Chapter 6

스택 변수, 함수내의 변수 vs 오브젝트내의 변수

이 챕터의 초반부에서 언급했듯이, C++에서는 두가지 타입의 메모리가 있습니다, 바로 스택(stack)과 힙(heap)입니다. 우선 스택에 대해서 알아보도록 하죠. 스택은 가장 쉽게 사용할 수 있는 메모리 타입이자, 가장 자주 사용하게 될 메모리타입이기도 합니다.

스택은 포인터나 new 키워드를 사용하지 않은 한, 함수 안, 혹은 클래스의 .h 내에서 **변수가 선언될 때 사용되는 메모리 타입**입니다.

스택이라고 불리우는 이유는 이것이 스택과 같이 구성되어있기 때문입니다. 프로그램에서 함수를 호출할 때, 메모리 내부에는 함수호출로 할당된 메모리 영역이 존재합니다. 이 특정 함수호출이 존재하는 동안 이 영역에 변수들이 생성될 수 있습니다.

이러한 변수들은 함수호출이 끝날때 사라집니다. 따라서 아래와 같은 예제는 동작할 수 없습니다:

```
void ofApp::setup(){
  int a = 0;
}

void ofApp::update(){
  a = 5; // setup의 스코프를 벗어났기 때문에 존재하지 않아서 에러
}
```

또한 이 함수가 다시 호출된다 하더라도, 이전 함수 호출시에 스택변수에 저장되었던 값을 기대할 수 없습니다. 즉 스택변수에는 값을 저장할 수 없습니다.

일반적으로 블록내에서 선언된 변수는 해당 블록 내에서만 존재한다고 말할 수 있습니다. c++에서 블록이란 {}로 명시됩니다. 따라서 아래의 예제 또한 동작하지 않습니다:
변수의 수명은 변수영역(scope)내에서 라고도 불립니다.

```
for (int i=0;i<10;i++){
  int a = 5;
}
cout << a << endl; // for{} 블록 바깥에서는 존재하지 않으므로 에러
```

함수 내에서 변수를 선언하는 부분을 .h파일의 클래스 선언부에서 변수를 선언하도록 분리해봅시다. 다음과 같이요:

```
class Ball{
 public:
  void setup();
  float pos_x;
}
```

이런 종류의 변수를 '인스턴스 변수' 라고 합니다. 이는 모든 클래스의 인스턴스나 오브젝트들은 카피되기 때문입니다. 이러한 동작은 함수내에서 스택 변수들을 선언할때의 동작과 완벽히 같습니다. 이들은 이들이 선언된 {} 내에서만 존재하며, 위의 경우 오브젝트의 {} 내에서 존재하게 됩니다.

이러한 변수들은 클래스 내 어디에서든 접근이 가능하기 때문에 이 데이터들이 필요할 경우엔, 위에서 선언한 클래스 오브젝트 내 함수를 통해서 접근할 수 있습니다.

스택 내의 메모리는 제한적인데, 사용하고 있는 컴퓨터의 아키텍쳐, 운영체제, 그리고 심지어 컴파일러에 따라서 사이즈가 달라집니다. 어떠한 시스템의 경우 어플리케이션이 실행중인 때에도 크기가 바뀌기도합니다. 하지만 대부분은 이 한계에 다다를 일은 없습니다.

만일 모든 변수들을 스택에 생성했다 하더라도, 일반적으로 크기가 가변적인 오브젝트들(예를들어 벡터라든가)은 내부적으로 컴퓨터에서 사용가능한 전체 메모리의 크기에 제한을 받는 힙영역을 사용하기 때문에, 우리가 걱정할 필요는 없습니다.

~~다음 섹션에서는 힙메모리가 어떻게 동작하는지, 그리고 힙과 스택을 사용할때의 장점들을 살펴보겠습니다.~~

Array, pointer, and dynamic allocation

1. 연중 일일 최고온도을 저장하기위한 배열을 선언하세요. (1년은 365일로 가정하고, 기온은 소수 첫째자리까지 저장한다.) 그리고 모든 기온을 0.0도로 초기화하세요.

2. 다음 주어지는 동물들로 enum class를 생성하세요: chicken, dog, cat, elephant, duck, and snake.
   그리고 각 동물의 다리 개수를 저장하는 배열을 정의한 뒤, 코끼리의 다리 개수를 출력하세요.

```
An elephant has 4 legs.
```

3. 다음 주어진 배열의 원소를 출력하세요.

```
double array[] { 0.0000001, 0.1, 0.2, 0.0, 0.3, 1.0, 0.5, 0.4, 0.7 };
```

4. 위 배열을 가지고 작업하세요.
   유저에게 1에서부터 9까지를 입력받으세요. 만약, 다른 값을 넣으면 반복해서 유저에게 물어 보세요. 1에서 9까지 중 하나를 입력받으면, 배열을 출력하고, 해당하는 인덱스의 원소도 출력하세요.

유저의 인풋 값 검사는 아래 코드를 활용하세요.

```
// if the user entered something invalid
if (std::cin.fail()) {
    std::cin.clear(); // unset failbit
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // skip
bad input
}
```

5. 문제 3의 배열을 가지고 작업하세요.
   최소 값을 출력하세요.

6. 문제 3의 배열을 오름차순 정렬하세요. (swap 이용)

정렬의 종류

- 선택정렬 (selection sort)
- 버블정렬 (bubble sort)
- 삽입정렬 (insertion sort)
- 머지정렬 (merge sort)
- 퀵정렬 (quick sort)

7. 포인터의 이해: 다음 코드를 상상해서 어떤 출력이 나올지 적으세요. (`short`는 2bytes, 32-bits 머신)

```
short value = 7; // &value = 0012FF60
short otherValue = 3; // &otherValue = 0012FF54

short *ptr = &value;

std::cout << &value << '\n';
std::cout << value << '\n';
std::cout << ptr << '\n';
std::cout << *ptr << '\n';
std::cout << '\n';

*ptr = 9;

std::cout << &value << '\n';
std::cout << value << '\n';
std::cout << ptr << '\n';
std::cout << *ptr  << '\n';
std::cout << '\n';

ptr = &otherValue;

std::cout << &otherValue << '\n';
std::cout << otherValue << '\n';
std::cout << ptr << '\n';
std::cout << *ptr << '\n';
std::cout << '\n';

std::cout << sizeof(ptr) << '\n';
std::cout << sizeof(*ptr) << '\n';
```

8. 아래 코드에서 틀린 점을 고치세요.

```
int value = 45;
int *ptr = &value; // declare a pointer and initialize with address of value
*ptr = &value; // assign address of value to ptr
```

9. 동적할당: 프로그램을 작성하세요.

- 조건1: 얼마나 많은 이름을 넣을 것인지 물어보세요.
- 조건2: 각각의 이름은 유저가 입력하세요.
- 조건3: 이름들을 정렬하는 함수를 호출하세요.
- 조건4: 정렬된 이름들을 출력하세요.

> 힌트

- 이름들을 저장하기 위해서 std::string을 활용한 가변배열을 하세요.
- std::string은 문자열을 비교하는 연산자 < 와 >를 제공합니다.

> 결과 예시

```
How many names would you like to enter? 5
Enter name #1: Jason
Enter name #2: Mark
Enter name #3: Alex
Enter name #4: Chris
Enter name #5: John

Here is your sorted list:
Name #1: Alex
Name #2: Chris
Name #3: Jason
Name #4: John
Name #5: Mark
```

10. range-based for loop: 프로그램을 작성하세요.
    다음의 이름을 가지는 고정된 배열을 선언하세요: Alex, Betty, Caroline, Dave, Emily, Fred, Greg, and Holly
    사용자에게 이름을 입력받으시고, `range-based for` 반복문을 사용하여 입력한 이름이 배열 안에 존재하는지 확인하세요.

> 결과 예시

```
Enter a name: Betty
Betty was found.

Enter a name: Megatron
Megatron was not found.
```

Comprehensive quiz

Q1

플레이어가 3가지 타입의 아이템(health potions, torches, and arrows)을 가질 수 있는 게임 만든다고 생각해봅시다. 각기 다른 아이템을 표현하기위해 `enum class`를 사용하시고, 고정된 크기의 배열을 이용해서 플레이어가 가지고 다니는 각 아이템의 개수를 저장하세요. 플레이어는 포션x2, 토치x5, 그리고 화살x10개를 가지고 시작합니다. CountTotalItems()함수를 작성해 플레이어가 가지고 있는 아이템 전체 개수를 반환하시고. main() 함수에서 그 결과를 출력하세요.

Q2

프로그램을 아래와 같은 조건을 만족하도록 작성하세요.

- 학생의 성씨(姓)와 성적 [0~100]을 가지는 구조체를 정의하세요.
- 사용자에게 얼마나 많은 학생을 만들 것인지 물어보세요.
- 학생들을 모두 담을 수 있는 가변 크기의 배열을 만드세요.
- 유저가 이름과 성적을 입력합니다.
- 유저가 입력을 모두 마치면, 성적을 내림차순 정렬하세요.
- 마지막으로 이름과 성적을 그 순서대로 모두 출력하세요.

> 결과 예시

```
Enter the number of students: 5
# Student 1
- First name: Joe
- Grade [0, 100]: 82
# Student 2
- First name: Terry
- Grade [0, 100]: 73
# Student 3
- First name: Ralph
- Grade [0, 100]: 4
# Student 4
- First name: Alex
- Grade [0, 100]: 94
# Student 5
- First name: Mark
- Grade [0, 100]: 88

Sorted grades of all of students in descending order.
Alex got a grade of 94
Mark got a grade of 88
Joe got a grade of 82
Terry got a grade of 73
Ralph got a grade of 4
```

Q3

아래 코드의 문제점을 찾아 고치세요.

> a

```
int main()
{
    int array[5] { 0, 1, 2, 3 };
    for (int count = 0; count <= 5; ++count)
        std::cout << array[count] << " ";

    return 0;
}
```

b

```
int main()
{
    int x = 5;
    int y = 7;

    const int *ptr = &x;
    std::cout << *ptr;
    *ptr = 6;
    std::cout << *ptr;
    ptr = &y;
    std::cout << *ptr;

    return 0;
}
```

c

```
void printArray(int array[])
{
        for (const int &element : array)
                std::cout << element << ' ';
}


int main()
{
        int array[] { 9, 7, 5, 3, 1 };
        printArray(array);

        return 0;
}
```

d

```
int* allocateArray(const int length)
{
    int temp[length];
    return temp;
}
```

> e

```
int main()
{
    double d(5.5);
    int *ptr = &d;
    std::cout << ptr;

    return 0;
}
```

Q4

카드 게임을 만들어 봅시다.

- 카드 덱은 서로 다른 52장으로 이루어져있습니다. (13 card ranks of 4 suits)
  랭크(2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace)와 문양(clubs, diamonds, hearts, spades)에 대해서 이넘을 만드세요.
- 각 카드는 `Card`라는 이름의 구조체로 표현할 수 있는데, `Card`는 랭크와 문양으로 구성됩니다.
- PrintCard()라는 함수를 만들고, `const Card` 레퍼런스를 파라미터로 받고 랭크와 문양을 2글자 코드로 반환하는 함수로 만드세요. (예를 들어, the jack of spades would print as JS)
- 카드 덱은 고정된 크기의 배열을 이용하고, 모든 카드를 초기화하세요.
  - 힌트: 이넘을 정수로 변화하고 싶다면 `static_cast`를 사용하세요.
- PrintDeck()이라는 함수를 만들고, 덱을 컨스턴트 레퍼런스 파라미터로 받아서 출력하세요.
  - 힌트: `range-based for` 반복문을 사용하세요.
- SwapCard() 함수를 만들고, 두 개의 카드를 받아서 스왑하는 함수를 만드세요.
- ShuffleDeck() 함수를 만들고, 덱을 섞으세요. 이것을 하기 위해선, 반복문을 사용하세요. [1, 52] 중 랜덤넘버 하나를 고르고, 현재 카드와 랜덤넘버의 카드로 SwapCard 함수를 호출하세요.
- main() 함수에서 덱을 섞고, 출력하세요.
- GetCardValue() 함수를 만들고, 카드의 값을 출력하세요. (예를 들어, 2 is worth 2, a ten, jack, queen, or king is worth 10. Assume an Ace is worth 11)
- 좋습니다! 이로써 간단한 블랙잭 게임의 틀을 만들었습니다.

Q5 (extra credit)

블랙잭 게임을 진행하세요.

> 게임 규칙

- 딜러는 시작할 때 카드 한 장을 얻습니다. (실제로는 딜러는 두 장을 받고, 그 중 한 장은 엎어 놓습니다. 이것은 코딩하기 어려워서 간단한 룰로 진행할 겁니다.)

- 플레이어는 두 장을 얻습니다.
- 플레이가 먼저 시작하고요.
- 플레이어는 "hit"을 반복하거나 "stand"를 선택할 수 있습니다.
- 만약 "stand"를 선택하면, 턴이 끝나고, 그들이 가지고 있는 카드의 점수를 계산합니다.
- 만약 "hits"를 선택하면, 카드 한 장을 더 받고 점수가 누적됩니다.
- Ace는 총점이 유리해지는 방향으로 1 혹은 11로 계산할 수 있습니다. (이것을 코딩하기는 힘들어서 간단하게 11로 고정하겠습니다.)
- 만약 카드 점수가 21을 넘으면, 그 즉시 게임에서 집니다.
- 플레이어 턴이 끝나면 딜러 차례입니다.
- 딜러는 17이 넘기전까지 (16까지) 계속해서 카드를 뽑습니다. 만약 넘어버리면 그 즉시 "stand" 됩니다.
- 만약 딜러가 21이 넘으면, 그 즉시 플레이어가 승이고요.
- 넘지 않았을 경우, 플레이어의 점수가 높으면 플레이어 승, 같거나 작은 경우는 패입니다.

> 코딩 조건

- PlayBlackjack() 함수를 만들고, 플레이어의 승패를 출력하세요.
  - 힌트: 무작위로 섞인 덱을 파라미터로 받으세요.
  - 힌트: 덱의 가장 처음 카드로 `p_card` 포인터를 초기화하세요. 이것을 사용해 덱의 모든 카드를 다룰 것입니다.

> 카드를 다루는 경우, 현재 카드의 값을 얻고 다음 카드로 포인터를 넘겨야하는데 이때는 아래처럼 코딩하시면 현재 카드의 값을 반환하고 다음 카드의 포인터로 전진할 수 있습니다. (반환된 값은 플레이어나 딜러의 총점에 더해저야합니다.)

```
GetCardValue(*p_card++);
```

  - 힌트: 플레이어와 딜러의 점수를 계산하기 위해 두 개의 정수형 변수를 선언하세요.
  - 힌트: 위 게임 규칙을 바탕으로 블랙잭을 코딩하세요.