7장. 포인터

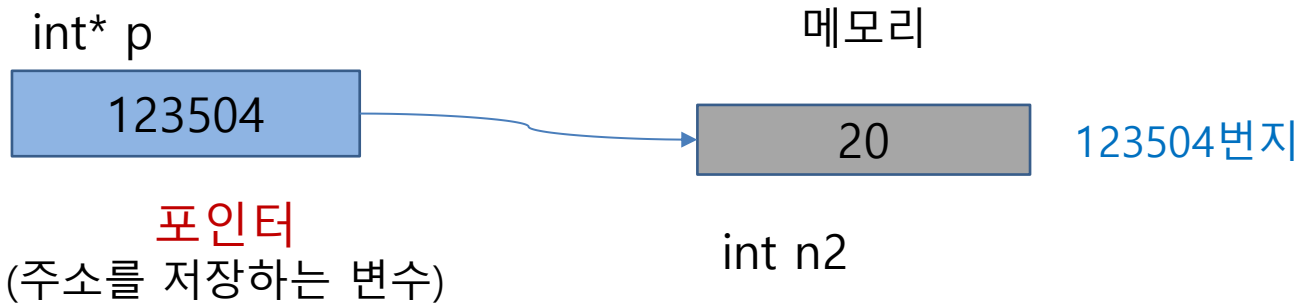
Visualstudio 2019

포인터(Pointer)

- 포인터?

  모든 메모리는 주소(address)를 갖는다. 이러한 메모리 주소를 저장하기 위해 사용되는 변수를 포인터 변수라 한다. 간단히 포인터(pointer)라고 부른다.

  (예) 택배 주소만 있으면 집을 찾을 수 있다.



포인터(Pointer)

- 포인터 변수의 선언 및 값 저장
  - 선언

자료형* 포인터 이름

```
char*  a;      // char형 포인터 변수
int* b;        // int형 포인터
double* c;     // double형 포인터 변수
```

  - 포인터의 크기 – 모든 자료형에서 4바이트로 동일하다.

sizeof(포인터)

포인터(Pointer)

- 포인터 변수의 선언 및 값 저장

```
char ch = 'J';
int iNum = 10;

char* cp;  //문자형 포인터 cp 선언
int* ip;   //정수형 포인터 ip 선언

cp = &ch;   //포인터에 ch의 주소 저장
ip = &iNum; //포인터에 iNum의 주소 저장

//*cp는 cp가 가리키고 있는 곳(ch의 주소)의 값
//*ip는 ip가 가리키고 있는 곳(iNum의 주소)의 값
printf("%x %c %d\n", cp, *cp, sizeof(cp));
printf("%x %d %d\n", ip, *ip, sizeof(ip));
```

포인터(Pointer)

- 역참조 연산자(*)

  포인터를 선언할 때도 *를 사용하고, 역참조 할때도 *를 사용

  - 포인터는 변수의 주소만 가리킴

  - 역참조는 주소에 접근하여 값을 가져옴

포인터(Pointer)

- 정수형 포인터 변수

```
int data = 10;
//int* ptr;
//ptr = &data;

int* ptr = &data;

printf("%x, %d₩n", &data, data);
printf("%x, %x₩n", &ptr, ptr);
printf("%d, %d₩n", *ptr, data);

printf("== 포인터 연산 ==₩n");
printf("%x %d %d₩n", &data, data, *ptr);
printf("%x %d %d₩n", &data + 1, data + 1, *ptr + 1);
printf("%x %d %d₩n", &data + 2, data + 2, *ptr + 2);
```

```
7af758, 10
7af74c, 7af758
10, 10
== 포인터 연산 ==
7af758 10 10
7af75c 11 11
7af760 12 12
```

포인터(Pointer)

- 문자형 포인터 변수

```
char a = 'A';
printf("a의 값은 %c₩n", a);

char *b;
b = &a;
printf("포인터 b의 값은 %c₩n", *b);

*b = 'B';  //변경
```

역참조 연산자는 값을 변경할 수 있음

```
printf("a의 값은 %c₩n", a);  //'B'
printf("b의 값은 %c₩n", *b); //'B'
printf("a의 주소 값은 %x₩n", &a);
printf("b의 주소 값은 %x₩n", &b);
```

값 & 참조에 의한 호출

- Call-by-value(값에 의한 호출) vs Call-by-reference(참조에 의한 호출)

값을 매개로 함수호출

CallByVal(int i)

↓

CallByVal(n)

주소를 매개로 함수 호출

CallByRef(int *i)

↓

CallbyRef(&n)

값 & 참조에 의한 호출

- Call-by-value(값에 의한 호출) vs Call-by-reference(참조에 의한 호출)

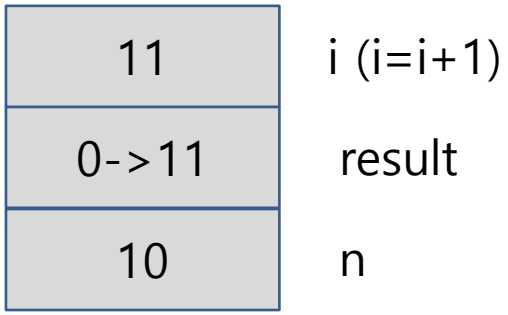
```c
int CallByVal(int);
int CallByRef(int*);

int main() {
    int n = 10;
    int result = 0;

    printf("--Call By Value--\n");
    result = CallByVal(n);
    printf("result = %d\n", result);
    printf("n = %d\n", n);

    printf("--Call By Reference--\n");
    result = CallByRef(&n);
    printf("result = %d\n", result);
    printf("n = %d\n", n);

    return 0;
}
```

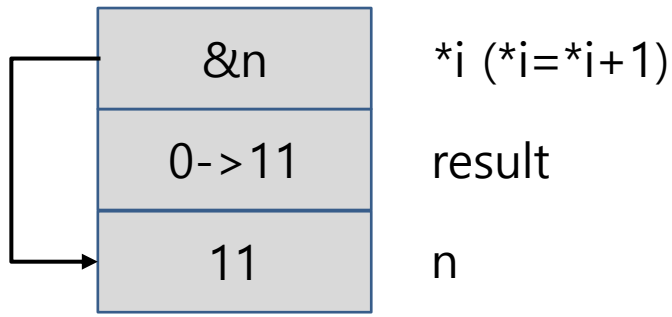
```c
int CallByVal(int i) {
    i = i + 1;
    return i;
}

int CallByRef(int* i) {
    *i = *i + 1;
    return *i;
}
```

```
--Call By Value--
result = 11
n = 10
--Call By Reference--
result = 11
n = 11
```

값 & 참조에 의한 호출

- Call-by-value(값에 의한 호출) vs Call-by-reference(참조에 의한 호출)





값 & 참조에 의한 호출

- Call-by-value(값에 의한 호출) vs Call-by-reference(참조에 의한 호출)

```c
void swapByVal(int, int);
void swapByRef(int*, int*);
int main() {
    int a = 1, b = 2;

    printf("==값에 의한 호출==\n");
    swapByVal(a, b);
    printf("a = %d, b = %d\n", a, b);

    printf("==주소에 의한 호출==\n");
    swapByRef(&a, &b);
    printf("a = %d, b = %d\n", a, b);

    return 0;
}
```

```c
void swapByVal(int x, int y) {
    int temp;
    temp = x;
    x = y;
    y = temp;
}

void swapByRef(int* x, int* y) {
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

```
==값에 의한 호출==
a = 1, b = 2
==주소에 의한 호출==
a = 2, b = 1
```

포인터의 연산

- 포인터의 연산

```
int a = 10, b = 20;
int total;
double avg;

int *pa, *pb, *ptot;
double *pavg;
pa = &a;
pb = &b;
ptot = &total;
pavg = &avg;

*ptot = *pa + *pb;
*pavg = *ptot / 2.0;
printf("total = %d, avg = %3.1lf", *ptot, *pavg);
```

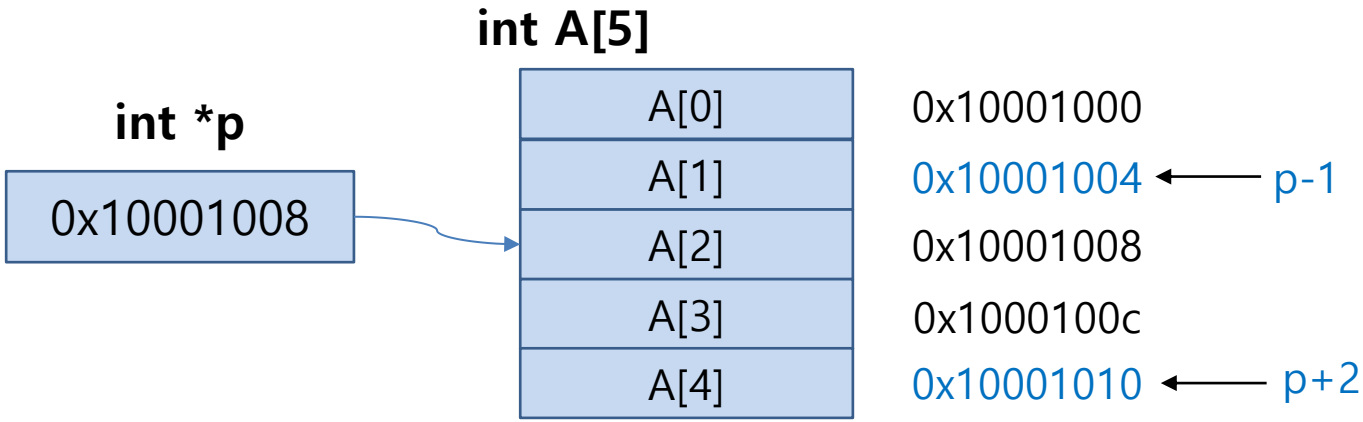
배열과 포인터

- 배열과 포인터

  - 배열은 데이터를 연속적으로 메모리에 저장한다.

    포인터 역시 메모리에 데이터를 저장하거나 저장된 데이터들을 읽어올수 있다.

```
int A[5], *p;        // 배열 A와 포인터 p선언
p = &A[2]            // p에 배열의 두 번째 항목 주소 복사
```



배열과 포인터(Pointer)

- 배열과 포인터의 연산(int형 포인터)

```
int a[5] = { 1, 2, 3, 4, 5 };
int i;

printf("주소:%d = %d₩n", a[0], &a[0]);
printf("주소:%d = %d₩n", a[1], &a[1]);
printf("배열 a = %d₩n", a);
printf("== 배열 a 출력 ==₩n");
for (i = 0; i < 5; i++) {
    printf("%2d", a[i]);
}
printf("₩n======================
```

```
//포인터 b에 배열 a 저장
int *b;
//b = &a[0];
b = a;

printf("a[0]의 주소값 = %d\n", b);
printf("a[0]의 저장값 = %d\n", a[0]);
printf("a[0]의 저장값 = %d\n", *b);

printf("== 포인터 배열 b 출력 ==\n");
for (i = 0; i < 5; i++) {
    printf("%2d", *(b + i));
}
//*b
//*(b+1)
//*(b+2)
```

포인터(Pointer)를 사용한 문자열 처리

◈ 포인터와 문자열

문자열은 문자들이 메모리 공간에 연속적으로 저장되어 있어서 주소로 관리되고, 문자열의 시작주소를 알면 모든 문자열에 접근할 수 있다.

```
char* p = "ABCD";
char* c = "C-language";
char* array[2] = { "Good morning", "Thank you" };

printf("%s \n", p);
printf("%s \n", p + 1);
printf("%s \n", p + 2);
printf("%s \n", p + 3);
printf("------------\n");
```

```
ABCD
BCD
CD
D
------------
C-language
language
------------
```

배열과 포인터(Pointer)

- 배열과 포인터

```c
#include <stdio.h>
#define _CRT_SECURE_NO_WARNINGS

int main() {
    char a[10];
    char* b;

    printf("문자를 입력하세요: ");
    scanf("%s", a);  //배열 첫번째 요소의 주소값
    //scanf_s("%s", a, sizeof(a));  //배열 첫번째 요소의 주소값
    //printf("저장된 문자열 : %s\n", a);

    b = a;
    printf("저장된 문자열 : %s\n", b);

    return 0;
}
```

포인터(Pointer)

- 함수의 매개변수로 포인터 사용하기

```c
void print(int*);
int main() {

    int num[5] = { 1, 2, 3, 4, 5 };
    print(num);

    return 0;
}

void print(int* num) {
    int i;
    for (i = 0; i < 5; i++) {
        printf("%2d", *(num + i));
    }
}
```

포인터와 배열

- 함수에서 포인터 활용

```c
void sayHello(char[]);
void sayHello2(char*);
int main() {

    sayHello("sunny");
    sayHello2("hyunsoo");

    return 0;
}

void sayHello(char name[]) {
    printf("Hello~ %s\n", name);
}

void sayHello2(char* name) {
    printf("Hello~ %s\n", name);
}
```

포인터(Pointer) 배열

- 포인터 배열에서 최대값과 최대값 위치 구하기

```c
int findMax(int*, int*);
int findMaxIdx(int*, int*);

int main() {
    //최대값과 최대값의 위치 찾기
    int arr[] = { 21, 35, 71, 2, 97, 66 };

    int max = findMax(arr, 6);  //최대값
    int maxIdx = findMaxIdx(arr, 6); //최대값의 위치

    printf("최대값 : %d\n", max);
    printf("최대값의 위치 : %d\n", maxIdx);

    return 0;
}
```

포인터(Pointer) 배열

- 포인터 배열에서 최대값과 최대값 위치 구하기

```
int findMax(int* arr, int* len) {
    int maxVal = *arr;
    int i;

    for (i = 1; i < len; i++) {
        if (maxVal < *(arr + i))
            maxVal = *(arr + i);
    }
    return maxVal;
}
```

```
int findMaxIdx(int* arr, int* len) {
    int maxIdx = 0;  //0번 인덱스
    int i;

    for (i = 1; i < len; i++) {
        if (*(arr + maxIdx) < *(arr + i))
            maxIdx = i;
    }
    return maxIdx;
}
```

포인터와 배열

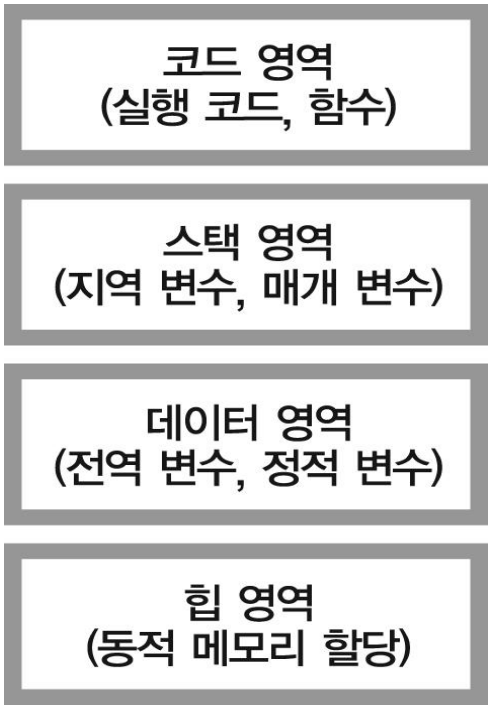
- switch ~ case문에서 포인터 활용

```c
int ranking = 3;
char *medalColor;

switch (ranking) {
case 1:
    medalColor = "Gold";
    break;
case 2:
    medalColor = "Silver";
    break;
case 3:
    medalColor = "Bronze";
    break;
default :
    medalColor = "None";
    break;
}
printf("%d등 메달의 색상은 %s입니다.\n", ranking, medalColor);
```

동적 메모리 할당

◈ 동적 메모리 할당의 필요성

동적 메모리 할당은 컴파일 중이 아닌 런타임 중 즉, 실행 시간에 이루어지는 데 힙 영역에 메모리가 할당 된다.




```
int a = 10;

int main() {
    int num1 = 10, num2 = 20;
    static int s = 30;

    printf("코드 영역 : %x %x ₩n", main, printf);
    printf("스택 영역 : %x %x ₩n", &num1, &num2);
    printf("데이터 영역 : %x %x ₩n", &a, &s);

    return 0;
}
```

동적 메모리 할당

◈ 동적 메모리 할당 - 배열을 사용하는 경우 문제점 해결 가능

▶ 선언된 배열 요소의 수가 사용된 요소 수보다 많은 경우 메모리의 낭비 발생

예) int arr[10];
arr[0], arr[1], arr[2],

▶ 선언된 배열 요소의 수가 사용된 요소 수보다 적은 경우 메모리의 부족 에러 발생

예) int arr[2];
arr[0], arr[1], arr[2],

동적 메모리 할당

◈ malloc 함수와 free 함수

- malloc() : 호출 성공시 메모리의 시작 주소를 반환하고, 호출 실패시 NULL을 반환함

  void* malloc(size_t size)

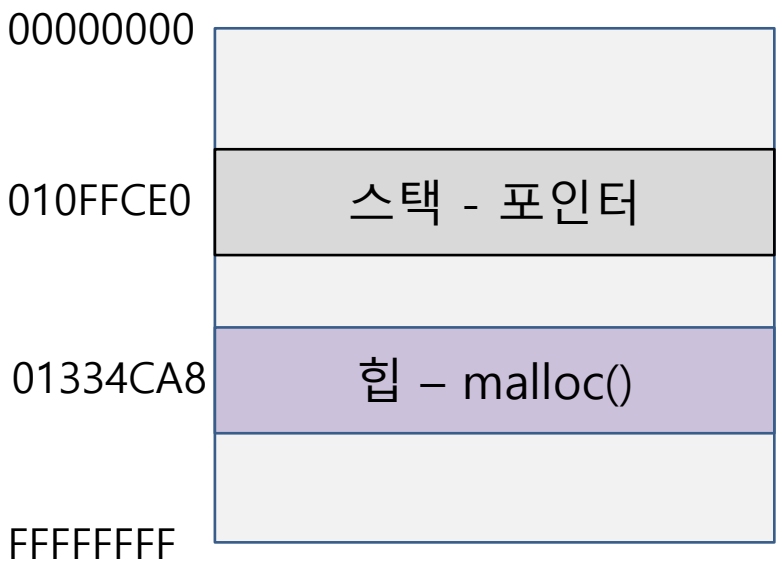
- free() : 동적 메모리 할당 공간을 해제함

  void free(void* p)

- <stdlib.h>에 정의되어 있음

동적 메모리 할당

```c
int num1 = 11;
int* numPtr1;    //스택 영역에 위치함
int* numPtr2;    //힙 영역에 위치함

numPtr1 = &num1;

numPtr2 = (int*)malloc(sizeof(int));
if (numPtr2 == NULL) {
    printf("동적 메모리 할당에 실패했습니다.\n");
    exit(1);   //강제 종료
}

printf("%p\n", numPtr1);
printf("%p\n", numPtr2);

free(numPtr2);
```



동적 메모리 할당 예제

◈ 정수형 포인터 ip에 5개 동적 메모리 할당

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

int main() {
    int* ip;
    int i;

    ip = (int*)malloc(sizeof(int) * 4);
    if (ip == NULL) {
        printf("동적 메모리 할당에 실패했습니다
        exit(1);
    }
    /* 배열로 저장하기
    ip[0] = 10;
    ip[1] = 20;
    ip[2] = 30;
    ip[3] = 40;*/
    //역참조로 저장하기
    *ip = 5;
    *(ip + 1) = 6;
    *(ip + 2) = 7;
    *(ip + 3) = 8;

    for (i = 0; i < 4; i++) {
        printf("%d %d\n", ip[i], *(ip + i));
    }

    free(ip);

    return 0;
}
```

동적 메모리 할당 예제

◈ 문자형 포인터 pc에 26바이트의 메모리를 할당

```
char *pc;
int i;
pc = (char *)malloc(sizeof(char) * 100);
if (pc == NULL) {
    printf("동적 메모리 할당에 실패했습니다. \n");
    exit(1);
}
for (i = 0; i < 26; i++) {
    *(pc + i) = 'a' + i;
}
*(pc + i) = 0;
printf("%s", pc);

free(pc);
```

동적 메모리 할당 예제

◈ 함수에서 동적 메모리 할당하기

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

//char name[15];  // 전역변수
char* getName();
int main() {

    char* name1;
    char* name2;

    name1 = getName();
    name2 = getName();

    printf("Hi, %s\n", name1);
    printf("Hi, %s\n", name2);

    return 0;
}
```

```c
char* getName() {
    char* name = (char* )malloc(sizeof(char) * 15);
    //char name[15];  //지역 변수

    printf("당신의 이름을 입력하세요 : ");
    gets(name);

    return name;
}
/*배열을 지역변수와 전역변수로 사용했을 경우 오류 발생..
  변수의 생성과 소멸 시기를 프로그램 실행 중에 설정할수 있는
  동적 메모리 할당이 필요함 */
```