

객체지향프로그래밍 실습과제보고서 #2

2019 년 3 월 15 일

김예원

소프트웨어학과

실습 과제 #2. 포인터

*2-1 포인터 배열을 만들어 이 배열을 문자열로 각각 초기화한 후 그 내용 찍어 보기

: 포인터배열을 문자배열로 초기화한 뒤 포인터배열을 함수로 넘겨 최소값을 찾아 출력하는 프로그램이었다.

(1) 자료구조

-함수 main

```
char *ptr_ary[5]; //포인터배열
char animal[5][20]; //문자배열
```

그림1 main에 선언한 포인터배열과 문자배열

char형 *ptr_ary[5]라는 포인터 배열을 선언했다. 5개의 char형 주소를 담을 수 있다. char형 문자 배열 animal[5][20]을 선언했는데, 단어의 글자 수는 임의로 20이라고 정했다.

-함수 int short_word(char **arr);

```
int short_word(char **arr) { //문자열 길이 비교하고 짧은 문자열의 인덱스를 반환하는 함수
    int min_index = 0;

    for (int i = 1; i < 5; i++) {
        if (strlen(arr[min_index]) > strlen(arr[i]))
            min_index = i;
    }

    return min_index;
}
```

그림2 포인터 배열을 매개변수로 받는다. strlen함수로 최소 길이의 단어를 담고 있는 인덱스를 찾아 반환한다.

지역 변수 int min_index=0을 선언해서 최소 길이를 가진 단어의 인덱스를 찾는데 사용될 수 있게끔 했다.

또한 int strlen(const char*)을 선언해서 문자열의 길이를 구하게 했다.

(2) 문자배열로 입력 받은 뒤 포인터배열을 초기화했다.

```
for (i = 0; i < 5; i++) { //반복문으로 5번 반복
    cout << "문자열을 입력하세요 : ";
    cin >> animal[i];
    ptr_ary[i] = animal[i]; //포인터배열을 문자배열로 초기화
    cout << "입력된 문자열 : " << animal[i] << "\n\n";
}
```

그림3 반복문을 사용했다.

반복문을 통해 5번 반복함으로써 인덱스를 바꿔가며 각각의 주소에 새로운 데이터를 담을 수 있게 했다.

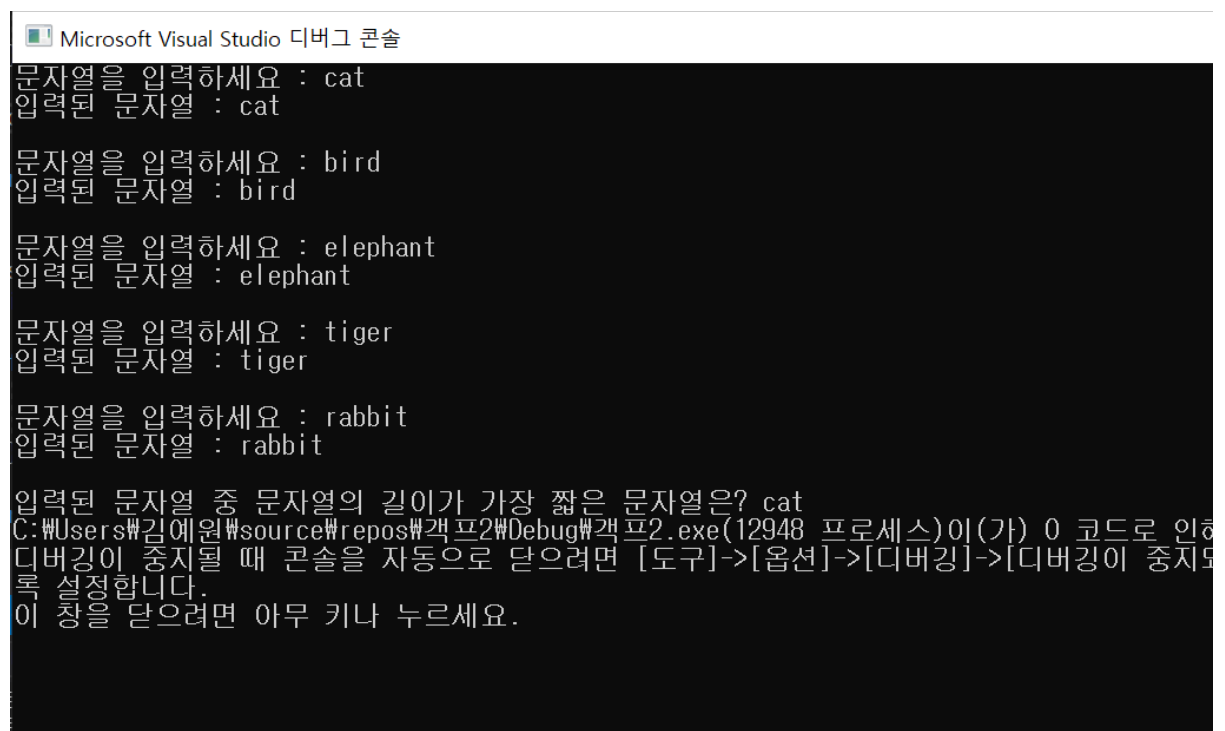
(3) `int short_word(char **arr);` 함수 제작

```
cout << "입력된 문자열 중 문자열의 길이가 가장 짧은 문자열은? " << animal[short_word(ptr_ary)];
//short_world 함수 호출, 반환된 값은 즉시 animal의 인덱스가 됨
```

그림 4 함수가 호출되어 출력하는 과정

문자열의 길이를 비교해 최소의 문자열의 인덱스를 반환하는 함수이다. 함수 호출은 다음과 같이 하여 반환된 즉시 문자배열 animal의 인덱스 역할을 하게끔 만들었다.

(4) 실행화면



```
Microsoft Visual Studio 디버그 콘솔
문자열을 입력하세요 : cat
입력된 문자열 : cat

문자열을 입력하세요 : bird
입력된 문자열 : bird

문자열을 입력하세요 : elephant
입력된 문자열 : elephant

문자열을 입력하세요 : tiger
입력된 문자열 : tiger

문자열을 입력하세요 : rabbit
입력된 문자열 : rabbit

입력된 문자열 중 문자열의 길이가 가장 짧은 문자열은? cat
C:\Users\김예원\source\repos\객프2\Debug\객프2.exe(12948 프로세스)이(가) 0 코드로 인해
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[옵션]->[디버깅]->[디버깅이 중지되
록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.
```

문자열을 입력하라는 명령과 입력된 문자열을 바로 확인시켜주는 과정이 5번 반복됨을 확인할 수 있으며, 가장 짧은 문자열이 출력되고 이러한 과정을 끝으로 프로그램이 종료되는 것을 확인할 수 있다.

*2-2 const 포인터 이해하기

: const 포인터의 개념에 대해 이해하고 const가 사용된 3가지 경우를 통해 오류 원인을 파악해 각각의 소스코드가 틀린 이유를 분석하는 과제였다.

A. 상수에 대한 포인터

```
#include <iostream>
using namespace std;

int main() {
    int i1 = 10;
    int i2 = 20;
    const int* pInt1; //상수에 대한 포인터 (Pointer to Vonstant)

    pInt1 = &i1;
    *pInt1 = 30;

    return 0;
}
```

`const int* pInt1;` 에서 `const`가 `int` 앞에 붙었으므로 `p`는 상수만 가리키는 정수형 포인터이다. 가리키고 있는 대상이 상수이므로 `*pInt1 = 30;` 처럼 가리키고 있는 대상을 변경할 수 없다. 하지만 포인터가 다른 대상을 가리킬 수 있으므로 `pInt1 = &i1;` 는 틀리지 않는다.

B. 상수 포인터

```
#include <iostream>
using namespace std;

int main() {
    int i1 = 10;
    int i2 = 20;
    int* const pInt2 = &i1; //상수포인터(Const Pointer)

    pInt2 = &i2;
    *pInt2 = 50;

    return 0;
}
```

`int* const pInt2 = &i1;` 에서 `pInt2`는 정수형 변수를 가리키는 포인터 상수이다. 따라서 `pInt2 = &i2;` 처럼 다른 값을 가리킬 수 없다. `*pInt2 = 50;`처럼 가리키는 값은 변경이 가능하다.

C. 상수에 대한 상수 포인터

```
#include <iostream>
using namespace std;

int main() {
    int i1 = 10;
    int i2 = 20;
    const int* const p = &i2; //상수에 대한 상수 포인터(Const Pointer to Constant)

    p = &i1;
    *p = 40;

    return 0;
}
```

`const int* const p = &i2;` 에서 `p`는 상수형 정수 포인터를 가리키고 있는 포인터 상수이다. 상수이므로 `p = &i1;` 처럼 다른 값을 가리킬 수 없다. 또한 `*p = 40;` 가리키고 있는 상수를 변경할 수 없다.

우선 이번 실습을 통해서 포인터 배열과 문자배열에 대해 다시 한번 공부할 수 있는 기회를 가질 수 있었다. 1학년 때 C언어 공부하면서 포인터 배열 같이 배열에 포인터가 활용된 부분들에 대해서는 이해가 잘 되지않아 제대로 학습하지 못한채로 넘어갔었다. 이번 실습을 통해 포인터 배열의 쓰임새와 구조에 대해서 잘 이해할 수 있었다. 실습 2번의 const는 C언어를 배울 때 잠깐 사용했던 것이어서, 자료형과 변수명 앞에 붙는 모습이 다소 생소하게 다가왔었다. 수업을 듣고 실습을 하면서 책을 읽거나 검색을 해보는 등 스스로 공부할 기회를 가졌는데, 자료형 앞뒤에 붙어서 상수로 선언한다는 점이 재밌게 느껴졌었다.

실습을 하면서 어려웠던 부분은 포인터배열을 매개변수로 넘길 때였다. 함수를 만들지 말고 그냥main에 쪽 이어서 작성할까 싶을 때도 있었지만 이번 기회를 통해서 확실히 배우자는 생각하게 되었다. 이중 포인터를 사용해보지 않아서 포인터를 쓰면서도 생소하다는 느낌이 계속 들었다. 포인터의 개념에 대해서 다시 명확하게 다지고 나니 약간의 확신이 들었고 성공적으로 매개변수를 넘길 수 있었다.

const가 어느 상황에 유용하게 쓰이는지 알고 싶다. c++에서 데이터가 변경되지 않도록 지켜야 하는 상황들에 어떤 것들이 있는지 더 자세하게 알고싶다는 생각이 든다.