작성자 : 김기희 (cupid072@naver.com)

함수포인터

- 학습목표
  - 함수명에 대해 설명할 수 있다.
  - 함수 포인터가 필요한 이유를 기술할 수 있다.
  - 함수의 인수로 함수를 전달할 수 있다.
  - 함수의 반환 값으로 함수를 반환 받을 수 있다.
  - 동적 바인딩과 정적 바인딩의 차이를 설명할 수 있다.

함수 포인터

- 함수 포인터는 메모리상의 함수 코드의 위치를 가리키는 것

- 함수 명은 함수 코드가 저장되어 있는 주소를 가리키는 주소상수이다.

```
func();          // 함수의 호출
(*func)();       // 함수가 가리키는 곳의 코드를 호출
```

- 함수명이 함수 코드가 저장된 주소를 가리키는 주소 상수이므로 함수 또한 포인터 변수에 저장이 가능하다.

- 함수를 저장하기 위한 함수 포인터는 저장 가능한 함수 타입을 가져야 한다.

- 함수 포인터를 사용하여 함수 포인터에 저장된 함수의 호출이 가능하다.

함수 포인터의 필요성

- 함수의 전달 인자로 함수를 전달하고자 하는 경우
- 함수의 반환 값으로 함수를 반환하고자 하는 경우
- 프로그램 실행 중 동적으로 실행함수를 변경하고자 하는 경우

정적 링크(Static Link)와 동적 링크(Dynamic Link)

- 정적 링크(Static Link)란? 컴파일 타임에 호출될 함수가 결정되는 것을 말한다.

```
int sum(int a, int b);
int multiple(int a, int b);
.....
int result = 0;
if (oper == '+') result = sum(10, 20);          // 컴파일 타임에 sum() 또는 multiple() 함수 호출이 결정
else result = multiple(10, 20);
```

- 동적 링크(Dynamic Link)란? 런 타임(실행 시)에 호출될 함수가 결정되는 것을 말한다.

```
int sum(int a, int b);
int multiple(int a, int b);
.....
int (*func)(int, int) = NULL;
if (oper == '+') func = sum;
else func = multiple;
int result = func(10, 20);                      // 컴파일 타임에는 func 포인터에 의해 참조되는 함수를 알 수 없으며 런타임시 결정됨
```

함수 포인터 변수의 선언

- 함수 포인터 변수의 선언 형식

  함수반환형 (*함수포인터명)(전달인자타입...);

- 함수 포인터 변수 선언 예

```
int sum(int a, int b);
int (*ptr)(int, int);
ptr = sum;
```

함수를 파라미터로 하는 함수

```
int executor(int signal) {
    printf("시그널 %d를 수신하였습니다.\n", signal);
    return signal;
}

void handle_signal(int signal, int (*handler)(int)) {
    int result = handler(signal);
    printf("시그널 핸들러에 의해 처리된 값 : %d\n", result);
}

.....
handle_signal(9, executor);
```

함수 파라미터 예제 #1

```
void clear();                                                  // 입력스트림을 비우는 함수
void forEach(int* start, int* end, void(*worker)(int*));       // 함수를 파라미터로 가지는 함수의 선언
void outputNumber(int *ptr);                                   // 전달 인자로 사용될 함수
void inputNumber(int *ptr);                                    // 전달 인자로 사용될 함수

int main() {
    int ary[5] = { 0 };
    size_t size = sizeof(ary) / sizeof(ary[0]);
    forEach(ary, ary + size, inputNumber);                     // 전달 인자로 함수 사용
    forEach(ary, ary + size, outputNumber);                    // 전달 인자로 함수 사용
    getc(stdin);
    return 0;
}
```

함수 파라미터 예제 #1-1

```
void clear() {
	while (getc(stdin) != '\n');
}

void forEach(int* start, int* end, void(*worker)(int*)) {
	while (start < end) {
		worker(start);
		start++;
	}
}

void outputNumber(int *ptr) {
	printf("%d ", *ptr);
}
```

함수 파라미터 예제 #1-2

```
void inputNumber(int *ptr) {
    while (1) {
        printf("정수입력 : ");
        int inputCount = scanf("%d", ptr);
        if (inputCount != 1) {
            clear();
            printf("잘못 입력하였습니다.\n");
            continue;
        }
        break;
    }
    clear();
}
```

함수 파라미터 예제 #2

```
int ascending(int a, int b) {
    return a - b;
}
int descending(int a, int b) {
    return b - a;
}
void sort(int *arr, size_t size, int (*compare)(int, int)) {
    int tmp;
        for (size_t i=0 ; i<size-1 ; i++) {
            for (int j=i+1 ; j<size ; j++) {
                if (compare(arr[i], arr[j]) > 0) {
                tmp = arr[i];
                arr[i] = arr[j];
                arr[j] = tmp;
            }
        }
    }
}
```

```
void print(int* arr, size_t size) {
    for (size_t i=0 ; i<size ; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int ary[] = { 3, 9, 1, 4, 8, 6, 5, 10, 2, 7 };
    size_t size = sizeof(ary) / sizeof(ary[0]);
    print(ary, size);
    sort(ary, size, ascending);
    print(ary, size);
    sort(ary, size, descending);
    print(ary, size);
    return 0;
}
```

함수를 반환하는 함수

```
int executor(int signal) {
    printf("시그널 %d를 수신하였습니다.\n", signal);
    return signal;
}

int (*handle_signal(int signal, int (*handler)(int)))(int) {
    int result = handler(signal);
    printf("시그널 핸들러에 의해 처리된 값 : %d\n", result);
    return handler;
}

.....
int (*last_handler)(int) = handle_signal(9, executor);
handle_signal(10, last_handler);
```

함수 포인터 배열

- 함수 포인터 배열

```
int sum(int, int);
int subtract(int, int);
int multiple(int, int);
int divide(int, int);
int modulo(int, int);

.....
int a = 10;
int b = 3;
int (*fptrs[5])(int, int) = { sum, subtract, multiple, divide, modulo };
for (int i=0 ; i<sizeof(ftprs)/sizeof(fptrs[0]) ; i++) {
    printf("%d\n", fptrs[i](a, b);
}
```

풀어 봅시다.

```
int ascending(int a, int b);                                  // 오름차순 정렬에 사용될 함수
int descending(int a, int b);                                 // 내림차순 정렬에 사용될 함수
void sort(int *arr, size_t size, int (*compare)(int, int));   // 정렬 실행 함수
void print(int* arr, size_t size);                            // 배열 출력함수

int main() {
    int ary[] = { 3, 9, 1, 4, 8, 6, 5, 10, 2, 7 };
    size_t size = sizeof(ary) / sizeof(ary[0]);
    print(ary, size);
    sort(ary, size, ascending);
    print(ary, size);
    sort(ary, size, descending);
    print(ary, size);
    return 0;
}
```