

C애플리케이션구현 포트폴리오

학과 : 컴퓨터정보공학과

학번 : 20160760

이름 : 박윤석

목 차

11) 문자와 문자열

- ▶ 문자와 문자열
- ▶ 문자열관련 함수
- ▶ 여러문자열 정리

12) 변수 유효범위

- ▶ 전역변수와 지역변수
- ▶ 정적 변수와 레지스터 변수
- ▶ 메모리 영역과 변수 이용

13) 구조체와 공용체

- ▶ 구조체와 공용체
- ▶ 자료형 재정의
- ▶ 구조체와 공용체의 포인터와 배열

11장

문자와 문자열

문자와 문자열

■문자와 문자열 개념

▸ 문자와 문자열의 개념

문자 : 영어의 알파벳이나 한글의 한 글자를 ' '를 이용하여 표기

ex) 'A', '가', '\$'

문자 선언 ex) `char ch = 'A';`

· 문자열

문자의 모임인 일련의 문자

일련의 문자 앞 뒤로 " "를 사용하여 표기

문자열은 작은따옴표('abc')로 둘러싸도 문자가 될 수 없으므로 오류발생

ex)"한글", "ENGLISH", "문자열"

문자열 선언 ex)`char c[] = "C language";`

문자와 문자열

■문자와 문자열의 선언

· 문자 배열

문자열을 저장하려면 사용해야함

문자열의 마지막을 의미하는 NULL문자 '\0'가 마지막에 저장되어야함

문자열이 저장되는 배열크기는 반드시 저장될 수보다 1이 커야함

배열 초기화 시 배열크기는 저장하지않는게 더 편리

만일 지정한 배열크기가 (문자수+1)보다 크면 나머지부분은 모두 '\0'문자로 채워진다.

■문자열을 위한 문자 하나하나의 초기화 선언

ex)char java[] = {'J','A','V','A', '\0'};

(마지막에 '\0'을 빼면 대입시에는 문제가 없지만 출력할때 문제가 발생함)

함수 printf()를 사용한 문자와 문자열 출력

■ 문자열 구성하는 문자 참조

- 문자열 상수를 문자 포인터에 저장하는 방식
- 문자열 출력도 함수 printf()에서 포인터 변수와 형식제어문자 %s로 처리할 수 있다
- 문자 포인터에 의한 선언으로는 문자 하나하나의 수정은 할 수 없다.

문자 포인터를 사용한 문자열 처리

```
char* java = "java";  
printf("%s", java);
```

```
int i = 0;  
while (java[i])  
    printf("%c", java[i++]);  
printf(" ");
```

문자 포인터가 가리키는 문자 이후를 하나하나 출력

함수 printf()를 사용한 문자와 문자열 출력

■ '\0' 문자에 의한 문자열 분리

- ▶ 함수 printf()에서 %s는 문자 포인터가 가리키는 위치에서 NULL 문자까지를 하나의 문자열로 인식함

```
char c[] = "C C++ Java";  
printf("%s\n", c);  
c[5] = '\0';  
printf("%s\n%s\n", c, (c + 6));
```

C C++ 까지 NULL로 문자열 분리

c만 출력한다면 여기까지만 출력

```
C C++  
,
```

다양한 문자 입출력

■ getchar()

- ▶ 문자의 입력에 사용됨
- ▶ 라인 버퍼링 방식을 사용

■ getche()

- ▶ 버퍼를 사용하지 않고 문자를 입력하는 함수
- ▶ 함수를 이용하려면 헤더파일 conio.h를 삽입해야함

■ getch()

- ▶ 문자 입력을 위한 함수
- ▶ 입력한 문자가 화면에 보이지 않는 특성이 있음

여러 컴파일러에서 함수 `getche()`, `getch()`는 `_getche()`, `_getch()`로 이름이 수정되어 서비스되고 있음

문자열 입력

■ gets()

- ▶ 한 행의 문자열 입력에 유용한 함수
- ▶ 마지막에 입력된 '\n'가 '\0'로 교체되어 인자인 배열에 저장된다.

■ puts()

- ▶ 한 행에 문자열을 출력하는 함수
- ▶ 마지막에 저장된 '\0'를 '\n'로 교체하여 버퍼에 전송

함수 printf()와 scanf()는 다양한 입출력에 적합하며, 문자열 출력 함수 puts()와 gets()는 처리속도가 빠르다는 장점이 있음

문자배열 라이브러리와 문자열 비교

■ 함수 strcmp()

- ▶ 문자열 비교와 복사, 그리고 문자열 연결 등과 같은 다양한 문자열 처리는 헤더파일 string.h에 함수원형으로 선언된 라이브러리 함수로 제공
- ▶ 인자인 두 문자열을 사전상의 순서로 비교하는 함수
 - 비교 방법은 인자인 두 문자열을 구성하는 각 문자를 처음부터 비교해 나간다.
 - 비교 기준은 아스키 코드값이다.
 - 문자가 다른 경우 앞 문자가 작으면 음수, 뒤 문자가 작으면 양수, 같으면 0을 반환한다.
 - 대문자가 소문자보다 아스키 코드값이 작으므로 strcmp("java","javA")는 양수를 반환한다.

```
strcmp(java, java) = 0
strcmp(java, jav = 1
strcmp(jav, java) = -1
strcmp(jav, java, 3) = 0
```


함수 strcmp()를 이용하여
비교

문자열 복사와 연결

■ 함수 strcpy()

- ▶ 함수 strcpy()와 strncpy()는 문자열을 복사하는 함수이다.
- ▶ 함수 strcpy()는 앞 인자 문자열 dest에 뒤 인자 문자열 source를 복사한다.

```
char dest[80] = "Java";  
char source[80] = "C is a language."  
  
printf("%s\n", strcpy(dest, source));  
printf("%s\n", strncpy(dest, "C#", 2));  
printf("%s\n", strncpy(dest, "C#", 3));
```



```
C is a language.  
C# is a language.  
C#
```

3번째 줄은 3바이트까지 복사
하므로 마지막이 널분자가 복
사됨

문자열 분리 및 다양한 문자열 관련 함수

■ 함수 strtok()

- ▶ 문자열에서 구분자인 문자를 여러 개 지정하여 토큰을 추출하는 함수
- ▶ 문장 `ptoken = strtok(str, delimiter);`으로 첫 토큰을 추출
- ▶ 결과를 저장한 `ptoken`이 `NULL`이면 더 이상 분리할 토큰이 없는 경우이다.

구분자

```
char str1[] = "C and C++ language are best!";
char *delimiter = " ,\t!";

printf("문자열 \"%s\" 을 >>\n", str1);
printf("구분자[%s]를 이용하여 토큰을 추출 >>\n", delimiter);
char* ptoken = strtok(str1, delimiter);

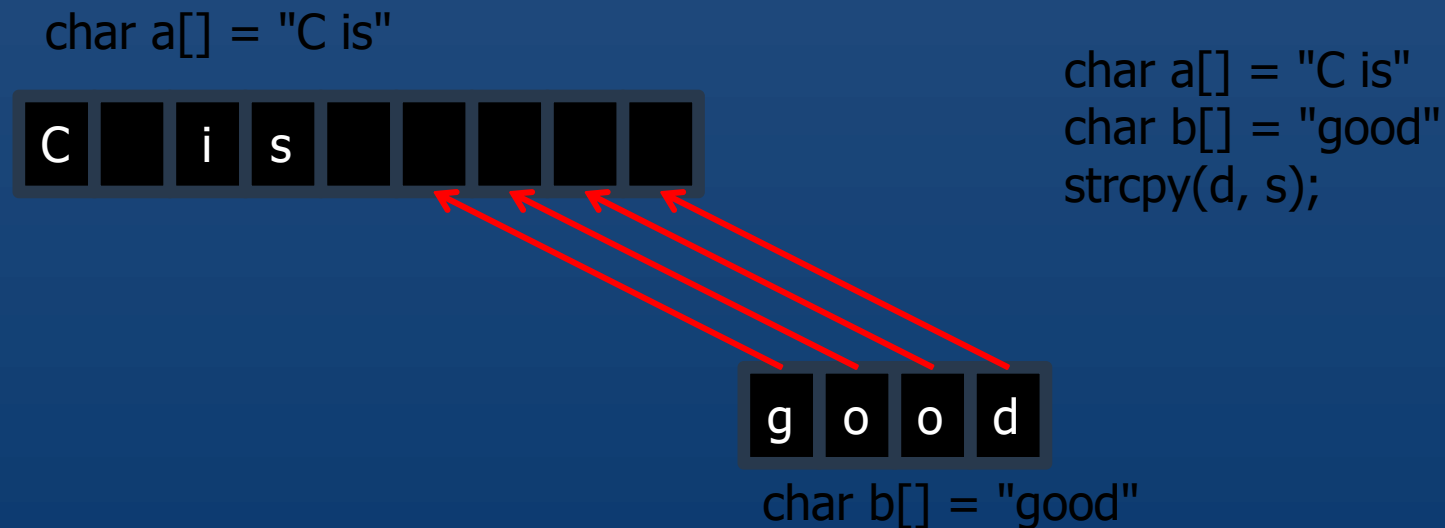
while (ptoken != NULL)
{
    printf("%s\n", ptoken);
    ptoken = strtok(NULL, delimiter);
}
```

```
문자열 "C and C++ language are best!"을 >>
구분자[ , \t !]를 이용하여 토큰을 추출 >>
C
and
C++
language
are
best
```

문자열 복사와 연결

■ 함수 strcat()

- ▶ 앞 문자열에 뒤 문자열의 null 문자까지 연결하여, 앞의 문자열 주소를 반환하는 함수이다.



문자열 분리 및 다양한 문자열 관련 함수

■ 함수 strtok()

- ▶ 함수 strtok()은 문자열에서 구분자인 문자를 여러개 지정하여 토큰을 추출하는 함수
- ▶ str은 문자열 상수를 사용할 수 없다.
- ▶ 문장 ptoken = strtok(str, delimiter);으로 첫 토큰을 추출한다.
- ▶ 결과를 지정한 ptoken이 NULL이면 더 이상 분리할 토큰이 없는 경우이다.

문자 포인터 배열과 이차원 문자 배열

■ 문자 포인터 배열

- ▶ 여러 개의 문자열을 처리하는 하나의 방법
- ▶ 여러개의 문자열을 참조할 수 있다.
- ▶ 각각의 문자열 저장을 위한 최적의 공간을 사용하는 장점을 가진다.

```
#include <stdio.h>

int main(void)
{
    char* abc[] = { "abc", "def", "ghi" };
    printf("%s ", abc[0]); printf("%s ", abc[1]); printf("%s ", abc[2]);
    return 0;
}
```



abc def ghi

문자 포인터 배열과 이차원 문자 배열

■ 이차원 문자 배열

- ▶ 여러 개의 문자열을 처리하는 다른 방법의 배열
- ▶ 벡열선언에서 이차원 배열의 열 크기는 문자열 중에서 가장 긴 문자열의 길이보다 1 크게 지정해야한다.
- ▶ 모든 열 수가 동일하게 메모리에 할당된다
- ▶ 낭비되는 메모리 공간이 있을 수 있다.
- ▶ 문자열을 수정할 수 있는 장점이 있다.

```
#include <stdio.h>

int main(void)
{
    char a[][5] = { "aaa", "bbb", "ccc" };
    printf("%s ", a[0]); printf("%s ", a[1]); printf("%s ", a[2]);
    return 0;
}
```



aaa bbb ccc

명령행 인자

■ `main(int argc, char *argv[])`

- ▶ 프로그램에서 명령행 인자를 받으려면 `main()` 함수에서 두개의 인자 `argc`와 `argv`를 (`int argc, char * argv[]`)로 기술해야 한다.
- ▶ 실행 프로그램 이름도 하나의 명령행 인자에 포함된다.
- ▶ Visual C++에서 명령행 인자를 설정하려면 메뉴 [프로젝트]/ [프로젝트 이름] 속성...]을 누르거나 단축기 `Alt+F7`을 눌러 대화상자에서 설정해야 한다.

연습문제 11-4 문자를 삭제하여 출력

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <string.h>
4
5  int main(void)
6  {
7      char str1[20] = "java";
8      char* ch = "a";
9
10     char* ptoken = strtok(str1, ch);
11
12     while (ptoken != NULL)
13     {
14         printf("%s", ptoken);
15         ptoken = strtok(NULL, ch);
16     }
17
18     return 0;
19
20 }
```

Microsoft Visual Studio 디버그 콘솔

```
jv
C:\Users\uni77\source\repos\
이 창을 닫으려면 아무 키나 누
```

임의의 문자를 하나 지정하여 삭제가 가능하다.

연습문제 11-6 단어를 반대로 출력시키기

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <string.h>
4
5  void reverse(char str[]);
6
7  int main(void)
8  {
9
10     char rmfwk[20];
11
12     printf("%s", "한 단어를 입력하세요. -> ");
13     scanf("%s", rmfwk);
14     reverse(rmfwk);
15     printf("입력한단어를 반대로 출력합니다. -> %12s\n", rmfwk);
16
17
18     return 0;
19 }
20 void reverse(char str[])
21 {
22     for (int i = 0, j = strlen(str) - 1; i < j; i++, j--)
23     {
24         char rmfwk = str[i];
25         str[i] = str[j];
26         str[j] = rmfwk;
27     }
28 }
```

한 단어를 입력하세요. -> programming
입력한단어를 반대로 출력합니다. -> gnimargorp

reverse()를 이용하여 단어를
반대로 출력 할 수 있다.

12장

변수 유효범위

전역변수와 지역변수

■ 변수 scope

- ▶ 변수와 참조가 유효한 범위
- ▶ 지역 유효 범위(local scope)와 전역 유효 범위(global scope)로 나뉜다.
- ▶ 지역 유효 범위는 함수 또는 블록 내부에서 선언되어 그 지역에서 변수의 참조가 가능한 범위
- ▶ 전역 유효 범위는 하나의 파일에서만 변수의 참조가 가능한 범위와 프로젝트를 구성하는 모든 파일에서 변수의 참조가 가능한 범위 두가지로 나뉜다.

변수 범위와 지역변수

■ 지역변수

- ▶ 함수나 블록 내부에서 선언되어 사용하는 변수
- ▶ 함수 또는 블록에서 선언된 변수
- ▶ 함수나 블록에서 지역변수는 선언 문장 이후에 함수나 블록의 내부에서만 사용이 가능
- ▶ 함수의 매개변수도 함수 전체에서 사용 가능한 지역변수와 같다.
- ▶ 선언 후 초기화하지 않으면 쓰레기값이 저장되므로 주의
- ▶ 선언된 부분에서 자동으로 생성되고 함수나 블록이 종료되는 순간 메모리에서 자동으로 제거된다.

전역 변수와 extern

■ 전역변수

- ▶ 함수 외부에서 선언되는 변수
- ▶ 일반적으로 프로젝트의 모든 함수나 블록에서 참조 할 수 있다.
- ▶ 자동으로 초기값이 자료형에 맞는 0으로 지정된다.
- ▶ 함수나 블록에서 전역변수와 같은 이름으로 지역변수를 선언할 수 있다.
- ▶ 프로젝트의 다른 파일에서도 참조가 가능하다
- ▶ 다른파일에서 선언된 전역변수를 참조하려면 키워드 extern을 사용하여야 한다..
- ▶ 어디에서든지 수정할 수 있으므로 사용이 편한 장점이 있다.
- ▶ 예상하지 못한 값이 저장된다면 프로그램 어느 부분에서 수정되었는지 알기 어려운 단점이 있다.

기억부류와 레지스터 변수

■ auto, register, static, extern

- ▶ auto, register, static, extern에 따라 할당되는 메모리 영역이 결정되고 메모리의 할당과 제거 시기가 결정된다.
- ▶ auto와 register는 지역변수에만 이용이 가능하고 static은 지역과 전역 모든 변수에 이용가능하며 extern은 전역변수에만 사용가능 하다.
- ▶ 키워드 extern을 제외하고 나머지 3개의 기억부류의 변수선언에서 초기 값을 저장가능하다.

| 기억부류 | 전역 | 지역 |
|----------|----|----|
| auto | X | O |
| register | X | O |
| static | O | O |
| extern | O | X |

기억부류와 레지스터 변수

■ 키워드 register

- ▶ 체커 내부의 레지스터에 할당되는 변수
- ▶ 키워드 register를 자료형 앞에 넣어 선언한다.
- ▶ 지역변수에만 이용이 가능하다.
- ▶ 일반 메모리에 할당되는 변수가 아니므로 주소연산자&을 사용할 수 없다.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    register int sum = 0;

    int max;
    printf("양의 정수 입력 >> ");
    scanf("%d", &max);

    for (register int count = 1; count <= max; count++)
        sum += count;

    printf("합: %d\n", sum);

    return 0;
}
```

레지스터 지역변수 선언

레지스터 블록 지역변수 선언

정적변수

■ 키워드 static

- ▶ 변수 선언에서 자료형 앞에 static을 넣어 정적변수를 선언 가능
- ▶ 초기 생성된 이후 메모리에서 제거되지 않으므로 지속적으로 저장값을 유지하거나 수정 할 수 있다.
- ▶ 초기값을 지정하지 않으면 자동으로 자료형에 따라 0이나 '\0' 또는 NULL값이 저장된다.
- ▶ 초기화는 상수로만 가능하다.

```
static int index = 1;
```

→ 정적변수 선언

정적변수

■ 정적 지역변수

- ▶ 함수나 블록에서 정적으로 선언되는 변수
- ▶ 함수나 블록을 종료해도 메모리에서 제거되지 않고 계속 메모리에 유지 관리되는 특성이 있다.

```
#include <stdio.h>

void increment(void);

int main(void)
{
    for (int count = 0; count < 3; count++)
        increment();
}

void increment(void)
{
    static int sindex = 1;
    auto int aindex = 1;

    printf("정적 지역변수 sindex: %2d, \t", sindex++);
    printf("자동 지역변수 aindex: %2d\n", aindex++);
}
```

| | |
|--------------------|-------------------|
| 정적 지역변수 sindex: 1, | 자동 지역변수 aindex: 1 |
| 정적 지역변수 sindex: 2, | 자동 지역변수 aindex: 1 |
| 정적 지역변수 sindex: 3, | 자동 지역변수 aindex: 1 |

정적 지역변수는 다음 호출에서 계속 사용되므로 숫자가 1씩 증가하지만 자동 지역변수는 1을 증가시키나 증가된 aindex를 사용할 일이 없다.

정적 지역변수

자동 지역변수

정적변수

■ 정적 전역변수

- ▶ 함수 외부에서 정적으로 선언되는 변수
- ▶ 선언된 파일 내부에서만 참조가 가능한 변수이다.
- ▶ 프로그램이 크고 복잡하면 전역변수의 사용은 원하지 않는 전역변수의 수정과 같은 부작용의 위험성이 항상 존재한다.

메모리 영역

■ 데이터, 스택, 힙 영역

- ▶ 메인 메모리의 영역은 프로그램 실행 과정에서 데이터영역, 힙,영역, 스택 영역 으로 나뉜다.
- ▶ 변수의 유효범위와 생존기간에 결정적 역할을 한다
- ▶ 변수는 기억부류에 따라 할당되는 메모리 공간이 달라진다.
- ▶ 데이터 영역은 전역변수와 정적변수가 할당되는 저장공간이다
- ▶ 힙 영역은 동적할당 되는 변수가 할당되는 저장공간이다.
- ▶ 스택 영역은 함수 호출에 의한 형식 매개변수 그리고 함수 내부의 지역변수가 할당되는 저장공간이다.

변수의 이용

■ 이용기준

- ▶ 실행속도를 개선하고자 하는 경우에는 레지스터 변수를 이용한다.
- ▶ 함수나 블록 내부에서 함수나 블록이 종료되더라도 계속적으로 값을 저장하고 싶을 때는 정적지역변수를 이용한다
- ▶ 해당파일 내부에서만 변수를 공유하려면 정적 전역변수를 이용한다.

변수의 종류

| 선언위치 | 상세 종류 | 키워드 | | 유효범위 | 기억장소 | 생존기간 |
|------|---------|----------------|--------|--------------|-----------------|------------------|
| 전역 | 전역 변수 | 참조선언 | extern | 프로그램 전역 | 메모리 (데이터 영역) | 프로그램 실행 시간 |
| | 정적 전역변수 | static | | 파일 내부 | | |
| 지역 | 정적 지역변수 | static | | 함수나 블록 내부 | 레지스터 | 함수 또는 블록 실행시간 |
| | 레지스터 변수 | register | | | 메모리 (스택 영역) | |
| | 자동 지역변수 | auto (생략가능) | | | | |

변수의 유효 범위

| 구분 | 종류 | 메모리할당시 기 | 동일파일 외부 함수에서의 이 용 | 다른파일 외부 함수에서의 이 용 | 메모리제거 시 기 |
|----|---------|--------------|-------------------------|-------------------------|--------------|
| 전역 | 전역변수 | 프로그램 시작 | O | O | 프로그램 종료 |
| | 정적 지역변수 | | O | X | |
| 지역 | 레지스터 변수 | | X | X | |
| | 자동지역변수 | 함수(블록)시 작 | X | X | 함수(블록)종 료 |
| | 자동지역변수 | | X | X | |

변수의 초기값

| 구분 | 종류 | 자동 저장되는 기본 초기값 | 초기값 저장 |
|----|---------|--------------------------------|-------------------|
| 전역 | 전역변수 | 자료형에 따라 0이나 'W0' 또는 NULL값이 저장됨 | 프로그램 시작 시 |
| | 정적 전역변수 | | |
| 지역 | 정적 전역변수 | 쓰레기값이 저장됨 | 함수나 블록이 실행 될 때 마다 |
| | 레지스터 변수 | | |
| | 자동 지역변수 | | |

연습문제 12-6 (1에서100까지 임의의 수 맞추기) 첫번째

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

int number;
int user;
int trycount;
static int min = 1;
static int max = 100;
void setNumber(void);
void printHead(void);
void printHigher(void);
void printLower(void);
void printAnswer(void);

int main() {
    setNumber();
    printHead();
    while (1) {
        trycount++;
        if (user > number) {
            printHigher();
        }
        else if (user < number) {
            printLower();
        }
        else {
            printAnswer();
            break;
        }
        if (trycount == 4) {
            printf("5번의 기회를 모두 사용했습니다. 난수 값 = %d 입니다.\n", number);
            break;
        }
    }
}
```

연습문제 12-6 (1에서100까지 임의의 수 맞추기) 두번째

```
void printAnswer() {
    printf("축하합니다! 정답은 %d 입니다.\n", user);
}

void printHigher() {
    max = user - 1;
    printf("%d. 맞추어야 할 정수가 입력한 정수 %d 보다 작습니다.\n", trycount, user);
    printf("%d 에서 %d 사이의 정수를 다시 입력하세요. > ", min, max);
    scanf_s("%d", &user, sizeof(int));
    puts("");
}

void printLower(void)
{
}

void printLower() {
    min = user + 1;
    printf("%d. 맞추어야 할 정수가 입력한 정수 %d 보다 큼니다.\n", trycount, user);
    printf("%d 에서 %d 사이의 정수를 다시 입력하세요. > ", min, max);
    scanf_s("%d", &user, sizeof(int));
    puts("");
}

void setNumber() {
    long seconds = (long)time(NULL);
    srand(seconds);
    number = rand() % 100 + 1;
}
```

연습문제 12-6 (1에서100까지 임의의 수 맞추기) 세번째

```
void printHead(void)
{
}

void printHead() {
    printf("%d 에서 %d 까지의 하나의 정수가 결정되었습니다. 이 정수를 맞추어 보세요? >", min, max);
    scanf_s("%d", &user, sizeof(int));
    puts("");
}
```

결과창

1 에서 100 까지의 하나의 정수가 결정되었습니다.
이 정수를 맞추어 보세요? >50

1. 맞추어야 할 정수가 입력한 정수 50 보다 작습니다.
1 에서 49 사이의 정수를 다시 입력하세요. > 25

2. 맞추어야 할 정수가 입력한 정수 25 보다 작습니다.
1 에서 24 사이의 정수를 다시 입력하세요. > 12

3. 맞추어야 할 정수가 입력한 정수 12 보다 큼니다.
13 에서 24 사이의 정수를 다시 입력하세요. > 18

4. 맞추어야 할 정수가 입력한 정수 18 보다 작습니다.
13 에서 17 사이의 정수를 다시 입력하세요. > 15

5번의 기회를 모두 사용했습니다. 난수 값 = 16 입니다.

1 에서 100 까지의 하나의 정수가 결정되었습니다.
이 정수를 맞추어 보세요? >50

축하합니다! 정답은 50 입니다.

13장

구조체와 공용체

구조체 개념과 정의

■ 구조체 개념

- ▶ 정수나 문자, 실수나 포인터 등을 묶어 하나의 자료형으로 이용하는 것
- ▶ 연관성이 있는 서로 다른 개별적인 자료형의 변수들

■ 유도 자료형

- ▶ 연관된 멤버로 구성되는 통합 자료형

■ 구조체 정의

- ▶ 키워드 struct 다음에 구조체 태그이름을 기술하고 중괄호를 이용하여 원하는 여러개의 변수로 선언
- ▶ 구조체를 구성하는 하나하나의 항목을 구조체의 멤버 또는 필드라고 한다.

구조체 변수 선언과 초기화

■ 구조체 변수 선언

- ▶ 자료형 struct account형 변수 mine을 선언하려면 struct account mine;으로 선언한다.

■ 구조체 변수선언

struct 구조체태그이름 변수명; 으로 설정한다.

```
int main(void) {  
    struct abc me;  
    struct abc you1, you2, you3;  
}
```

구조체 자료형 변수 선언 및 초기화

구조체 변수 선언과 초기화

■ 구조체 변수의 초기화

- ▶ 초기화 값은 다음과 같이 중괄호 내부에서 구조체의 각 멤버 정의 순서대로 초기값을 쉼표로 구분하여 기술
- ▶ 배열과 같이 초기값에 기술되지 않은 멤버값은 자료형에 따라 기본값인 0, 0.0, '\0' 등으로 저장

```
struct abc  
{  
    char name[10];  
    int actnum;  
    double balance;  
};  
  
struct abc me = { '박윤석', 100, 10000 }
```

구조체태그이름

변수명

초기값

구조체 변수의 초기화

구조체 변수 선언과 초기화

■ 구조체의 멤버 접근 연산자 . 와 변수 크기

- ▶ 선언된 구조체형 변수는 접근연산자 .를 사용하여 멤버를 참조 할 수 있다.
- ▶ 일반적으로 컴파일러는 시스템의 효율성을 위하여 구조체 크기를 산술적인 구조체의 크기보다 크게 할당할 수 있다.
- ▶ 시스템은 정보를 4바이트 혹은 8바이트 단위로 전송 처리하므로 이에 맞도록 메모리를 할당하다 보면 중간에 사용하지 않는 바이트를 삽입할 수 있다.
- ▶ 실제 구조체의 크기는 멤버의 크기의 합보다 크거나 같다.

공용체 활용

■공용체 개념

- ▶ 동일한 저장 장소에 여러 자료형을 저장하는 방법
- ▶ 한번에 한 종류만 저장하고 참조가능

■union을 사용한 공용체 정의 및 변수 선언

- ▶ 서로 다른 자료형의 값을 동일한 저장공간에 저장하는 자료형
- ▶ 공용체 변수의 크기는 멤버 중 가장 큰 자료형의 크기로 정해짐

```
union abc
{
    char q;    문자형
    int  r;    정수형
    double e; 실수형
}abcd;
```

```
union qwer
{
    char a[5]; char형 배열
    int b;      정수형
    double c;   실수형
};
```

구조체 정의

자료형 재정의 typedef

■ typedef의 구문

- ▶ typedef는 이미 사용되는 자료 유형을 다른 새로운 자료형 이름으로 재정의 할 수 있도록 만들어준다.
- ▶ typedef int profit;은 profit을 int와 같은 자료형으로 새롭게 정의한느 문장이다.
- ▶ 자료형을 재정의 하는 이유는 프로그램의 시스템 간 호환성과 편의성을 위해 필요

```
typedef unsigned int budget; → int를 재정의

int main(void){
    budget year = 24500000; 재정의를 하여 int를 붙이지 않음
    typedef int profit;
    profit month = 4600000;
    printf("올 예산은 %d, 이달의 이익은 %d 입니다.\n", year, month);
    return 0;
}

void test(void){
    budget year = 24500000;
}
```

올 예산은 24500000, 이달의 이익은 4600000 입니다.

구조체 자료형 재정의

■ struct를 생략한 새로운 자료형

```
struct date
{
    int year;
    int month;
    int day;
};

typedef struct date date;
```

구조체 struct date를 date로 재정의

```
typedef struct
{
    char title[30];
    char company[30];
    char kinds[30];
    date release;
} software;
```

typedef구문에서 software형을 정의

구조체 포인터

■ 포인터 변수 선언

- ▶ 포인터는 각각의 자료형 저장 공간의 주소를 저장하듯이 구조체 포인터는 구조체의 주소값을 저장 할 수 있는 변수이다.

```
lecture os = { "운영체제", 2, 3, 3 };  
lecture c = { "C프로그래밍", 3, 3, 4 };  
lecture *p = &os;
```

변수 os를 문장선언

lecture 포인터 변수p에 os를 저장

구조체 포인터

■ 포인터 변수의 구조체 멤버 접근 연산자 ->

- ▶ 연산자 ->와 . 은 우선순위 1위이고 결합성은 좌에서 우이며, 연산자 *은 우선순위 2위이고 결합성은 우에서 좌이다.

| 접근 연산식 | 구조체 변수 os 와 구조체 포인터 변수 p 인 경우의 의미 |
|--------------------------|---|
| <code>p->name</code> | 포인터 <code>p</code> 가 가리키는 구조체의 멤버 <code>name</code> |
| <code>(*p).name</code> | 포인터 <code>p</code> 가 가리키는 구조체의 멤버 <code>name</code> |
| <code>*p.name</code> | <code>*(p.name)</code> 이고 <code>p</code> 가 포인터이므로 <code>p.name</code> 은 문법오류가 발생함 |
| <code>*os.name</code> | <code>*(os.name)</code> 를 의미하며 한글인 경우에는 실행 오류가 발생함 |
| <code>*p->name</code> | <code>*(p->name)</code> 을 의미하며, 포인터 <code>p</code> 가 가리키는 구조체의 멤버 <code>name</code> 이 가리키는 변수이다. |

구조체 포인터

■ 공용체 포인터

- ▶ 공용체 변수도 포인터 변수 사용이 가능하다.
- ▶ 공용체 포인터 변수로 멤버를 접근하려면 접근연산자 ->를 이용한다.

```
union data
{
    char ch;
    int cnt;
    double real;
} value, *p;
```

```
p = &value;
```

```
p->ch = 'a';
```

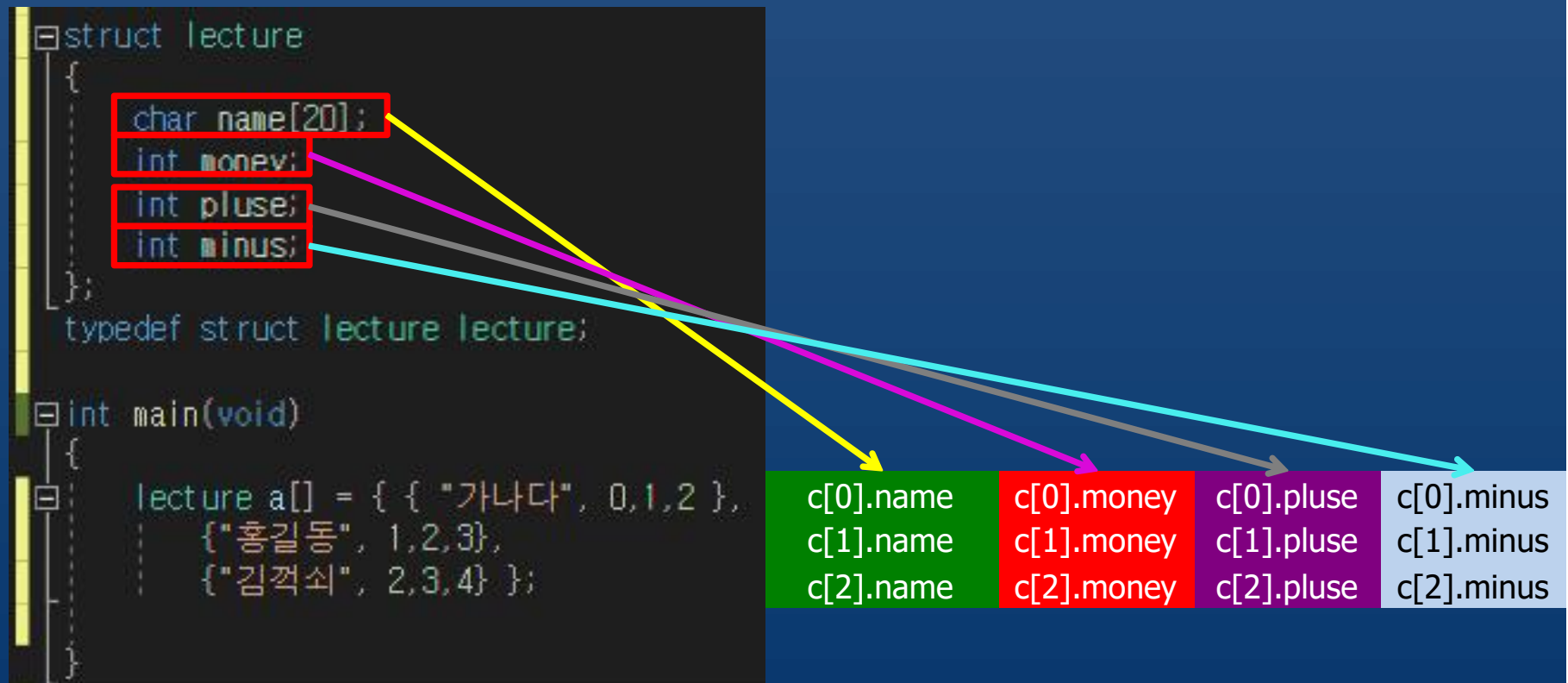
→ 포인터 p에 value의 주소값을 저장

→ value.ch='a';와 같음

구조체 배열

■ 구조체 배열 변수 선언

- ▶ 다른 배열과 같이 동일한 구조체 변수가 여러개 필요하다면 구조체 배열을 선언하여 이용할 수 있다.
- ▶ 구조체 배열의 초기값 지정 구문에서는 중괄호가 중첩되게 나타난다.



연습문제 13-1 (각각의 분자와 분모 입력 및 출력)

```
#include <stdio.h>

typedef struct {
    int numerator;
    int denominator;
}fraction;

int main(void)
{
    fraction a;
    fraction b;

    printf("첫번째 분자와 분모를 입력하세요 : ");
    scanf_s("%d %d", &a.numerator, &a.denominator, sizeof(int));

    printf("두번째 분자와 분모를 입력하세요 : ");
    scanf_s("%d %d", &b.numerator, &b.denominator, sizeof(int));

    printf("a = %d/%d, b = %d/%d", a.numerator, a.denominator, b.numerator, b.denominator);
}
```

결과창

```
첫번째 분자와 분모를 입력하세요 : 1 2
두번째 분자와 분모를 입력하세요 : 3 4
a = 1/2, b = 3/4
```


연습문제 13-2 (13-1에서 구한 두 분수의 곱을 출력)

```
#include <stdio.h>

typedef struct {
    int numerator;
    int denominator;
} fraction;

int main(void)
{
    fraction a;
    fraction b;

    printf("첫번째 분자와 분모를 입력하세요 : ");
    scanf_s("%d %d", &a.numerator, &a.denominator, sizeof(int));

    printf("두번째 분자와 분모를 입력하세요 : ");
    scanf_s("%d %d", &b.numerator, &b.denominator, sizeof(int));

    printf("a = %d/%d, b = %d/%d \n", a.numerator, a.denominator, b.numerator, b.denominator);

    printf("%d/%d * %d/%d의 결과는 %d/%d \n", a.numerator, a.denominator, b.numerator, b.denominator,
        a.numerator * b.numerator, a.denominator * b.denominator);
}
```

```
첫번째 분자와 분모를 입력하세요 : 1 2
두번째 분자와 분모를 입력하세요 : 3 4
a = 1/2, b = 3/4
1/2 * 3/4의 결과는 3/8
```

여기까지 읽어주셔서
감사합니다

Thank You