의 합과 차를 동시에 계산하는 프로그램을 작성하시오
 - 합과 차는 포인터 매개변수를 사용

```
#include <stdio.h>

int get_sum_diff (int x, int y, int *p_sum, int *p_diff)
{
    *p_sum = x+y;    // 합
    if(x>=y)         // 차이를 계산할 때, x가 y보다 큰 경우
        _____;
    else             // y가 x보다 큰 경우
        _____;
}

int main()
{
    int a=1000;
    int b=300;
    int sum, diff;

    get_sum_diff(a,b,&sum, &diff);

    printf("Diff is %d, Sum is %d\n", sum, diff);
    return 0;
}
```

감사합니다
