〈목요일7조〉

전기전자기초설계및실습
〈#1, Array and Pointer〉

담당교수 : 조용범 교수님
실험날짜 : 2019. 03. 21
조 : 목요일 7 조
조원 : 201411342 김지형

전기전자기초실습

- C Programming Language
- 담당교수: 조용범 교수님
- 작성 기준일: 2019. 03. 28. (목)

1. Title

Array and Pointer

2. Name

7조 201411342 김지형

3. Abstract

Array와 Pointer의 기본 개념을 이해하고, 함수의 호출 및 동적 할당과 해제 방법을 익힌다.

4. Environments

- OS: macOS Mojave 10.14.3
- IDE: VSCode
- Compiler: gcc 4.2.1.

5. Background

A. Array

- An array is a collection of data items, all of the same type, accessed using a common name.
- A one-dimensional array is like a list; A two dimensional array is like a table; The C language places no limits on the number of dimensions in an array, though specific implementations may.
- Some texts refer to one-dimensional arrays as vectors, two-dimensional arrays as matrices, and use the general term arrays when the number of dimensions is unspecified or unimportant.

B. Array and Pointer

- An array is a fundamental data structure built into C. A thorough understanding of arrays and their use is necessary to develop effective applications. Misunderstandings of array and pointer usage can result in hard-to-find errors and less than optimal performance in applications. Array and pointer notations are closely related to each other and can frequently be used interchangeably in the right context.

C. Misconceptions of Pointers and Arrays

- A common misconception is that an array and a pointer are completely interchangeable. An array name is not a pointer. Although an array name can be treated as a pointer at times, and array notation can be used with pointers, they are distinct and cannot always be used in place of each other. Understanding this difference will help you avoid incorrect use of these notations. For example, although the name of an array used by itself will return the array's address, we cannot use the name by itself as the target of an assignment.

6. Experimental Results

A. 실험 1

- Source Code

```c
#include <stdio.h>

#define STDNUM 150

int main()
{
   char str[STDNUM];
   int index = 0, count = 0;

   printf("Plz input your text: ");
   scanf("%[^₩n]s", str);

   while (str[index])
   {
      if (str[index] == ' ')
         count++;
      index++;
   }
   printf("Number of blanks is: %d₩n", count);

   return 0;
}
```

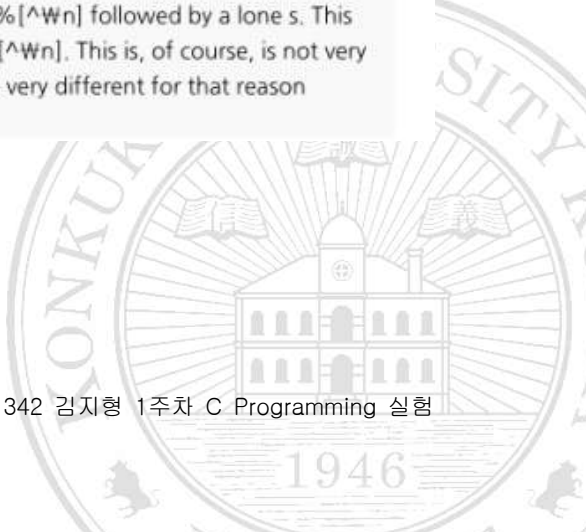
- Data

```
✘ KJH@KimJihyeongui-MacBook-Pro ~/OneDrive - konkuk.ac.kr/School_2019_1st/expreriment .dist/a.out
Plz input your text: Hello World! My Bro!
Number of blanks is: 3
```

- Discussion

- 전처리기 define을 이용해 배열의 MAX 값을 설정할 수 있었다.
- %[^₩n]s를 통해 입력값을 받음으로써 공백을 포함한 문자열을 받을 수 있었다. 여기서 []와 s는 독립적인 format specifier로서, %[^₩n]을 단독으로 쓸 수 있다.

> **difference between return values of scanf("%s") and scanf("%[^₩n]s")**
>
> Your scanf("%[^₩n]s") contains two format specifiers in its format string: %[^₩n] followed by a lone s. This will require the input stream to contain s after the sequence matched by %[^₩n]. This is, of course, is not very close in behavior to plain %s. Matching sequences for these formats will be very different for that reason alone.



B. 실험 2

- Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define INT_MAX 500
#define INT_MIN 1

void statistic(int *, int, float *, int *, int *);

int main(int argc, char *argv[])
{
   printf("---------- Statistics Program ----------₩n");

   int N = 0;
   int *ptr = NULL;
   float avr = 0;
   int max = 0;
   int min = 0;
   int i;
   // int fi;

   srand((unsigned)time(NULL));

   if (argc < 2)
   {
      printf("Check CMD Input!₩n");
      return -1;
   }
   N = atoi(argv[1]);
   if (N < 1)
   {
      printf("Check CMD Input!₩n");
      return -1;
   }

   printf("N = %d₩n", N);

   ptr = (int *)malloc(N * sizeof(int));

   for (i = 0; i < N; i++)
   {
      ptr[i] = rand() % INT_MAX;
      printf("ptr[%3d] = %3d₩n", i, ptr[i]);
   }
   statistic(ptr, N, &avr, &max, &min);

   printf("Average: %f₩n", avr);
   printf("    max: %d₩n", max);
   printf("    min: %d₩n", min);
   free(ptr);

   return 0;
```

```
}

void statistic(int *arr, int arr_size, float *avr, int *max, int *min)
{
    float sum = 0;
    int i;
    // int fi = 0;
    *min = INT_MIN;
    *max = INT_MAX;

    for (i = 0; i < arr_size; i++)
    {
        sum += *(arr + i);
        // printf("i: %d / sum: %f / arr: %d\n", i, sum, *(arr + i));
    }
    *avr = sum / i;
}
```

- Data

```
✘ KJH@KimJihyeongui-MacBook-Pro ~/OneDrive - konkuk.ac.kr/School_2019_1st/expreriment .dist/a.out 5
---------- Statistics Program ------------
N = 5
ptr[  0] = 433
ptr[  1] = 101
ptr[  2] = 449
ptr[  3] = 135
ptr[  4] = 453
Average: 314.200012
    max: 500
    min: 1
```

- Discussion

```
int atoi (const char * str);
```

- atoi() 함수
  - Convert string to integer
  - Parses the C-string str interpreting its content as an integral number, which is returned as a value of type int.
  - The function first discards as many whitespace characters (as in isspace) as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many base-10 digits as possible, and interprets them as a numerical value.
- malloc()을 통한 동적할당을 통해 힙 메모리에 저장할 수 있음을 확인했다.

```
void* malloc( size_t size );
```

- Allocates size bytes of uninitialized storage.
- If allocation succeeds, returns a pointer to the lowest (first) byte in the allocated memory block that is suitably aligned for any object type with fundamental alignment.

7. Analysis

2가지의 예제를 통해서 다음과 같은 내용을 알 수 있었다. - 전처리기 define을 통한 배열의 최댓값 설정 - %[^₩n]의 기능 - atoi() 함수 - malloc(), calloc()

8. Conclusion

배열과 포인터는 유사한 공통점을 지니고 있으나 최대 공간의 설정 필요성, 동적 할당 등의 이유로 다르게 사용된다. 이번 실험을 통해서 배열과 포인터가 적재적소에 사용할 수 있는 케이스를 확인 할 수 있었다.

9. References

- Array: https://www.cs.uic.edu/~jbell/CourseNotes/C_Programming/Arrays.html
- Array and Pointer / Misconceptions of Pointers and Arrays: https://www.oreilly.com/library/view/understanding-and-using/9781449344535/ch04.html
- atoi: http://www.cplusplus.com/reference/cstdlib/atoi/