

perfect c

포트폴리오



20193434

홍연

목차

1.문자와 문자열

-설명

-LAB

-연습문제

2.변수 유효범위

-설명

-LAB

-연습문제

3.구조체와 공용체

-설명

-LAB

-연습문제

4. 맷음말

11.문자와 문자열

문자란?

영어의 알파벳이나 한글의 한 글자를 작은 따옴표로 둘러싸서 'A'와 같이 표기, C 언어에서 1 바이트인 자료형 char 로 지원한다. 작은따옴표에 의해 표기된 문자를 **문자상수**라고 한다.

문자열이란? 문자들의 모임. 앞뒤로 큰따옴표를 사용 "JAVA"로 표시한다.

문자와 문자열의 선언

- 문자열을 저장하려면 문자의 모임인 **문자배열**을 사용해야한다.
- 주의할점은 문자열의 마지막을 의미하는 NULL 문자 '\0'가 마지막에 저장되어야한다.
따라서 문자열이 저장되는 배열 크기는 반드시 저장될 문자수보다 1 커야한다.
- 편리한 방법으로는 배열 선언시 저장할 큰따옴표를 사용하여 문자열 상수를 바로 대입하는 방법이다.
- 배열 초기화 시 배열크기는 지정하지 않는 것이 더 편리하며, 만일 지정한다면 마지막 문자인 '\0'을 고려해 실제 문자 수보다 1 이 더 크게 배열크기를 지정해야 한다.
- 지정한 배열크기가 (문자수+1)보다 크다면 나머지 부분은 모두 '\0'문자로 채워진다.
- 배열크기가 작으면 문자열 상수가 아닌 단순한 문자배열이 되므로, 문자열 출력 등에서 문제가 발생한다.

```
Char ch = 'A'; //ch 에 문자 A 를 저장
```

```
Char csharp[3]; // 배열 크기는 저장될 문자수보다 1 커야한다.
```

```
Csharp [0] = 'C'; csharp[1] = '#'; csharp[2] = '\0'; // "C#"을 저장하려면  
마지막원소
```

에 \0 을 저장해야 한다.

```
Char java [] ={'J','A','V','A','\0'}; // 마지막에 \0 을 빼면, 대입시에 문제는
```

함수 *printf* 를 사용한 문자와 문자열 출력

함수 *printf()*에서 %c 는 문자를 출력, %s 는 문자열을 출력한다. *puts(csharp)*과 같이사용하면 한줄에 문자열을 출력한후 다음 줄에서 출력을 준비한다. *printf(c)*와같이 바로 배열을 인자로 사용해도 출력이 가능하다.

문자열을 구성하는 문자참조

문자열 상수를 문자포인터에 저장하는 방식이다. 형식제어 문자 %s 로 간단히 처리할 수 있다. 이와 같이 문자 포인터에 의한 선언으로는 문자 하나하나의 수정은 할수 없다.

'\0'문자에 의한 문자열 분리

*printf()*에서 %s 는 문자 포인터가 가리키는 위치에서 null 문자 '\0'를 하나의 문자열로 인식한다.

다양한 문자 입출력

문자의 입력에 사용되는 함수는 *getchar()*이고 출력에 사용되는 함수는 *putchar()*이다.

함수 *getche()*

- 함수는 버퍼를 사용하지 않고 문자 하나를 바로바로 입력할 수 있는 함수이다.
- 함수를 이용하려면 헤더파일 *conio.h* 를 사용해야한다.

함수 *getch()*

- 함수 *getch()*도 버퍼를 사용하지 않는 문자 입력 함수이다.
- 함수 *getch()*도 *conio.h* 파일에 함수원형이 정의되어 있어 사용하려면 *conio.h* 를 삽입해야 한다.

- 여러 컴파일러에서 함수 getch(),getch()는 _getch(),_getch()로 이름이 수정되어 사용하고 있다.

함수	Scanf("%c",&ch)	Getchar()	Getche() _getche()	Getch() _getch()
헤더파일	Stdio.h		Conio.h	
버퍼이용	O		X	
반응	[ENTER]키를 눌러야 작동		문자 입력마다 바능	
문자입력의 표시(echo)	누르면 바로 표시		누르면 바로 표시	표시 안됨
입력문자 수정	가능		불가능	

<문자입력함수 scanf(), getchar(), getche(). Getch()의 비교>

문자 배열 변수로 scanf()에서 입력

Gets

- 함수 gets()는 한 행의 문자열 입력에 유용한 함수이다. gets_s()의 대체함수로 사용을 권장한다
- .gets()의 유의점으로는 마지막에 입력된 'wn'가 'w0'으로 교체되어 인자인 배열에 저장된다.

Puts

- 함수 puts()는 한 행에 문자열을 출력하는 함수이다.오류가 발생하면 EOF 를 반환하는데 EOF 는 (End Of File)파일의 끝이라는 의미로 stdio.h 헤더파일에 정수 -1 로 정의되어있다.

- 함수 puts 는 gets 와는 반대로 문자열 마지막에 저장된 '\0'을 '\n'으로 반환하여 전송한다. 그러므로 문자열의 한 행 출력 함수는 puts()가 더 효과적이다.

printf()와 scanf()는 다양한입출력에, gets()와 puts()는 문자열 입출력에 사용하면 처리가 빠르다.

다양한 문자열 라이브러리 함수

string.h 에 함수원형으로 선언된 라이브러리 함수로 제공된다.

이름	설명
STRCMP()	인자인 두 문자열을 사전 상의 순서로 비교하는 함수
STRNCMP()	두 문자를 비교할 문자의 최대 수를 지정하는 함수
STRCPY(), STRNCPY()	문자열을 복사하는 함수 strcpy()는 앞인자 문자열 dest 에 뒤 문자열 source 를 복사
STRCAT()	앞 문자열 뒤에 뒤 문자열의 null 문자까지 연결하여 앞의 문자열 주소를 반환
STRTOK()	문자열에서 구분자인 문자를 여러개 지정하여 토큰을 추출하는 함수
STRLWR()	인자를 모두 소문자로 반환
STRUPR()	인자를 모두 대문자로 반환한다.
STRLEN()	NULL 문자를 제외한 문자열 길이를 반환.

문자 포인터 배열

각각 문자열 저장을 위한 최적의 공간을 사용하기위한 방법중 하나는 포인터 배열을 이용하는 방법이다. 하지만 문자열 상수의 수정은 불가하다.

이차원 문자 배열

배열선언에서 열크기는 문자열중 가장 긴길이 +1 로 지정한다.

```
Char ca[] [5] ={"JAVA","C#","C++"}; // ca[] 는 단어 행의 개수(무한), [5]는 글자수(5)
//각각의 3 개 문자열 출력
Printf("%s ", ca[0]); printf("%s",ca[1]);printf("%s\n",ca[2]);
```

명령행 인자

```
main(int argc,char,*argv[])
```

프로그램에서 명령행 인자를 받으려면 main()함수에서 두 개의 인자 argc 와 argv 를

intargc,char*argv[]로 기술해야한다.

매개변수 argc 는 명령행에서 입력한 수의 문자열수이며 argv[]는 명령행에서 입력한 문자열 전달받는 문자포인터배열이다.

여기서 주의할점은 실행프로그램 이름도 하나의 명령행 인자에 포함된다는 사실이다.

LAB 11-3

<문제>

여러 문자열을 각각의 일차원 문자배열 STR1,STR2,STR3 에 저장한 후, 문자 포인터배열 PTR 을 선언 하면서 문자배열이름을 초기화로 저장한다. 변수 STR1, STR2,STR3 과 PTR 을 사용해 저장된 문자열과 문자를 적절히 출력해보도록 한다.

-char 일차원 배열 str1, str2, str3 을 선언하면서 문자열 "JAVA" "C#","C++"를 저장

-char 포인터 배열 ptr 을 선언하면서 문자열 포인터인 str1, str2, str3 을 저장

<코딩>

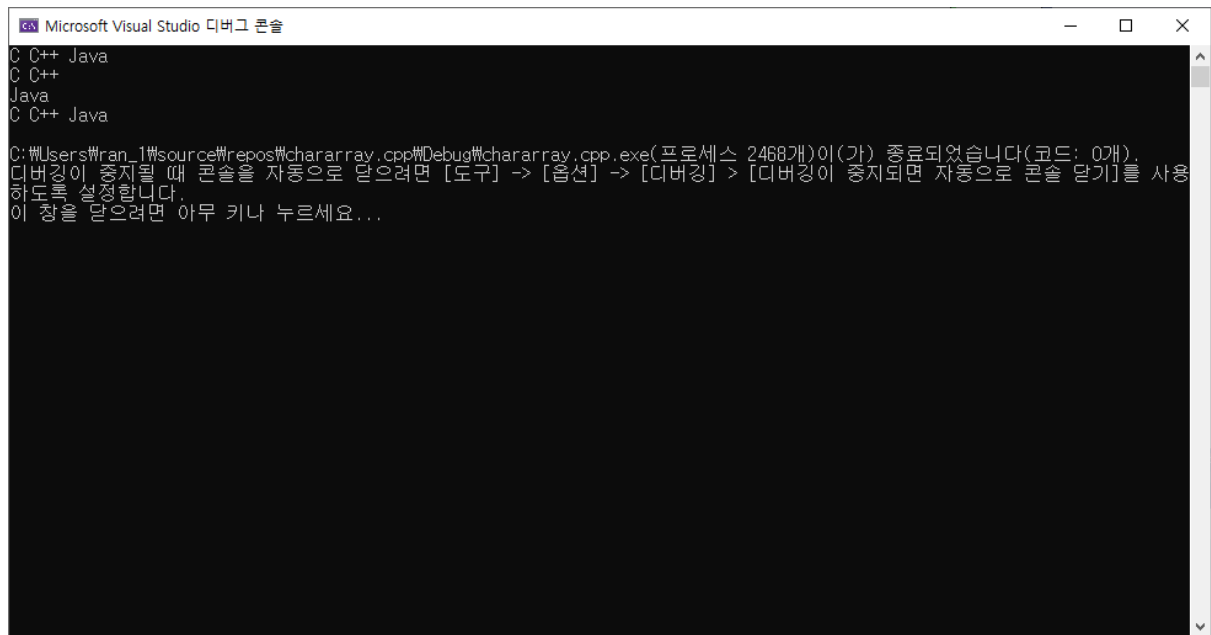
```
#include <stdio.h>

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

    c[5] = ' ';
    char* p = c;
    while (*p)
        printf("%c", *p++);
    printf("\n");

    return 0;
}
```

<실행화면>



프로그래밍 연습

08.한 줄의 문자열을 표준입력으로 입력받아 단어의 문자를 역순으로 출력하는 프로그램을 작성하시오.

```
#include <stdio.h>
#include <string.h>

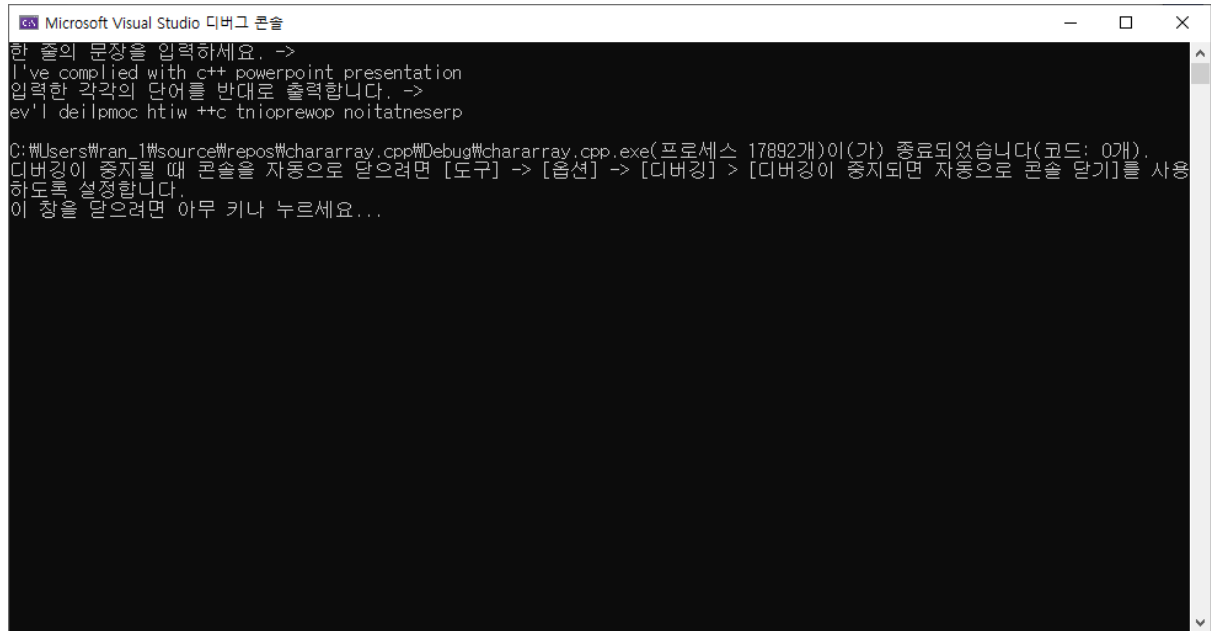
int main(void)
{
    char line[100];
    int size, i, j, k = 0;
    puts("한 줄의 문장을 입력하세요. ->");
    gets_s(line);
    size = strlen(line);
    puts("입력한 각각의 단어를 반대로 출력합니다. ->");
    for (i = 0; i <= size; ++i) {

        if (line[i] == ' ' || i == size) {

            for (j = i - 1; j >= k; --j)
                putchar(line[j]);
            k = i + 1;
            printf(" ");
        }
    }

    printf("\n");
    return 0;
}
```

}



The screenshot shows the 'Microsoft Visual Studio 디버깅 콘솔' (Microsoft Visual Studio Debugger Console) window. The console has a black background with white text. The text in the console is as follows:

```
한 줄의 문장을 입력하세요. ->
I've compiled with c++ powerpoint presentation
입력한 각각의 단어를 반대로 출력합니다. ->
ev'I deilpmoc htiw ++c tnioprewop noitatneserp

C:\Users\ran_1\source\repos\chararray.cpp\Debug\chararray.cpp.exe(프로세스 17892개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

12.변수 유효범위

변수 scope

변수의 참조가 유효한 범위를 변수의 유효범위 (scope)라고 한다.

지역유효범위(local scope)	전역유효범위(global scope)	
함수 또는 블록내부에 선언. 지역내 변수의 참조가 가능한 범위	하나의 파일에서만 변수의 참조가 가능한 범위	프로젝트를 구성하는 모든파일에서 참조가 가능한 범위

지역변수

지역변수	전역변수
함수 또는 블록에서 선언된 변수. 내부변수 또는 자동변수라고 부른다.	함수 외부에서 선언되는 변수, 외부변수라고 부른다.
선언문장 이후 함수나 블록의 내부에서만 사용 가능.(단 다른함수나 블록에선 사용 불가)	일반적으로 프로젝트의 모든 함수나 블록에서 참조 가능
함수의 매개변수도 함수전체에서 사용가능한 지역변수와 같다.	함수나 블록에서 전역변수와 같은이름으로 지역변수 선언 가능하다.
선언후 초기화하지 않으면 쓰레기값이 저장된다	초기값은 자료형에 맞는 0 으로 지정, 정수형은 0, 문자형은 null,등

Extern

Extern 을 사용한 참조선언 구문은 변수선언 문장 맨 앞에 extern 을 넣는 구조이다. Extern 참조선언 구문에서 자료형은 생략가능하다. 키워드 extern 을 사용한 변수 선언은 새로운 변수를 선언하는 것이 아니며, 단지 이미 존재하는 전역변수의 유효범위를 확장하는 것이다. 변수 선언 위치에 따라 변수는 전역/지역으로 나뉘는데, 마찬가지로 변수는 4 가지의 기억부류(storage class)인,auto, register, static, extern 에따라 할당되는 메모리 영역이 결정되고 메모리의 할당과 제거시기가 결정된다

기억부류 종류	전역	지역	초기값 저장
Auto	X	O	O
Register	X	O	O
Static	O	O	O
Extern	O	X	X

Register

- 레지스터 변수는 변수의 저장공간이 일반 메모리가 아니라 CPU 내부의 레지스터에 할당되는 변수이다.
- 레지스터 변수는 키워드 register 를 자료형 앞에 넣어 선언한다.
- 레지스터 변수는 지역변수에만 이용이 가능하다. 즉 레지스터 변수도 지역변수로서 함수나 블록이 시작되면서 cpu 의 내부 레지스터에 값이 저장되고, 함수나 블록을 빠져나오면서 소멸되는 특성을 갖는다.
- 레지스터는 cpu 내부에 있는 기억장소이므로 일반 메모리보다 빠르게 참조될 수 있다. 그러므로 레지스터 변수는 처리 속도가 빠른장점이 있다.
- 레지스터 변수는 일반 메모리에 할당되는 변수가 아니므로 주소연산자 &를 사용할수 없다.

그러므로 레지스터 변수에는 주소연산자 &를 사용하면 문법오류가 나타난다.

Static

- 변수선언 자료형 앞에 키워드 static 을 넣어 정적변수 (static variable)를 선언할 수 있다.
- 정적변수는 초기 생성된 이후 메모리에서 제거되지 않으므로 지속적으로 저장값을 유지하거나 수정할 수 있는 특성이 있다.
- 정적변수는 프로그램이 시작되면 메모리에 할당되고, 프로그램이 종료되면 메모리에서 제거된다.
- 정적변수는 초기값을 지정하지 않으면 자동으로 자료형에 따라 0 이나 'w0' 또는 null 값이 저장된다.
- 정적변수의 초기화는 단 한번만 수행된다. 그러므로 한번 초기화된 정적변수는 프로그램 실행 중간에 더 이상 초기화 되지 않는 특성을 갖는다. 주의할 점은 초기화는 상수로만 가능하다는 것이다.

정적 지역변수

함수나 블록에서 정적으로 선언되는 변수가 지역변수이다. 정적 지역변수의 유효범위는 선언된 블록 내부에서만 참조 가능하다. 정적 지역변수는 함수나 블록을 종료해도 메모리에서 제거되지 않고 계속 메모리에 유지관리되는 특성이 있다.

정적 전역변수

함수 외부에서 정적으로 선언되는 변수가 정적 전역변수이다. 일반 전역변수는 파일 소스가 다르더라도 extern 을 사용하여 참조가 가능하다. 그러나 정적 전역변수는 선언된 파일 내부에서만 참조가 가능한 변수이다. 즉 정적 전역변수는 extern 에 의해 다른 파일에서 참조가 불가능하다. 프로그램이 크고 복잡하면 전역변수의 사용은 원하지 않는 전역변수의 수정과 같은 부작용(side effect)의 위험성이 항상 존재한다. 그러므로 가급적

전역변수의 사용을 자제하는 것이 좋으며, 부득이 전역변수를 이용하는 경우에는 파일에서만 전역변수로 이용할 수 있는 정적 전역변수를 이용하는 것이 바람직하다.

메모리/영역

메인 메모리의 영역은 프로그램 실행 과정에서 데이터(data)영역, 힙(heap)영역, 스택(stack)영역 세 부분으로 나뉜다. 이러한 메모리 영역은 변수의 유효범위 (scope)와 생존기간(life time)에 결정적 열할을 하며, 변수 기억부류(storage class)에 따라 할당되는 메모리 공간이 달라진다.

이름	설명	선언	참조	저장장소	기본값
Auto	함수 내부에서 사용되는 지역변수	Auto 자료형 변수이름;	선언된 함수나 블록내부에서 참조	주기억장치	쓰레기값
Extern	파일의 모든 함수에서 사용되는 전역변수	Extern 자료형 변수이름;	여러 프로그램 모듈에서 참조	주기억장치	0
Register	레지스터에 저장된 지역 변수	Register 자료형 변수이름;	선언된 함수나 블록 내부에서 참조	CPU 의 레지스터	쓰레기값
Static	함수를 실행한 후 호출한 곳으로 이동하더라도 저장값이 그대로 남아있는 변수	Static 자료형 변수이름;	선언된 함수나 블록내부에서 지역적으로 선언된 프로그램 파일내부에서 전역적으로	주기억장치	0

변수의 이용

일반적으로 전역변수의 사용을 자제하고 지역변수를 주로 이용한다. 그러나 다음의 경우에는 그 특성에 맞는 변수를 이용한다.

- 실행속도를 개선하고자 하는 경우에 제한적으로 특수한 지역변수인 레지스터 변수를 이용한다.

- 함수나 블록 내부에서 함수나 블록이 종료되더라도 계속적으로 값을 저장하고 싶을 때는 정적 지역변수를 이용한다.
- 해당 파일 내부에서만 변수를 공유하고자 하는 경우는 정적 전역변수를 이용한다.
- 프로그램의 모든 영역에서 값을 공유하고자 하는 경우는 전역변수를 이용한다. 가능하면 전역변수의 사용을 줄이는 것이 프로그램의 이해를 높일 수 있으며 발생할 수 있는 프로그램 문제를 줄일 수 있다.

LAB 12-3

은행 계좌의 입출금을 구현하기 위해 전역변수 total 과 두 함수 save()와 withdraw()의 정적 지역변수 amount 를 사용하여 몇 개의 입출금에 대해 다음 그림과 같이 출력해 보도록 한다.

- 전역변수 total 에는 초기 금액과 계좌 잔고가 저장
- 함수 save()와 withdraw()는 각각 매개변수 금액의 입출금을 구현하는데, 정적 지역변수 amount 를 사용하여 총 입금액과 총출금액을 관리하여 출력.

```
- #include <stdio.h>
- // 전역변수
- int total = 10000;
- //입금 함수 원형
- void save(int);
- //출금 함수 원형
- void withdraw(int);
- int main(void)
- {
-     printf("   입금액   출금액   총입금액   총출금액       잔고\n");
-     printf("=====W\n");
-     printf("%4d\n", total);
-     save(50000);
-     withdraw(30000);
-     save(60000);
-     withdraw(20000);
-     printf("=====W\n");
-
-     return 0;
- }
- //입금액을 매개변수로 사용
- void save(int money)
- {
-     static int amount;
-     total += money;
-     amount += money;
-     printf("%7d %17d %20d\n", money, amount, total);
- }
- //출금액을 매개변수로 사용
- void withdraw(int money)
- {
```

```

- //총 출금액이 저장되는 정적 지역변수;
- static int amount;
- total -= money;
- amount += money;
- printf("%15d %20d %9d\n", money, amount, total);
- }

```

The screenshot shows a Windows console window titled "Microsoft Visual Studio 디버깅 콘솔". It displays a table with five columns: "입금액", "출금액", "총입금액", "총출금액", and "잔고". The table contains three rows of data. Below the table, there is a message in Korean stating that the debugger has finished (code: 0) and providing instructions on how to configure the console to close automatically when debugging stops.

입금액	출금액	총입금액	총출금액	잔고
50000		50000		10000
	30000		30000	80000
60000	20000	110000	50000	30000
				90000
				70000

C:\Users\ran_1\source\repos\Project3\Debug\Project3.exe(프로세스 10092개)이(가) 종료되었습니다(코드: 0개).
 디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용
 하도록 설정합니다.
 이 창을 닫으려면 아무 키나 누르세요...

프로그래밍 연습

다음 조건을 만족하도록 1 에서 100 까지의 하나의 난수를 저장하여 사용자가 이 값을 맞추는 프로그램을 작성하시오.

-함수 setNumber()에서 1 에서 100 까지의 하나의 난수를 발생하여 전역변수 number 에 저장

- -시스템이 정한 number 를 사용자가 맞출 때까지 계속 진행
- -사용자가 정답을 맞추지 못하는 경우는 다음과 같이 힌트를 주도록
- -힌트를 주기 위하여 변수 min,max 를 이용하며, 이 변수는 정적 외부변수로 선언
- -함수는 main()과 함수 setNumber(),printHead(),printHigher(),printLower(),printAnswer()로

구성

- -사용자가 정답을 맞추기 위하여 시도한 횟수를 저장하는 변수 trycount 를 이용하여, 매번 이 값이 출력되도록
- -사용자가 정답을 맞추기 위한 시도 횟수를 최대 5 로 지정하도록

```

int number; //전역변수, 1~100사이의 난수를 저장
int user; //사용자가 입력하는 값을 저장
int trycount; // 시도횟수를 5번으로 지정하도록

```

```

static int min = 1;
static int max = 100; //1~100까지의 값 입력

```

```

void setNumber(void); //1~100사이의 난수를 생성하는 함수
void printHead(void); //게임의 시작
void printHigher(void); //사용자 입력값이 시스템값보다 클때
void printLower(void); //사용자 입력값이 시스템값보다 작을때
void printAnswer(void); //정답을 맞출경우

```

```

int main() {
    setNumber(); //게임 시작 전 시스템의 난수 생성
    printHead(); //처음 게임을 시작하는 함수를 수행
    while (1) { //5번 안에 정답을 맞출때까지
        trycount++;
        if (user > number) { //입력값 > 정답
            printHigher();
        }
        else if (user < number) { //입력값 < 정답
            printLower();
        }
        else { //정답성공
            printAnswer();
            break;
        }
        if (trycount == 4) { //5기회를 모두사용시 while문을 벗어남
            printf("5번의 기회가 끝났습니다. 정답은 %d 입니다.\n", number);
            break;
        }
    }
}

void printAnswer() { //정답을 맞출경우
    printf("축하합니다! 정답은 %d입니다.\n", user);
}

void printHigher() {
    max = user - 1; //난수가 입력값보다 작다면 입력값이 최대값이 되도록
    printf("%d 맞추어야 할 정수가 입력한 정수 %d보다 작습니다.\n", trycount, user);
    printf("%d에서 %d 사이의 정수를 다시 입력하세요. >", min, max);
    scanf_s("%d", &user, sizeof(int));
    puts("");
}

void printHigher() {
    max = user + 1; //난수가 입력값보다 크다면 입력값이 최소값이 되도록
    printf("%d 맞추어야 할 정수가 입력한 정수 %d보다 큼니다.\n", trycount, user);
    printf("%d에서 %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;
}

void printHead() {
    printf("%d에서 %d까지의 하나의 정수가 결정되었습니다.\n이 하나의 정수를 맞추어보세요? > ", min, max);
    scanf_s("%d", &user, sizeof(int));
    puts("");
}

```

13.구조체와 공용체

구조체의 개념

정수나 문자, 실수나 포인터 그리고 이들의 배열 등을 묶어 하나의 자료형으로 이용하는 것이 구조체이다. 연관성이 있는 서로다른 개별적인 자료형의 변수들을 하나의 단위로 묶은 새로운 자료형을 구조체라 한다. 구조체는 연관된 멤버로 구성되는 통합 자료형으로 대표적인 유도 자료형이다. 즉 기존자료형으로 새로이 만들어진 자료형을 유도자료형 이라 한다.

구조체의 정의

구조체를 자료형으로 사용하기 위해서 구조체 정의가 필요하다. 구조체를 사용하려면 먼저 구조체 틀(template)을 정의하여야 한다.

정의하는 방법은 키워드 struct 다음에 구조체 태그이름을 기술하고 중괄호를 이용하여 원하는 멤버를 여러 개의 변수로 선언하는 구조다. 구조체를 구성하는 하나하나의 항목을 구조체 멤버, 또는 필드 라고 한다.

-구조체 정의는 변수의 선언과는 다른것으로, 변수선언에서 이용될 새로운 구조체 자료형을 정의하는 구문이다.

-구조체 내부의 멤버선언구문은 하나의 문장
이므로 반드시 세미콜론으로 종료해야한다.

-멤버가 int credte; int hour 처럼 자료형이 연속
적으로 놓일 경우 간단히 콤마연산자를 사용하여
Int crdit, hour; 로도 가능하다.

```
Struct lecture // struct 구조체 태그이름
{
    Char name [20]; // 자료형 변수명 1;
    Int credit; // 자료형 변수명 2;
    Int hour // 자료형 변수명 3;
}; // 세미콜론은 반드시 필요
```

- 또한 한 구조체 내부에서 선언되는 구조체 멤버의 이름은 모두 유일해야 한다.
- 구조체 멤버로는 일반변수, 포인터 변수, 배열, 다른구조체 변수 및 구조체 포인터도 허용된다.

구조체 변수 선언

구조체가 정의되었다면 이제 구조체형 변수 선언이 가능하다. 즉 구조체 struct 구조체태그이름 이 새로운 자료유형으로 사용될수 있다.

구조체 변수의 초기화

배열과 같이 구조체 변수도 선언시 중괄호를 이용한 초기화지정이 가능하다. 초기화값은 다음과 같이 중괄호 내부에서 구조체의 각 멤버 정의 순서대로 초기값을 쉼표로 구분하여 기술한다. 배열과 같이 초기값에 기술되지 않은 멤버값은 자료형에 따라 기본값인 0,0.0,'W0'등으로 저장된다.

구조체의 멤버 접근 연산자 . 와 변수 크기

선언된 구조체형 변수는 접근연산자 . 을 사용하여 멤버를 참조할 수 있다.

일반적으로 컴파일러는 시스템의 효율성을 위하여 구조체 크기를 산술적인 구조체의 크기보다 크게 할당할 수 있다.

시스템은 정보를 4 바이트 혹은 8 바이트 단위로 전송 처리하므로 이에 맞도록 메모리를 할당하다보면 중간에 사용하지 않는 바이트를 삽입할 수 있다.

그러므로 실제 구조체의 크기는 멤버의 크기의 합보다 크거나 같다.

구조체 활용

구조체 멤버로 이미 정의된 다른 구조체 형 변수와 자기자신을 포함한 구조체 포인터 변수를 사용할 수 있다.

구조체 변수의 대입과 동등비교

동일한 구조체형의 변수는 대입문이 가능하다. 즉 구조체 멤버마다 모두 대입할 필요 없이 변수 대입으로 한번에 모든 멤버의 대입이 가능하다.

공용체 활용

공용체의 개념은 동일한 저장장소에 여러 자료형을 저장하는 방법으로, 공용체를 구성하는 멤버에 한번에 한 종류만 저장하고 참조할 수 있다.

공용체 union 은 서로 다른 자료형의 값을 동일한 저장공간에 저장하는 자료형이다. 공용체 선언방법은 union 을 struct 로 사용하는 것을 제외하면 구조체선언 방법과 동일하다. 물론 구조체와 같이 공용체를 정의하면서 바로 변수를 선언 할 수 있다.

- 공용체 변수의 크기는 멤버 중 가장 큰 자료형의 크기로 정해진다.
- 공용체의 멤버는 모든 멤버가 동일한 저장공간을 사용하므로 동시에 여러 멤버의 값을 동시에 저장하여 이용할 수 없으며, 마지막에 저장된 단 하나의 멤버 자료값만을 저장한다.
- 공용체의 초기화 값은 공용체 정의 시 처음 선언한 멤버의 초기값으로만 저장이 가능하다.

```
Union data // union 공용체 태그이름
{
    Char ch; // 자료형 멤버변수명 1;
    Int cnt ; // 자료형 멤버변수명 2;
    Double real; // 자료형 멤버변수명 3;
} data1; // [변수명] ; 세미콜론은 반드시 필요
```

공용체 멤버 접근

공용체 변수로 멤버를 접근하기 위해서는 구조체와 같이 접근연산자 . 을 사용한다. 유형이 char 인 ch 를 접근하면 8 바이트 중에서 첫 1 바이트만 참조하며, int 인 cnt 를 접근하면 전체 공간의 첫 4 바이트만 참조하고, double 인 real 을 접근하면 8 바이트 공간을 모두 참조한다.

공용체 멤버의 참조는 가능하나 항상 마지막에 저장한 멤버로 접근해야 원하는 값을 얻을 수 있다. 그러므로 공용체를 참조할 경우 정확한 멤버를 사용하는 것은 프로그래머의 책임이다.

자료형의 재정의 typedef

Typedef 는 이미 사용되는 자료 유형을 다른 새로운 자료형의 이름으로 재정의할 수 있도록 하는 키워드이다. 문장 `typedef int profit;` 은 `profit` 을 `int` 와 같은 자료형으로 새롭게 정의하는 문장이다.

일반적으로 자료형을 재정의하는 이유는 프로그램의 시스템 간 호환성과 편의성을 위해 필요하다. 터보 C++ 컴파일러에서 자료유형 `int` 는 저장공간 크기가 2 바이트이나 `visual C++` 에서 작성한 프로그램은 터보 C++에서는 문제가 발생한다. 2 바이트로는 2000000 을 저장할 수 없기 때문이다.

따라서 `typedef int myint;` >> `typedef long myint;` 를 수정하면 터보 C++에서도 이소스를 그대로 이용 가능하다.

문장 `typedef` 도 일반변수와 같이 그 사용범위를 선언된 이후의 그 함수에서만 재정의된다면 재정의된 이후 그 파일에서 이 용 구조체 자료형의 재정의

- 구조체 `struct date` 가 정의된 상태에서 `typedef` 사용하여 구조체 `data` 를 `Date` 로 재정의 할 수 있다.
- `Date` 가 아닌 `datatype` 등 다른 이름으로도 재정의가 가능하다.

```
struct date
{
    Int year ; //년
    Int month ; //월
    Int day; //일
};

typedef struct dat date; //자료유형인
date 는
Struct date 와 함께 동일한 자료유형으로
```


구조체 포인터

포인터 변수 선언

포인터는 각각의 자료형 저장공간의 주소를 저장하듯이 구조체 포인터는 구조체의 주소값을 저장할 수 있는 변수이다. 구조체 포인터 변수의 선언은 일반 포인터 변수 선언과 동일하다.

포인터 변수의 구조체 멤버 접근 연산자 ->

접근연산식 구조체 변수 os 와 구조체 포인터변수 p 인 경우의 의미

P -> name	포인터 p 가 가리키는 구조체의 멤버 name
(*p).name	포인터 p 가 가리키는 구조체의 멤버 name
*p.name	*(p.name)이고 p 가 포인터이므로 p.name 은 문법오류가 발생
*os.name	*(os.name)를 의미하며, 구조체 변수 os 의 멤버 포인터 name 이 가리키는 변수로, 이 경우는 구조체 변수 os 멤버 강좌명의 첫 문자다. 다만 한글인 경우에는 실행 오류가 난다.
*p-> name	*(p->name)을 의미하며, 포인터 p 이 가리키는 구조체의 멤버 name 이 가리키는 변수로 이 경우는 구조체 포인터 p 가 가리키는 구조체의 멤버 강좌명의 첫 문자다. 마찬가지로 한글인 경우에는 실행오류가 난다.

공용체 포인터

공용체 변수도 포인터 변수 사용이 가능하며, 공용체 포인터를 변수로 멤버를 접근하려면 접근연산자 ->를 이용한다.

구조체 배열

구조체 배열 변수 선언

단, 배열과 같이 동일한 구조체 변수가 여러 개 필요하다면 구조체 배열을 선언하여 이용할 수 있다. 다음은 구조체 `lecture` 의 배열크기 3 인 `c` 를 선언하고 초기값을 저장하는 구문이다. 구조체 배열의 초기값 지정 구문에서는 중괄호가 중첩되게 나타난다. 외부 중괄호는 배열 초기화의 중괄호이며, 내부 중괄호는 배열 원소인 구조체 초기화를 위한 중괄호이다.

LAB 13-3

영화 정보를 표현하는 구조체의 배열

영화의 제목과 감독, 관객수를 표현하는 구조체 struct movie 는 멤버로 char * title, int attendance, char director[20]로 구성된다. 이 프로그램에서 구조체 movie 의 배열을 선언하고 초기화로 영화 세 편의 정보를 저장한다. 세 번째 영화의 감독을 "류승완"을 저장하고 모든 영화의 정보를 다시 출력해 보도록 한다.

- 구조체 struct movie

Struct movie

```
{ char * title; //영화제목  
  
    int attendance; //관객수  
  
    char director [20]; //감독  
  
};
```

구조체 movie 의 배열 box 를 선언하면서 초기화

```
#define _CRT_SECURE_NO_WARNINGS  
#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
    //영화 정보 구조체  
    typedef struct movie  
    {  
        char* title;           //영화제목  
        int attendance;        //관객수  
        char director[20];     //감독  
    } movie;  
}
```

```

movie box[] = {
    { "명량", 17613000, "김한민" },
    { "국제시장", 14257000, "윤제균" },
    { "베테랑", 13383000 } };

//영화 베테랑의 감독을 류승완으로 저장
strcpy(box[2].director, "류승완");

printf("   제목       감독   관객수\n");
printf("=====Wn");
for (int i = 0; i < 3; i++)
    printf("[%8s] %6s %d\n",
           box[i].title, box[i].director, box[i].attendance);

return 0;
}

```



```

Microsoft Visual Studio 디버그 콘솔
제목   감독   관객수
=====
[명량] 김한민 17613000
[국제시장] 윤제균 14257000
[베테랑] 류승완 13383000
C:\Users\ran_1\OneDrive\바탕 화면\과제\호연\기말과제\ch01\ch11\wchararray.c\Debug\test.exe(프로세스 27764개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

```

프로그래밍 연습

01-02 다음 내용을 참고로 구조체 fraction 을 정의하고, 2 개의 분수를 선언하여 적당한 값을 입력하고 출력하는 프로그램을 작성하시오.

-구조체 fraction 멤버 구성 : 정수형의 분자(numerator)와 분모(denomination)

-구조체 fraction 에서 두 분수의 곱을 출력하는 프로그램을 작성하시오.

구조체 fraction 의 예: $4/5 * 3/7$ 의 결과는 $12/35$

```
#include <stdio.h>

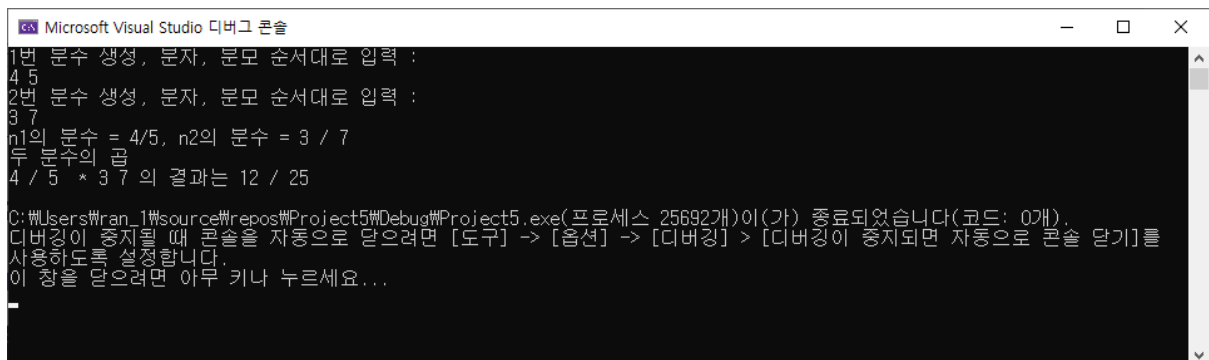
typedef struct { //구조체 정의
    int numerator;
    int denominator;
}fraction;

int main() {

    fraction n1; //구조체 n1 생성
    fraction n2; //구조체 n2 생성

    printf("1번 분수 생성, 분자, 분모 순서대로 입력 : \n");
    scanf_s("%d %d", &n1.numerator, &n1.denominator, sizeof(int)); //구조체 n1 분수 생성
    printf("2번 분수 생성, 분자, 분모 순서대로 입력 : \n");
    scanf_s("%d %d", &n2.numerator, &n2.denominator, sizeof(int)); //구조체 n2 분수 생성

    printf("n1의 분수 = %d/%d, n2의 분수 = %d / %d \n", n1.numerator, n1.denominator,
n2.numerator, n2.denominator);
    printf("두 분수의 곱 \n%d / %d * %d %d 의 결과는 %d / %d\n", n1.numerator, n1.denominator,
n2.numerator, n2.denominator,
n1.numerator*n2.numerator, n1.denominator*n2.denominator);
}
```



```
Microsoft Visual Studio 디버그 콘솔
1번 분수 생성, 분자, 분모 순서대로 입력 :
4 5
2번 분수 생성, 분자, 분모 순서대로 입력 :
3 7
n1의 분수 = 4/5, n2의 분수 = 3 / 7
두 분수의 곱
4 / 5 * 3 7 의 결과는 12 / 25

C:\Users\Wan_1\source\repos\Project5\Debug\Project5.exe(프로세스 25692개)이(가) 종료되었습니다(코드: 0x0).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를
사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

맺음말

c언어를 공부하면서 가장 크게 느낀 것은 한가지 문제에 대한 해답이 무궁무진하다는 것이다. 그것이 바로 프로그래밍의 재미이다.

내가 어렵고 복잡하게 짠 코딩을보고 뿌듯해하고 있지만 책에나와있는 해답이나 친구의 간략한 코딩을보면 감탄을 자아내기도 한다.

아직은 간추리고 조금더 쉽게 코딩을 짜기까지 생각의 범위를 키우고 노력하는데에는 지금보다 더 필요한 시간이 많을것이라 생각한다.

또한 첫시간의 교수님의 말씀처럼 연습과 복습이 정말로 중요하다는 것을 깨닫는다. 지금당장 시간을 줄이고자 띄엄띄엄 강의를 듣고 빨리넘기고, 하다가는 다시 강의를 이해하고 듣느라 시간을 배로 쓰게된다.

마음이 급해서 뛰어넘지말고 처음 개념부터 교수님이 알려주신대로 차근차근 듣는게 중요하다는걸 다시금 깨닫는다.

입학 초 교수님이 알려주신 연습법대로 공부하는 날에는 확실히 궁금증을 가진 채로 수업에 임하게 되니 더 이해가 되고, 발전이 되고 도움이 된다. 하지만 연습을 하지않을경우 복습으로 채울수는 있지만 연습만큼 따라가지는 못한다는걸 깨닫는다.

1년동안 열심히 배운 C언어 이다. 아직도 이 프로그래밍 언어를 정복하기에는 무궁무진한 시간들이 소모될것이라 생각한다. 그럼에도 불구하고 계속 놓지 못하는 이유는 알면 알수록 더 궁금해지는 C언어의 세계를 알고싶어서가 아닐까.

자바와는 달리 조금더 거리감이 있던 언어지만 순차적으로 절차를 읽어가며 코딩을 해본다면 처리속도도 빠른 C언어가 매력적으로 느껴질수밖에 없다. 하지만 오류를 발견하게된다면 다시 또 머리를 조아리게 되겠지. 그래도 오류 해결의 짜릿함을 느껴본 사람이라면 쉽게 코딩을 놓진 못할것이라 생각한다.

프로그래밍 기초와 알고리즘밖에 몰랐던 나에게 C언어를 배우는 시간은 정말 소중한했다. 다만 잘못된게 있다면 나의 게으름밖에 없을것이다. 감사했던 시간들과 배움의 즐거움을 알게되는 소중한 시간들이었다고 생각하며 글을 마친다.