Structs and Classes

프로그래밍 입문(2)

중간고사

- 일정: 10월 26일 월요일 오후 4시
- 장소: 당일 공지
- 범위: 이번주에 배우는 내용까지
- 대부분의 문제는 코드를 작성하거나 해석하는 것.
- 단순히 코드만 쓰는 것이 아닌 왜 그렇게 해야하는지 이유를 잘 기억할 것.

Topics

- **typedef**
- **Namespace**
- **구조체(struct)**
- 클래스(class)

typedef

- 새로운 이름으로 데이터형을 정의하는 명령어.
- 기본적으로 주어진 데이터형의 별칭(alias)를 만드는 것.
- `typedef <type def> <type name>`
  - 순서에 주의.
  - def → name로 이름이 바뀐다고 생각하고 기억하면 좋습니다.
- 일반적으로 typedef를 이용한 데이터형의 이름은 _t를 뒤에 붙여 표현.

typedef

- 코드 가독성을 높이기 위해 사용가능.
- 플랫폼 독립적인 코딩에도 이용.
- 복잡한 타입을 간단하게 변경.
- C++11에서 사용가능한 문법
  - using <type name> = <type def>

```
typedef int size_t;
typedef double ratio_t;
using score_t = double;

size_t arr_size = 100;
ratio_t screen_ratio = 0.3;
score_t my_score = 4.0;

typedef int int_arr_t[2];
int_arr_t arr[3] =
    {
        {1, 2},
        {2, 3},
        {3, 4}
    };
```

플랫폼 독립적 데이터형

- C99 표준에서 모든 경우 같은 크기를 갖도록 보장하는 고정 너비 정수를 정의 하였음.
  - #include<stdint.h>
- C++는 C++11 표준에서 이를 채택.
  - #include<cstdint>
  - std Namespace에 포함.

```
typedef short int16_t;
typedef int  int32_t;
```

Namespace

```
namespace my_func {
    int atoi(const char* s) {
        return std::atoi(s) + 1;
    }
}
```

```
int num = std::atoi("999") + 1;
int num2 = my_func::atoi("999") + 1;
std::cout << "num = " << num << std::endl;
std::cout << "num2 = " << num2 << std::endl;
```

- Namespace란 모든 식별자(identifier)가 고유하도록 보장하는 영역을 의미.

- #include를 이용해 다양한 헤더파일을 포함시킬 때, 이름이 같은 경우가 있으면 충돌이 일어나게 됨.

- Namespace를 정의하여 이런 충돌을 방지할 수 있음.

  - e.g.) 김씨 → 서울사는 김씨

Namespace 사용

- 범위 해석 연산자(Scope Resolution Operator)인 '::'를 사용.
- Namespace 중첩도 가능.
- Namespace에 별칭도 지정할 수 있음.

```
namespace my_func {
    int atoi(const char* s) {
        return std::atoi(s) + 1;
    }
    namespace extra {
        int atoi(const char* s) {
            return std::atoi(s) + 10;
        }
    }
}
```

```
int num3 = my_func::extra::atoi("999") + 1;
namespace ext = my_func::extra;
int num4 = ext::atoi("999") + 1;
std::cout << "num3 = " << num3 << std::endl;
std::cout << "num4 = " << num4 << std::endl;
```

구조체(struct)

- 기본적으로 제공되는 기본 데이터형만으로는 다양하고 복잡한 데이터를 표현하는데 한계가 있음.

- 이런 기본 데이터형을 하나로 묶어 만든 보다 복잡한 구조의 데이터를 일반적으로 **레코드*(Record)*형 데이터**, 또는 구조체(struct, structure) 데이터라고 함.

- 프로그래밍 언어마다 struct 또는 record를 사용.

- 변수, 함수 등과 달리 **새로운 데이터형을 정의**할 수 있음.

구조체 정의

- 정의는 구조체 이름과 필드(또는 멤버)의 선언으로 이루어집니다.

- struct <struct_name> {
      <field_decl>;
      ......
      <field_decl>;
  };

- <field_decl> → int field1

- 구조체 이름은 변수 등과 달리 각 단어의 첫글자를 대문자로 하는 Camel Case로 작성합니다.

```
struct Person {

    int id; //field1
    std::string name; //field2
    int age; //field3

}; //;를 잊지 말 것.


struct PetShop {

    Person owner;
    std::string address;

};
```

구조체 참조

- 구조체의 필드에 접근하기 위해서는 Member Selection Operator인 '.'을 이용.
- <struct>.<field>
- 지정하는 방법이 다를 뿐, 일반적인 변수와 동일하게 사용할 수 있습니다.
- 마찬가지로 초기화하지 않은 필드를 참조하면 알 수 없는 값이 나옴.

```
//Field에 값 지정.
Person dva;
dva.id = 1000;
dva.name = "Song Hana";
dva.id += 100;

std::cout << "D.VA's ID: " << dva.id;
```

```
std::cout << "D.VA's Age: " << dva.age;
```

구조체 초기화

- 구조체 필드 각각에 값을 입력하여 변수를 초기화하듯이 하는 방법.

- 이 경우 필드가 많아지면 매우 번거로움.

- C++에서는 초기화 목록을 이용하여 필드를 순서대로 초기화할 수 있음.

- `<struct> <name> = { <field1_val>, <field2_val>, ... <fieldn_val> };`

- 명시적으로 지정하지 않은 것은 0으로 초기화 (배열과 마찬가지).

```
Person genji = {1111, "Genji", 35};

//C++11
Person winston {9999, "Winston"};
```

구조체의 중첩

- 하나의 구조체(PetShop) 안에 다른 구조체(Person)의 변수를 필드로 사용할 수 있음.

- 이 경우 '.'연산자를 중첩해서 사용하여 내부 구조체의 필드에 접근할 수 있음.

- 구조체의 이름(Person)이 아니라 구조체로 선언된 필드의 이름(owner)을 사용한다는 것에 주의.

```
struct PetShop {
    Person owner;
    string address;
};

PetShop store = { winston, "Gibraltar"};
cout << "Store Owner's name is ";
cout << store.owner.name << endl;
```

구조체와 배열

```
struct Person {
    int id;
    string name;
    int age;
};

Person* people = new Person[100];
Person people2[100] = { {1000, "Garen", 100} };
```

- 구조체를 이용해 배열을 선언하는 것도 가능.
- 다른 데이터형의 배열과 동일하게 동작함.
- 초기화의 경우에는 다차원 배열과 유사한 방식으로 초기화.

구조체와 포인터

```
//구조체 포인터
Person keanu = { 1, "Keanu Reeves", 56 };
Person* neo = &keanu;
cout << "His name is " << keanu.name << endl;
cout << "Neo's real name is " << neo->name << endl;
```

- 구조체의 포인터 변수도 선언 가능.
- 일반 변수와 동일한 방식.
- 포인터를 이용해 구조체의 필드에 접근하기 위해서는 '.' 대신 '->' 연산자를 사용함.

함수와 구조체 사용

```
Person clone(Person p) {
    return { p.id+1000, p.name, 1 };
}
```

```
//함수로 호출
Person newPerson = clone(keanu);
cout << "New person is also " << newPerson.name << endl;
cout << "But his age is " << newPerson.age << endl;
```

- 구조체를 함수의 매개변수나 반환값으로 사용하는 것이 가능.
- 이 경우 동시에 다양한 값을 하나로 묶어 함수에 넘기거나, 결과를 넘겨 받는 것이 가능함.
- 구조체는 사용자가 새로 정의한 데이터형으로 동작하기 때문.

Summary

- typedef
- Namespace
- 구조체