작성자 : 김기희 (cupid072@naver.com)

타입 재정의

- 학습목표
  - 타입 재정의의 개념을 설명할 수 있다.
  - 타입 재정의의 장점을 기술할 수 있다.
  - 기본 자료형에 대한 타입 재정의를 할 수 있다.
  - 배열 타입을 재정의 할 수 있다.
  - 함수 포인터 타입을 재정의 할 수 있다.

typedef에 의한 타입 재정의

- 타입 재정의는 기존의 자료형을 다른 이름으로 다시 정의 하는 것을 말한다.
- 타입 재정의를 사용하면 기존의 복잡한 자료형을 간단한 형명으로 기술할 수 있다.
- 타입 재정의를 사용하면 프로그램의 유지 보수가 간단해 진다.

예) 변수 a의 크기가 반드시 4Byte 이어야 할 때

```
int a;              // int형 변수 a의 크기는 4Byte
```

- 16비트 시스템에서의 int형의 크기는 2Byte이다. 그러므로 4Byte를 확보하기 위해서는 위의 변수는 long형으로 선언되어야 한다.
- 아래와 같이 재정의 된 타입을 사용하는 경우 타입 재정의만 수정하면 된다.

```
typedef int int32_t;        // 타입 재정의를 수정하면
int32_t a;                  // 재정의 타입을 사용하는 모든 변수의 타입이 변경된다.
```

일반적인 자료형의 타입 재정의

- 일반적인 자료형의 타입 재정의는 다음과 같다.

  typedef 구형명 새형명

- typedef에 의한 타입 재정의 시 마지막 토큰만을 재정의 타입명으로 한다.
- 일반적으로 typedef에 의해 재정의 된 형명은 _t 접미사를 사용한다.

```
typedef int i32_t;
typedef unsigned long long int u64_t;
```

포인터 타입의 재정의

- 포인터 타입의 재정의는 기본 자료형의 재정의와 동일하다.

  typedef 포인터타입 재정의명

- 예)

```
typedef int * iptr_t;
int arr[] = { 1, 2, 3, 4, 5 };
iptr_t ptr = arr;
```

- 문제)

```
typedef int * iptr_t;
iptr_t ptr1, ptr2;          // ptr1과 ptr2 모두 int * 타입이다.
int * ptr1, ptr2;           // ptr1과 ptr2의 타입은 모두 int * 인가?
```

배열 타입의 재정의

- 배열 타입의 재정의는 배열의 선언 앞에 typedef 를 지정한다.

  typedef 배열의선언

- 타입의 재정의는 초기화 리스트를 기술할 수 없다.

```
typedef int intarr_t[];                 // 크기가 정해지지 않은 int형 배열 타입
intarr_t arr = { 1, 2, 3 };             // int arr[] = { 1, 2, 3 } 과 동일

typedef char menu_t[][20];              // 2차원 char형 배열타입 재정의
menu_t menu = { "입력하기", "출력하기", "종료하기" };

typedef char* menu_t[3];                // 원소가 3개인 포인터 배열타입 재정의
menu_t menu = { "입력하기", "출력하기", "종료하기" };
```

함수 포인터의 타입 재정의

- 함수 포인터의 재정의는 함수 선언 앞에 typedef 를 지정한다.

  typedef int (*재정의형명)(int, int); // func_t 타입이 재정의된 형 명

- 타입의 재정의는 초기화 리스트를 기술할 수 없다.

  - 함수타입 파라미터를 가지는 함수의 선언

  ➢ 타입 재정의를 사용하지 않을 경우

  void print(int* ptr, size_t size, void (*printer)(int*));

  ➢ 타입 재정의를 사용할 경우

  typedef void (*printer_t)(int*);
  void print(int* ptr, size_t size, printer_t printer);

함수 포인터 타입의 재정의

- 함수 포인터 타입 재정의를 사용하지 않을 경우

```
#include <stdio.h>


int executor(int signal) {
        printf("입력된 신호 : %d\n", signal);
        return signal;
}
int(*handle_signal(int signal, int(*handler)(int)))(int) {
        handler(signal);
        return handler;
}
int main() {
        int sig = -9;
        int (*handler)(int) = handle_signal(sig, executor);
        handler(sig);
        return 0;
}
```

- 함수 포인터 타입 재정의를 사용할 경우

```
#include <stdio.h>
typedef int (*handler_t) (int);

int executor(int signal) {
        printf("입력된 신호 : %d\n", signal);
        return signal;
}
handler_t handle_signal (int signal, handler_t handler) {
        handler(signal);
        return handler;
}
int main() {
        int sig = -9;
        int (*handler)(int) = handle_signal(sig, executor);
        handler(sig);
        return 0;
}
```