작성자 : 김기희 (cupid072@naver.com)

void 포인터

- 학습목표
  - void 포인터에 대해 설명할 수 있다.
  - void 포인터의 필요성을 설명할 수 있다.
  - void 포인터를 사용하여 자료형에 의존하지 않는 함수를 작성할 수 있다.

void 포인터

- void 포인터는 자료형이 정해지지 않은 포인터이다.

  ```
  void *ptr;
  ```

- 포인터를 참조하여 값을 대입하고자 하는 경우 해당 포인터의 자료형에 맞는 데이터를 입력해야만 하며 포인터 변수에 저장되는 주소 또한 자료형이 일치해야 한다.

  ```
  int a;
  int *ptr = &a;
  *ptr = 3.14;                    // ERROR : intptr이 가리키는 곳의 자료형은 int 형이다.
  ```

- void 포인터는 자료형이 정해지지 않았으므로 모든 타입의 주소를 저장할 수 있다.

  ```
  int a = 10;
  void *ptr = &a;
  ```

- void 포인터는 자료형이 정해지지 않았으므로 void 포인터를 사용한 참조가 불가능하다.

  ```
  *ptr = 3.14;                    // ERROR : ptr이 가리키는 곳의 자료형을 알 수 없다.
  ```

void 포인터의 필요성

- 자료형에 관계 없이 동일한 작업을 수행하는 프로그램을 작성하기 위해 void 포인터를 사용한다.

- 예1) 동적 메모리 할당 함수인 malloc(), calloc(), realloc() 함수는 할당된 메모리의 선두 번지의 주소 만을 반환한다. malloc(), calloc(), realloc() 함수는 모든 타입의 데이터를 저장할 수 있는 메모리 공간을 할당 받을 수 있다.

```
void* malloc(size_t size);
void* calloc(size_t count, size_t eltsize);
void* realloc(void* ptr, sizt_t newsize);
```

- 예2) memset() 함수는 첫 번째 인수로 전달받은 포인터가 가리키는 메모리 공간을 초기화 한다. 이때 memset() 함수는 자료형에 관계 없이 포인터가 가리키는 공간의 모든 바이트를 초기화 한다.

```
void memset(void* ptr, int value, size_t num);
```

void 포인터의 형 변환

- void 포인터는 자료형이 정해져 있지 않으므로 모든 자료형의 주소가 저장될 수 있다.

```
int ary[] = { 1, 2, 3, 4, 5 };
int a = 10;
void* voidptr;
voidptr = &a;                    // int 형 주소 저장
voidptr = function;              // 함수의 주소 저장
voidptr = ary;                   // 배열의 저장
```

- void 포인터는 자료형을 알 수 없으므로 참조가 불가능하다. 적당한 타입으로 형 변환(cast) 하여 참조하여야 한다.

```
*(int *)voidptr = 100;           // int 형 포인터로 형변환 후 대입
((void (*)())voidptr)();         // 함수 포인터로 형 변환 후 호출
((int *)voidptr)[0] = 10;        // int 배열로 형 변환 후 대입
```

2차원 배열의 형변환은?

- 아래의 2차원 배열을 가리키는 void 포인터의 값을 출력하기 위한 형변환을 기술해 보세요.

```
int ary[][3] = { { 1, 2, 3 }, { 4, 5, 6 }, {7, 8, 9 } };
void* ptr = ary;
```

- void 포인터를 사용할 때 아래와 같이 미리 형 변환 후 사용하면 보다 사용하기 쉬워진다.

```
int (*arr)[3] = (int(*)[3])ptr;
printf("%d\n", arr[1][1]);          또는      printf("%d\n", *(*(ptr + 1) + 1));
```

풀어보세요.

- [문제 1] 아래의 코드에서 선언된 함수를 정의해 보세요.

```
#include <stdio.h>
int intCompare(void* p1, void* p2);                                                          // int값 비교 함수
void intSwap(void* p1, void* p2);                                                            // int값 교환 함수
void intPrinter(void* p);                                                                    // int값 출력 함수
void print(void* ptr, int size, int count, void (*printer)(void*));                          // 배열 내의 모든 원소 출력 함수
void sort(void* ptr,  int size,  int count,  int(*compare)(void*, void*),  void (*swap)(void*, void*));   // 배열을 정렬하는 함수

int main() {
    int intArray[] = { 3, 5, 1, 2, 4 };
    int intSize = (int)sizeof(intArray[0]);
    int intCount = (int)(sizeof(intArray) / sizeof(intArray[0]));
    print(intArray, intSize, intCount, intPrinter);
    sort(intArray, intSize, intCount, intCompare, intSwap);
    print(intArray, intSize, intCoiunt, intPrinter);
    return 0;
}
```