

# C언어 보고서

김한솔

# 목차

## C언어의 기본 구조 및 표준 입출력

- 상수
- 기호상수
- 변수
- 연산자
- 표준입출력

## 제어문

- 조건문
  - if
  - else if
  - switch

### 반복문

- for
- while
- do while

## 배열

- 1차원배열
- 2차원배열

## 함수

## 포인터

- 포인터란?
- 포인터 관련 연산자의 의미
- 포인터와 배열의 관계
- 배열의 이름이 갖는 의미
- []를 이용한 배열 접근과 포인터 간의 관계
- 배열을 이용해 선언한 문자열과 포인터를 이용해 선언한 문자열 간의 차이
- scanf 함수에서 &연산자를 쓰는 이유? 배열은 쓰지 않는 이유?
- 포인터와 함수의 관계
- Call-By-Value 방식의 동작 과정
- Call-By-Reference 방식의 동작 과정
- 둘 간의 차이
- 함수의 인자로 배열을 전달했을 때의 실행 구조

# C언어 특징

1972년 발표 (데니스 리치 & 켄 톰슨 in AT&T Bell 연구소)

(설에 따르면 켄 톰슨이 게임을 즐기기 위해 만들었다고함)

첫 컴파일러는 어셈블리어로 제작되었다.

고급언어중 하나이다.

고급언어임에도 불구하고 하드웨어단계까지 접근 및 제어가 가능하다.(C언어를 제대로 사용하려면 하드웨어에 관한 지식도 많아야 한다고 한다.)

많은 언어들에게 영향을 주었다.

운영체제, 임베디드, 암호학, 웹서버 등에 사용된다.

윈도우에서는 주로 VS, dev c++,cobol등 다양하게 쓰인다.

리눅스에서는 vi, vim, nano등을 쓴다.

문법이나 라이브러리함수가 적고 간단하다, 그러나 직접 구현해야 하는 게 많다는 것이고 구현과 더불어 최적화를 얼마나 하느냐가 관건인 언어이다.

# C언어의 기본 구조 및 표준 입출력

```
1 #include <stdio.h> // 헤더파일 (적처리기 처리)
2 int main(void){ // 프로그램이 동작하는 메인함수
3     printf("Unifox : Whatever you want, Moreover we have"); // 기본 출력함수
4     return 0; // 메인함수 종료
5 }
```

#include <stdio.h> stdio.h라는 헤더파일 안에 있는 다양한 정보를 이 프로그램에서 사용할 수 있도록 한다.

int main(void){} c프로그램에 필요한 함수로 모든 프로그램은 main 함수에서 시작한다.(이 함수가 없으면 컴파일오류 발생)

printf(“Unifox ~~”); stdio.h파일에 정의되어있는 “와 ”사이에 있는 내용을 출력한다.

return 0; main함수를 종료하는 명령어이다.

```
1 #include <stdio.h>
2 int main(void){
3     printf("Unifox : ");|
4     printf("Whatever you want, Moreover we have");
5     return 0;
6 }
```

위와 같이 여러 줄의 printf()문을 사용하여도 줄 바꿈은 되지 않는다. printf()함수는 출력만 하지 커서를 다음 줄로 넘기지 않기 때문이다. 이때 커서를 다음 줄로 넘기려면 개행이라는 이스케이프문자인 \n을 사용하여야 한다.

```
1 #include <stdio.h>
2 int main(void){
3     printf("Unifox : \n");|
4     printf("Whatever you want, Moreover we have");
5     return 0;
6 }
```

```
1 #include <stdio.h>
2 int main(void){
3     printf("Unifox :\nWhatever you want, Moreover we have");
4     return 0;
5 }
```

위와 같이 작성하면 아래와 같이 줄바꿈되어 출력된다.

Unifox :

Whatever you want, Moreover we have

이스케이프문자는 \n뿐만 아니라 다양하다.

|    |                      |
|----|----------------------|
| \n | 커서를 줄 바꿈한 뒤 맨 앞으로 이동 |
| \t | 커서를 탭 간격만큼 이동        |
| \0 | NULL을 나타낸다.          |
| \a | 비프음을 출력              |
| \b | 커서를 한 칸 앞으로 이동       |
| \r | 커서를 현재 줄의 맨 앞으로 이동   |
| \v | 수직 탭                 |

아래는 이스케이프문자를 이용한 프로그램의 예시이다.

```
1 #include <stdio.h>
2 int main(void){
3     printf("test1\n");
4     printf("test2\t \n");
5     printf("test3\a \n");
6     printf("test5\b4 \n");
7     printf("TEST0\rtest5 \n");
8     printf("test6 \e test7 \f \n");
9     printf("test8 \0 Unifox");
10    return 0;
11 }
```

"C:\Users\Wgoen\Desktop\Unifox\프로그래밍 보고서\wbint

```
test1
test2
test3
test4
test5
test6 ← test7 ♀
test8
Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

\e 와 \f는 특수문자를 출력한다. 그러나 교과서에는 \f 가 ‘폼 피드라고 적혀 있다. 커서를 새로운 페이지로 이동시키는 것이라 하는데 제대로 작동하지 않는 듯하다. 그리고 printf()는 “”사이에 있는 문자열은 그대로 출력한다고 되어있지만 NULL문자가 나올 경우에는 따로 예외처리를 한다.

## 상수

위에서 기본 문자열을 화면에 출력했는데 이러한 문자열도 상수에 속한다. 상수란 변하지 않는 값으로 프로그래밍에서는 프로그램이 실행되는 동안 변하지 않는 값으로, 문자열 상수 외에도 수치상수, 문자상수, 리터럴상수, 심볼릭상수등이 있다.

### 수치상수

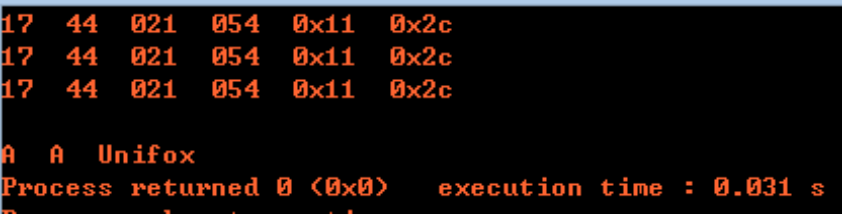
|              |        |                  |
|--------------|--------|------------------|
| 고정소수점 상수(정수) | 8진 정수  | 010, 013, 0621   |
|              | 10진 정수 | 100, 1234, 1120  |
|              | 16진 정수 | 0x11, 0x9B, 0x2E |
| 부동소수점 상수(실수) | 지수 표현  | 7e2              |
|              | 소수점 표현 | 1.236, 22.22     |

수치상수를 제대로 사용하기 위해서는 변환문자열(서식지정자)를 사용하여야 한다.

### 변환문자열

|    |  |
|----|--|
| %d | 인수를 10진수 형태로 출력                                      |
| %o | 인수를 8진수 형태로 출력                                       |
| %x | 인수를 16진수 형태로 출력                                      |
| %f | 인수를 실수형식으로 출력 (소수점)                                  |
| %e | 인수를 실수형식으로 출력 (지수형식)                                 |
| %g | 인수를 %f %e중 짧고 보기 좋은 형태로 출력한다. (%f이면 맨 우측의 필요 없는 0제거) |
| %c | 인수를 문자 형식으로 출력한다.                                    |
| %s | 인수를 문자열 형태로 출력한다.                                    |

```
1  #include <stdio.h>
2  int main(void){
3      printf("%d %d %o %o 0x%x 0x%x",17,44,17,44,17,44);
4      printf("\n");
5      printf("%d %d %o %o 0x%x 0x%x",021,054,021,054,021,054);
6      printf("\n");
7      printf("%d %d %o %o 0x%x 0x%x",0x11,0x2c,0x11,0x2c,0x11,0x2c);
8      printf("\n\n");
9      printf("%c %c %s",65,'A', "Unifox" );
10
11      return 0;
12 }
```



위 이미지는 변환문자를 사용한 프로그램의 예제이다. 위에서부터 차례대로 10진수, 8진수, 16진수를 변환문자를 이용하여 출력한 것이다.

## 기호상수

전처리기 지시자인 #define을 이용한 것으로 주로 매크로라고 알려져 있다. #define의 특징이라면 그대로 가져간다는 것이다. 그대로 가져옴에 따라

프로그램이 예상하지 못한 결과를 가져올 수 도 있지만 잘 알고 쓰면 매우 유용하다.

```
#include <stdio.h>
#define MAX 20
#define MIN (MAX-5)
int main(void){
    int test=MIN*2;
    printf("%d",test);
    return 0;
}
```

그림1

그림1과 그림2의 차이점은 MIN이라는 상수를 선언할 때 괄호의 유무이다. 그림1을 설명하자면 MAX는 20이고 MIN은 (MAX-5)라는 값을 가지고 main함수에 있는 test변수에는 (MAX-5)\*5값, 30이 들어가지만 그림2에서는 MIN의 값은 MAX-5이고 test변수에는 MAX-5\*2로 들어가 버린다. 위와 같이 #define을 사용하게 되면 괄호에 따라 다음과 같은 문에서 해석을 제어하기 때문에 중요하다. 기호상수는 정수뿐만 아니라 실수, 문자열, 함수 등 다양한 것을 정의할 수 있다. #define 말고도 const키를 사용하여 상수선언을 할 수 있다. const키를 이용한 상수선언은 주로 함수 내에서 사용한다.

상수선언을 할 때에는 지켜야하는 것이 있다. 바로 선언과 동시에 초기화가 이루어지며 그 후에는 다시 초기화를 할 수 없다는 것이다.

```
#include <stdio.h>
#define MAX 20
#define MIN MAX-5
int main(void){
    int test=MIN*2;
    printf("%d",test);
    return 0;
}
```

그림2

```
#include <stdio.h>
#define MAX
#define MIN
int main(void){
    const int Sym=11;
    const int Bol=10;
    return 0;
}
```

왼쪽이미지는 잘못된 코드이다. 그 이유는 Sym상수는 처음에 선언과 초기화가 올바르게 되었지만 그 후에 다시 초기화를 시도하여 오류가 난다. 그리고 Bol상수는 선언과 동시에 초기화가 이루어지지 않았다.

```
#include <stdio.h>
#define MAX
#define MIN
int main(void){
    const int Sym=10;
    const int Bol;
    Sym=11;
    Bol=10;

    return 0;
}
```

왼쪽이 오류가 나지 않는 코드이다. 한 가지 해결하지 못한 의문점이 있다. 매크로로 선언한 MAX와 MIN상수는 선언과 함께 초기화가 이루어지지 않았지만 오류나 경고도 뜨지 않았다. 그러나 초기화를 main함수 안에서 시도하면 바로 오류를 뱉어낸다. 이것에 대해서는 좀 더 실험해보고 찾아봐야 한다.

## 변수

프로그래밍의 변수는 데이터의 저장공간이다. 상수와 달리 값이 정해진 것이 아니라 계속 변할 수 있는 것이다. #define을 이용한 매크로상수같은 경우에는 선언된 그대로 가져와서 매크로상수=값이란 느낌이지만 변수는 이름을 가진 컨테이너라는 개념이다.

프로그램에서 변수를 사용하기 위해서는 먼저 변수를 사용하겠다고 선언을 해주어야 사용 가능하다. 변수를 선언할 때에는 따라야하는 규칙이 있다.

- \* 영문자, 수치문자, 밑줄을 사용할 수 있으며 영문자는 대소문자를 구분한다.
- \* 처음 시작할 때는 수치문자로 시작할 수 없다.
- \* 예약어(키워드)를 변수 이름으로 사용할 수 없다.

‘\$’는 특수문자 중에서도 예외적으로 사용이 가능하다. (Codeblocks-Mingw GNU gcc).

예약어 : int, auto, unsigned, switch, sizeof, return, const,break, default, do, if, union, for, long, void, enum etc..

예약어는 IDE나 텍스트에디터에서 색으로 강조되는 것이라 생각하면 쉽게 이해 가능하다

변수를 선언하기 위해서는 선언하려는 값에 상응하는 자료형을 써줘야 한다. 자료형을 써줘야 컴퓨터는 그 값을 처리하는 방식을 정한다. 아래는 자료형에 관한 표이다.

| 데이터형        | 바이트                           | 표현 범위  |
|-------------|-------------------------------|--|
| int         | 4byte                         | -2,147,483,648 ~ 2,147,438,647                   |
| short       | 2byte                         | -32,768~32,767                                   |
| long        | 4byte                         | -2,147,483,648 ~ 2,147,438,647 (OS에 따라 다르다 고한다)  |
| long int    | 4byte                         | long 과 동일하다                                      |
| float       | 4byte                         | 3.4E-38(-3.4*10^38) ~ 3.4E+38(3.4*10^38)         |
| double      | 8byte                         | 1.79E-308(-1.79*10^308) ~ 1.79E+308(1.79*10^308) |
| long double | 12byte (ms문서에는 double과 같다고나옴) | double과 동일                                       |
| char        | 1byte                         | -128 ~ 127                                       |

위에 있는 자료형을 올바르게 입출력하기 위해서는 위에 설명 했던 제어문자를 사용한다.

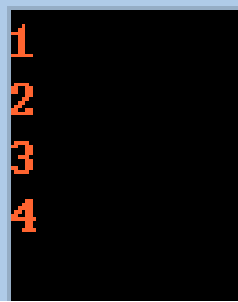
변수에는 지역변수, 전역변수, 정적변수가 있다.

지역변수는 그 변수가 선언된 함수 내에서만 사용 가능한 변수이다. main()함수에 선언된 변수도 지역변수이다.

전역변수는 주로 전처리기 처리영역과 함수선언 영역 사이에 선언하는데 전역변수는 그 코드내부에서 어디든 접근과 이용이 가능하다.

정적변수는 선언할 때는 지역변수와 마찬가지로 함수 내부에서 하지만 지역변수처럼 함수가 끝나도 사라지지 않는다.

```
#include <stdio.h>
int test(){
    static int cnt=0;
    cnt++;
    printf("%d\n",cnt);
}
int main(){
    test();
    test();
    test();
    test();
    return 0;
}
```



<정적변수의 예시>

## 연산자

C언어에서는 다양한 수식을 계산, 결과를 도출해 내기 위해 다양한 연산자를 지원, 사용가능하다. 연산자의 종류로는 산술, 대입, 관계, 논리, 비교연산자등이 있다.

산술연산자 : 수치 계산에 이용되는 연산자로 다음과 같이 덧셈, 뺄셈, 곱셈, 나눗셈을 수행한다. 그리고 정수에 대한 나눗셈 연산(/)에서는 소수점 이하는 버린다. 나머지 연산인 a%b a를 b로 나누었을 때의 나머지를 나타내며, 정수 연산만 가능하다. 그리고 증감연산자(++/--)를 사용할 때 조심해야 할 점이 있는데, 바로 연산자를 어디에 두느냐에 따라 뜻이 바뀐다. 연산자를 변수의 앞에다 두면 변수를 사용하기 전에 연산을 하고 뒤에다 두면 변수를 사용한 후에 연산을 한다.



|    |                                |
|----|--------------------------------|
| +  | 두 피연산자를 더함                     |
| -  | 앞의 피연산자에서 뒤의 피연산자를 뺌           |
| *  | 두 피연산자를 곱함                     |
| /  | 앞의 피연산자에서 뒤의 피연산자를 나눔          |
| %  | 앞의 피연산자에서 뒤의 피연산자를 나눈 나머지      |
| ++ | 피연산자에 1증가 (연산자의 위치에 따라 순서가 바뀜) |
| -- | 피연산자에 1감소 (연산자의 위치에 따라 순서가 바뀜) |

관계연산자 : 좌변과 우변의 값을 비교하여 크기관계를 평가하며, 그 결과를 참과 거짓으로 나타낸다. 대입연산자와 마찬가지로 두 개의 기호를 사용해야 할 경우(!=, >= etc) ‘=’는 두 개의 기호중 항상 뒤로 간다.

|    |                                 |
|----|---------------------------------|
| == | 같다 ('='은 대입연산자와 겹치므로 '=='을 사용함) |
| != | 같지 않다                           |
| <  | 작다                              |
| <= | 작거나 같다                          |
| >  | 크다                              |
| >= | 크거나 같다                          |

```
#include <stdio.h>
int main(void) {
    int result, a=1, s=1, d=1, f=0;
    result= a==s;
    printf("result : %d\n", result);
    result= a<s;
    printf("result : %d\n", result);
    result= s!=d;
    printf("result : %d\n", result);
    result= a!=f;
    printf("result : %d\n", result);
    return 0;
}
```

```
result : 1
result : 0
result : 0
result : 1
```

관계연산자의 결과는 0과 1로 나타낸다. 뒤에 후술할 if도 0과 1을 이용하여 조건의 참\*거짓을 판별한다.

대입연산자 : 변수의 값을 변경하기 위해서 사용하는 연산자이다. 오른쪽에 잇는 값을 왼쪽에 대입하는 데 사용된다. 대입연산자에는 단순히 대입만 하는 것이 아닌 연산 후 대입을 하는 복합 대입연산자가 있다. 바로 +=나 \*= 같은 것으로 대입연산자에 산술연산자가 복합된 형태이다. a=a+1; 은 a라는 변수는 a변수가 가지고 있는 값에 1을 더한 값이다. 라는 뜻이다. 그러나 이 연산을 복합 대입연산자를 사용해서 간단히 만들면 a+=1; 로 줄일 수 있다.

|    |                              |
|----|------------------------------|
| =  | 우변의 값을 좌변에 대입                |
| += | 좌변에 우변의 값을 더한 결과를 좌변에 대입     |
| -= | 좌변에서 우변 값을 뺀 결과를 좌변에 대입      |
| *= | 좌변에 우변 값을 곱한 결과를 좌변에 대입      |
| /= | 좌변에 우변 값을 나눈 결과를 좌변에 대입      |
| %= | 좌변에 우변 값을 나눈 나머지를 좌변에 대입     |
| &= | 좌변과 우변을 AND 비트연산한 결과를 좌변에 대입 |
| ^= | 좌변과 우변을 XOR 비트연산한 결과를 좌변에 대입 |

논리연산자 : 한 가지 이상의 조건을 동시에 요구하는 경우가 있다. 여러 가지의 조건을 판단하기 위해서 주로 쓰인다.

|    |  |
|----|--|
| && | AND(논리곱) 양쪽 항이 모두 참일 때만 결과가 참이 된다.     |
|    | OR(논리합) 양쪽 항 중에 하나만 참이어도 결과가 참이 된다.    |
| !  | NOT(논리부정) 피연사자의 값이 참이면 거짓, 거짓이면 참이 된다. |

캐스트연산자 : C언어에서는 임의로 자료형을 변형하여 연산을 할 수 있다. 이 때 사용하는 연산자가 바로 캐스트연산자이다.



```
#include <stdio.h>
int main(void){
    int a=10;
    double s=20.0;
    printf("%d",a+(int)s);
    return 0;
}
```

위와 같이 (자료형이름)변수명을 적어준다. 위와 같이 적으면 double형 변수가 잠시 int형으로 변형되어 계산된다. 캐스트연산자는 자료형이 잠시 변형되어 연산을 하는 것이지 변수 s에 저장된 값이 바뀌는 것은 아니다.

## 표준 입출력

printf()는 출력, scanf()는 입력할 때 쓰인다.

printf()는 “”사이에 있는 문자열은 그대로 출력하고 NULL문자를 만나면 ”를 만나지 않아도 출력을 멈춘다.

변수나 상수를 출력할 때에는 제어문자를 사용하여 출력한다.

```
#include <stdio.h>
int main(void){
    float a;
    char s;
    int d,f;
    a=10;
    s='A';
    f=d=0x10;
    printf("%g %c %d %d",a,s,d,f);
    return 0;
}
```

(표준출력)

다양한 자료형의 변수를 선언하고 알맞게 값을 넣는다. 그 후 알맞은 제어문자를 출력하게 하고 “” 밖에다가 함수내부에 적었던 제어문자의 순서대로 변수나 상수명을 적어주면 정확하게 출력이 된다.

```
#include <stdio.h>
int main(void){
    float a;
    char s;
    int d,f;
    scanf("%f %c %d %d",&a,&s,&d,&f);
    printf("%g %c %d %d",a,s,d,f);
    return 0;
}
```

(표준입력)

표준입력함수인 scanf()는 printf()와 마찬가지로 입력받고 싶은 변수와 순서에 맞게 제어문자를 적은 후 뒤에 변수명을 적는데 이 때 한 가지 유의해야 하는 것이 있다. 바로 변수 앞에 ‘&’을 붙이는 것이다. 붙이는 이유는 scanf에 a라는 변수를 넘겨주었다면, scanf는 a라는 변수 자체를 사용하는 것이 아니라 scanf내부에 다른 변수를 선언해서 그 변수에 a값을 대입하고, 대입한 값을 이용하는 것이기 때문이다.

# 제어문

## 조건문

제어문이란 프로그램의 실행 순서를 변경하는 것으로 if문, switch문, else문 등이 있다.

if문 : 조건에 따라 프로그램의 흐름이 변경되기 때문에 분기문 이라고도 하며 조건식이 참이면 내용을 실행하고 거짓이면 실행하지 않는다. 조건을 판단하는 것은 연산자를 통해 판단한다.

```
a=s
Process
Press a
#include <stdio.h>
int main(void){
    int a=1,s=1;
    if(a==s){
        printf("a=s");
    }
    else{
        printf("a!=s");
    }
    return 0;
}
```

<그림1>

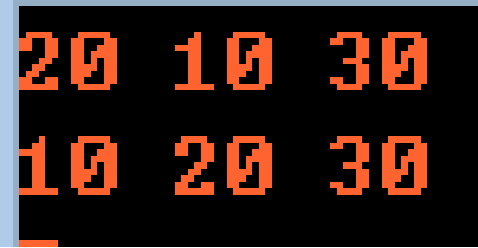
<그림1>에서는 변수 a와 s의 값이 같으므로 첫 번째 조건인 a==s, a의 값과 s의 값이 서로 동일한지를 묻는 조건이 참이 되었으므로 “a=s”를 출력한다. <그림2>는 첫 조건문이 거짓이 되므로 else문으로 넘어가게 된다.

if문 바로 아래에 if문을 중첩하여 쓸 수 있다.

```
#include <stdio.h>
int main(void){
    int a,s,d;
    scanf("%d %d %d", &a, &s, &d);
    if (a >= s){
        int temp = a;
        a = s;
        s = temp;
    }
    if (a >= d){
        int temp = a;
        a = d;
        d = temp;
    }
    if (s >= d) {
        int temp = s;
        s = d;
        d = temp;
    }
    printf("%d %d %d", a, s, d);
    return 0;
}
```

```
a!=s
Process
Press a
#include <stdio.h>
int main(void){
    int a=1,s=0;
    if(a==s){
        printf("a=s");
    }
    else{
        printf("a!=s");
    }
    return 0;
}
```

<그림2>



이 예시는 반복 if문을 사용하여 정수형 변수들을 오름차순 정렬하여 출력하는 소스이다. 1번째 if로 a <= s가 확정되고 2번째 if로 a <= d가 확정된 다음 마지막 if로 s <= d가 되니 a <= s <= d, 모든 경우에 대해 정렬이 된다.

else if문은 if문을 중첩시킨 것과 구조는 비슷하지만 if와 else if문들 중 처음으로 해당되는 조건문만 실행하고 바로 종료해버린다.

```
#include <stdio.h>
int main(void){
    int w;
    scanf("%d",&w);
    if (w < 18){
        printf("저체중");
    }
    else if (w < 23){
        printf("정상");
    }
    else if (w < 25){
        printf("과체중");
    }
    else if (w < 30){
        printf("경도비만");
    }
    else{
        printf("중등도비만");
    }
    return 0;
}
```

w 라는 변수에 20이 입력되면 첫 번째 조건문은 통과하고 두 번째에서 문장이 실행되고 main함수가 종료된다.

이와 같이 if 와 else if 문이 중첩되어 쓰이면 처음으로 조건에 맞는 것만 실행하고 바로 종료시킨다.

지금까지의 if, else if 문에는 조건식을 하나만 썼지만 하나만 쓰는 것 외에도 좀 더 쓸 수 있다.

```
#include <stdio.h>
int main(void) {
    int a,s,d,f;
    a=s=d=f=1;
    printf("%d %d %d %d\n\n",a,s,d,f);
    if((a==f && s==f)&&((a==s && d==f)|| (a!=f || s==d))) {
        printf("성공");
    }
    return 0;
}
```

```
1 1 1 1
성공
Process returned 0
Press any key to continue
```

위의 예시는 지금까지 써왔던 조건문보다 훨씬 복잡한 조건식을 가지고 있다. 위의 조건문을 살펴보면

(a==f&&s==f) && ( (a==s && d==f) || (a!=f || s==d) )  
( 1 && 1 ) && ( ( 1 && 1 ) || ( 0 || 1 ) )  
( 1 ) && ( ( 1 ) || ( 1 ) )  
( 1 ) && ( 1 )  
( 1 )

로 정리할 수 있다. 논리연산자를 사용하여 다양한 수식을 조건문 안에 넣을 수 있다.

switch문

if문은 조건에 따라 참과 거짓의 두 가지 경우를 처리하는데 적합한 반면, switch문은 다양한 경우의 처리에 적합하다. switch문은 수식의 값에 따라 여러 방향으로 분기하기 위한 것이다.

```
#include <stdio.h>
int main(void){
    int w;
    scanf("%d",&w);
    switch(w){
        case 1 : printf("변수 w는 1입니다.\n");
        case 2 : printf("변수 w는 2입니다.\n");
        case 3 : printf("변수 w는 3입니다.\n");
        default : printf("잘 모르겠어");
    }
    return 0;
}
```

```
1
변수 w는 1입니다.
변수 w는 2입니다.
변수 w는 3입니다.
잘 모르겠어
Process returned 0
```

switch문의 형식은 위에 있는 소스코드처럼 되어있다. 그러나 위 코드처럼 작성하고 실행한 다음 1을 입력하면 case 1, case 2, case 3 그리고 default까지 출력하게 된다. 이것 막기 위해서는 각 case마다 break;를 써준다. break의 의미는 그 제어문을 벗어난다는 뜻을 담고 있다. switch문에서는 처음 수식에 넣을 변수는 꼭 정수이어야만 하는 것이 아니라 실수, 문자형도 가능하다. 문자형이 가능한 이유는 C언어에서의 문자는 처리될 때 숫자로 처리되기 때문에 가능한 것이다.

```
#include <stdio.h>
int main(void){
    int a=100;
    char s='A';
    printf("%c %c %d %d",a,99,s,'C');
    return 0;
}
```

```
d c 65 67
Process returned 0
Press any key to continue
```

위와 같은 예시를 통해 C언어 내부에서는 문자형을 처리할 때에는 아스키코드에 대응되는 숫자로 처리하는 것을 알 수 있다.

## 반복문

반복적인 일을 제어, 실행하기 위해 사용한다. 수많은 작업을 사람이 일일이 코드로 옮겨서 작성하면 오래 걸리고 매우 비효율적일 것이다. 그러나 반복문을 사용하면 적은 코드로 반복적인 작업을 매우 많이 실행할 수 있게 한다. C언어에서의 반복문은 for, while, do while이 있다.

### for문


주어진 조건식이 참인 동안에는 특정 부분의 작업을 반복적으로 수행하는 제어문이다.

```
for (초기식; 조건식; 변환식) {
    실행시킬 작업;
}
```

for문을 처음 만나면 초기식을 먼저 계산한다. 그리고 조건식이 참인지를 확인하여 참이면 작업을 실행하고 변환식 계산, 그 후에 다시 조건식으로 돌아와 조건을 평가한다. 조건식을 평가할 때 거짓이면 작업을 실행하지 않고 for문을 종료한다. 또한 수식을 생략해도 되는데, 생략을 하더라도 ‘;’(세미콜론)은 생략 불가능하고 조건수식을 생략한 경우에는 무한루프가 돌아버린다. 변환식에서도 또 다른 변형이 가능하다. 주로 변환식에는 증감연산자를 이용하거나 대입연산자를 이용하는데 이러한 연산자들 말고도 입출력문을 사용 가능하다. 아래는 변환식에 표준출력문을 사용하여 기본적인 별찍기에제를 푼 코드이다.

```
#include <stdio.h>
int main(void) {
    for(int a=0;a<5;a++){
        for(int s=0;s<=a;printf("*"),s++);
        printf("\n");
    }

    return 0;
}
```



2번째 반복문의 변환식 부분에서 printf(), 표준출력문을 사용한 예시이다. 저렇게 코드를 사용하는 것 보다는 출력문을 아래쪽으로 내린 코드가 좀 더 가독성이 좋다. 그저 저것은 변환식에는 다양한 것들이 들어갈 수 있다는 예제이다.

### while문

while문은 주어진 조건이 참인 동안에 반복문 내의 작업을 실행하고 그렇지 않으면 반복문을 종료한다. for문과는 달리 구조가 단순하다.

```
while (조건식) {
    실행시킬 작업;
}
```

for문에는 있는 초기식과 변환식이 없는 대신에 while문 내부에 같은 동작을 하는 코드를 만들어서 사용한다. for문과는 다르게 조건식을 생략하면 컴파일이 되지 않는다. 무한루프를 돌리고 싶다면 조건식에 1만 작성하면 무한루프가 돈다.

### do-while문

do-while문은 문장을 우선 실행하고 반복문 마지막에서 조건식을 검사하는 반복문이다.

```
do{
    printf("*");
}while(1);
```

do-while문은 반드시 while(조건식) 뒤에는 ‘;’(세미콜론)이 붙는다는 점과 while문은 조건식을 검사하고 실행하는 반면에 do-while문은 실행하고 조건식을 검사하는 것 말고는 차이점이 없다.

break & continue문

break문은 for, while, do-while, switch문을 강제로 벗어나게 하는 데 사용된다. 주로 조건문과 같이 쓰며 중첩된 반복문 내에 있을 경우에는 가장 가까운 반복문만 벗어난다.

continue문은 현재의 반복을 멈추고 다음반복을 하게한다. continue를 만나게 되면 코드의 밑에 있는, 그 때의 반복해야 할 작업들을 무시하고 반복문의 처음으로 돌아간다.(for문은 변화식을 실행, 조건식 검사) break문과는 달리 switch문에서는 사용할 수 없다.



## 배열

특정한 자료형 변수의 집합이다. 같은 이름을 사용하지만 index로 구분되는 데이터구조이다.

```
자료형 배열명[배열의 크기];
int arr[5];
```

선언은 위와 같이한다. 배열을 사용하는 이유는 같은 목적을 가진 일련의 변수들을 묶어서 사용하는 것이다. 예를 들면 학생을 3명을 관리한다면 student1, student2, student3 이라는 변수를 직접 생성하지만 학생을 30명을 관리하게 된다면 student1, student2, student3~~처럼 일일이 선언하고 관리하는 데에는 매우 힘이 드니 배열이라는 것을 이용하여 student[30]을 선언하면 학생을 관리할 수 있는 각각의 변수가 30개가 바로 생성이 된다.

배열을 사용할 때에는 주의할 점이 하나 있다. 배열의 Index를 참조할 때는 배열이 처음 선언된 숫자 n-1까지 참조할 수 있다. 그 이유는 배열의 Index는 0부터 매겨지기 때문이다. 만약 arr1[5]이라는 배열은 Index가 0부터 4, arr2[13]은 0부터 12까지 매겨진다.

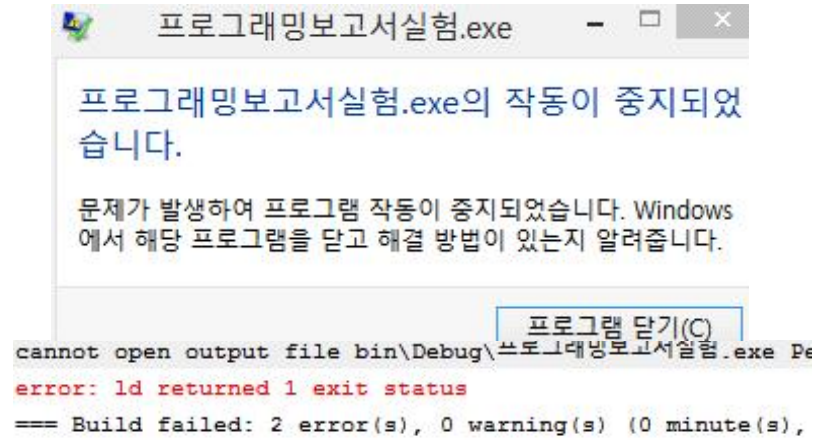
그러나 보고서를 쓰면서 실험해본 결과 이상한 점을 하나 발견했다.

```
#include <stdio.h>
int main(){
    int arr[5];
    int i;
    for(i=0;i<9;i++){
        arr[i]=i;
    }
    for(i=0;i<9;i++){
        printf("%d\n",arr[i]);
    }
    return 0;
}
```

0  
1  
2  
3  
4  
5  
6  
7  
8

위는 int형 배열의 크기를 5만큼 선언해 주고 값의 대입과 참조는 9까지 이루어지고 있다. 그러나 컴파일과 실행은 오류 없이 이루어졌다. 배열의 크기를 5이상 선언해 주면 저런 현상이 일어났다.

배열의 크기를 4이하로 선언해 주면 아래와 같은 오류가 뜨면서 프로그램이 종료되었다.



2차원배열은 선언할 때 대괄호를 2쌍을 이용하여 선언한다. C언어는 첫 번째 쌍은 행, 두 번째 쌍은 열을 나타낸다.

```
#include <stdio.h>
int main(){
    int arr[3][2]={{1}, {2,3}, {4,5}};
    return 0;
}
```

위와 같이 첫 번째 행의 2열을 비우면서 선언도 가능하다.

배열은 주로 7차원배열까지 선언할 수 있다고 한다.



함수

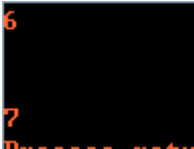
C언어에서 함수란 일반 수학에서의 함수와 비슷하다. 일정한 값을 주면 그 값을 토대로 함수 내부에서 작업을 수행하여 결과를 내놓는다. 함수를 사용하는 이유는 어떤 기능을 반복적으로 실행해야하는 부분이 있으면 함수로 분할하여서 코드를 작성한다. 함수를 사용하면 가독성이 좋아지고 오류수정이 쉽다. 그러나 너무 많이 남발하게 되면 본래의 가독성을 잃고 그 많은 함수들을 일일이 관리하기 힘들다는 단점이 있다.

함수는 아래와 같이 분류할 수 있다.

- 전달인자 O 반환값 O
- 전달인자 O 반환값 X
- 전달인자 X 반환값 O
- 전달인자 X 반환값 X

전달인자가 있다는 것은 함수를 호출할 때 함수로 값을 전달해 준다는 것이다. 마치 이차함수  $f(x)=x^2+x+1$ 이 있다고 가정하면  $f(2)$ 는 7이다. 이와 같이 함수를 호출할 때 함수 내부에서 계산될 값을 전달해 주는데 이것이 전달인자, 그리고 함수에서 전달인자를 받을 때는 매개변수라는 것을 이용하여 값을 받는다. 함수내부에서 처리한 뒤에는 값을 반환한다. 반환값은 함수의 값이다. 값을 반환하는 것은 꼭 그 함수의 자료형을 따라야하며 함수의 자료형이 void일 경우에는 반환하지 않아도 된다. 값을 반환하게 되면 호출되었던 함수는 반환된 값으로 치환된다.

```
#include <stdio.h>
int addone(int s){
    return s+1;
}
int main(){
    int a,result;
    scanf("%d",&a);
    result=addone(a);
    printf("\n\n%d",result);
    return 0;
}
```



정수를 하나 입력받아 함수에 전달하고 s라는 매개변수에 1을 더하여 값을 반환한다.  
위의 소스의 result=addone(a); /\* a=6 \*/ 코드는 함수가 호출된 후에는 result=7; 로 치환된다.

재귀함수

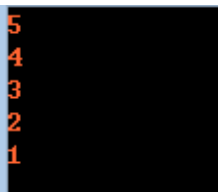
재귀함수는 함수 자신을 재귀적으로 호출하는 함수이다. 장점은 일반함수와 같으나 단점은 다르다. 재귀함수는 함수내부에서 함수를 다시 호출하는 형태이다. 재귀호출을 하면서 함수 내부에서 선언되는 변수는 계속 새로 선언되는 변수들이다.(새로 호출된 함수의 변수와 기존에 호출된 함수의 변수는 서로 다른 것) 즉, 재귀함수에서 끝나는 조건(기저사례)을 확실히 해놓지 않으면 무한루프에 빠지게 되면서 메모리문제를 일으킬 수 있다.

```
#include <stdio.h>
void tobin(int a){
    if(a==0 || a==1){
        printf("%d",a);
        return;
    }
    tobin(a/2);
    printf("%d",a%2);
}
int main(){
    int a;
    scanf("%d",&a);
    tobin(a);
    return 0;
}
```



위에는 정수를 입력받아 이진수로 바꾸는 프로그램을 재귀함수로 작성한 것이다.  
모든 재귀함수는 반복문으로 대체할 수 있다고 한다.

```
#include <stdio.h>
int a=5;
int main(){
    if(a==0){
        return 0;
    }
    printf("%d\n",a--);
    main();
}
```



그리고 main()함수도 함수이기 때문에 재귀함수로 사용가능하다. 그러나 main()함수는 주로 C언어의 뼈대함수, 주 프로그램코드가 동작하는 곳이기 때문에 main()함수를 재귀 호출하는 코드를 짜는 것은 매우 좋지 않은 생각이다.

포인터

변수가 선언하면 메모리의 일정 영역에 해당하는 변수가 생성된다. 그리고 메모리에는 각 부분을 구분하기 위해 주소가 부여되어있으므로 변수에는 대응되는 주소가 있다. 이러한 주소를 저장하기 위한 목적으로 사용되는 것이 포인터이고 포인터변수는 주소를 저장하는 변수이다.

자료형 \*변수;

선언은 위와 같이 한다. int형 포인터변수를 선언하고 싶으면 int \*a;, 실수형은 double \*s;처럼 선언한다.

굳이 주소인데 자료형을 명시해야 하는 이유는

포인터 관련 연산자의 의미

포인터에 관련된 연산자는 &(앰퍼센드) 와 \*(에스테리스크) 가 있다.

```
#include <stdio.h>
int main(){
    int a=10;
    int *p;
    printf("a의 주소 : %x\n",&a);
    p=&a;
    printf("a의 주소를 포인터변수에 대입 : %x\n",p);
    printf("p가 가리키는 값 : %d\n",*p);
    return 0;
}
```

a의 주소 : 28ff08  
a의 주소를 포인터변수에 대입 : 28ff08  
p가 가리키는 값 : 10

일반 변수 앞에 &를 붙이면 그 변수의 주소값을 출력하게 된다. 그러나 포인터변수는 &를 붙이지 않아도 주소를 출력한다. 그 이유는 포인터변수는 처음부터 주소값을 담기위해 설계되었기 때문이다.

\* 는 참조연산자라고 해서 주소를 가지고 있는 변수(포인터변수)앞에 사용하게 되면 변수가 가지고 있는 주소값을 참조한다는 뜻이다. 마지막 출력문 밑에 \*p=123;을 적고 \*p를 정수형식으로 출력해주면 123이 출력될 것이다.

포인터와 배열의 관계

```
#include <stdio.h>
int main(){
    int arr[5]={1,2,3,4,5};
    int *p;
    int i;
    for(p=arr,i=0;i<5;i++){
        printf("%d ",*p++);
    }
    return 0;
}
```

1 2 3 4 5  
Process return  
Press any key

위 코드에서 배열 이름인 arr은 &arr[0], 즉 배열 첫 번째의 주소를 의미하며 배열의 시작점을 알려준다, 상수로 취급한다. 실행하면 포인터변수 p는 arr[]을 가리키게 되고 \*p는 p가 가리키는 주소에 있는 값을 가리키므로 배열의 인자들이 출력된다.

배열을 이용해 선언한 문자열과 포인터를 이용해 선언한 문자열 간의 차이

```
#include <stdio.h>
int main(){
    char arr1[]={ "Unifox" };
    char *arr2;
    arr2="Unifox";
    printf("%s\n%s\n\n", arr1, arr2);
    for(;*arr2;arr2++){
        printf("%s\n",arr2);
    }
    return 0;
}
```

Unifox  
Unifox  
  
Unifox  
nifox  
ifox  
fox  
ox  
x

위 코드를 실행하면 arr1과 arr2가 가리키는 문자열을 출력한다. 그리고 반복문에서는 arr2++를 실행하면 arr2는 두 번째 글자를 가리키게 되고 점점 뒤에 있는 문자를 가리킨다.

scanf 함수에서 &연산자를 쓰는 이유? 배열은 쓰지 않는 이유?

표준입력함수인 scanf()를 사용할 때에는 변수 앞에 ‘&’을 붙이는 것이다. 붙이는 이유는 scanf에 a라는 변수를 넘겨주었다면, scanf는 a라는 변수 자체를 사용하는 것이 아니라 scanf내부에 다른 변수를 선언해서 그 변수에 a값을 대입하고, 대입한 값을 이용하는 것이기 때문이다. 그러나 배열은 배열의 이름이 그 배열의 주소를 가리키는 것과 마찬가지로이기 때문에 쓰지 않지만 각 배열의 요소에 접근하고 싶으면 써야한다.

포인터와 함수의 관계

Call by value (값에 의한 전달)

Call by value는 함수를 호출할 때 전달인자로 단순히 값만을 넘겨주는 형태의 호출을 Call by value라고 한다. 주로 일반적인 함수호출 형태이다. 이 형식으로 전달한 값은 지역변수처럼 함수내부에서만 영향을 끼친다.

Call by reference (주소에 의한 전달)

Call by reference는 값을 전달하는 것이 아닌 변수의 주소값을 전달하는 방법으로, 함수 실행 중에 값을 조정하게 되면 전달인자의 값을 조정가능하다.

두 전달 방법의 차이

값에 의한 전달은 함수의 매개변수에 값만을 전달하지만 주소에 의한 전달은 매개변수(포인터변수)에 주소값을 전달한다. 이 두 방법을 구분할 수 있는 방법은 매개변수를 보고 판단한다. 아래는 두 전달 방법을 모두 이용하는 프로그램코드이다. 밑의 코드에서 주소에 의한 전달을 했을 때 a의 주소와 b의 주소를 전달하면 x와 y에 저장되어 각각 a와 b를 가리킨다. 이 때 x가 가리키는 내용 a와 y가 가리키는 내용인 b를 교환하고 함수를 종료한 후 a와 b를 출력하면 교환된 10 과 5가 출력된다.

```
#include <stdio.h>

void CallbyValue(int x,int y){
    int temp=x;
    x=y;
    y=temp;
}

void CallbyReference(int *x,int *y){
    int temp=*x;
    *x=*y;
    *y=temp;
}

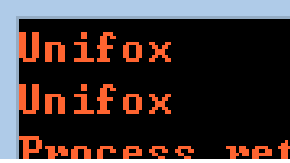
int main(){
    int a=5,b=10;
    ///값에 의한 전달
    CallbyValue(a,b);
    printf("값에 의한 전달 : %d %d\n",a,b);
    ///주소에 의한 전달
    CallbyReference(&a,&b);
    printf("주소에 의한 전달 : %d %d",a,b);
}
```

값에 의한 전달 : 5 10  
주소에 의한 전달 : 10 5  
Process returned 23 (0x17)  
Press any key to continue

함수의 인자로 배열을 전달했을 때의 실행 구조

C언어에서 함수의 매개변수에 배열을 선언을 허용하지 않는다고 한다. 그래서 배열의 주소값을 전달하는 Call by reference를 이용하여 배열을 전달한다. 함수의 매개변수는 전달하려는 함수의 인자는 반드시 그 타입을 가리키는 포인터를 이용해야 한다.

```
#include <stdio.h>
void Print(char *p){
    printf("%c%c%c%c%c%c\n",p[0],p[1],p[2],p[3],p[4],p[5]);
    printf("%s",p);
}
int main(){
    char arr[]={"Unifox"};
    Print(arr);
}
```



arr배열에 문자열을 넣고 문자열의 주소를 함수로 전달, 함수에서는 그것을 포인터변수로 주소를 전달받아 출력한다.  
위와 같은 방법이 가능한 이유는 배열과 포인터는 매우 밀접한 관련이 있기 때문이다. 배열의 이름은 배열의 주소나 마찬가지로 포인터변수에 배열의 이름을 전달할 수 있다.