

프로그램 시작과 종료

main() 함수는 프로그램이 실행되면 가장 먼저 시작되는 부분
위에서 아래로, 좌에서 우로, 문장이 위치한 순서대로 실행

키워드

문법적으로 고유한 의미를 갖는 예약된 단어
‘예약’되었다는 의미는 프로그램 코드를 작성하는 사람이 이 단어들을 다른 용도로 사용해서
는 안된다는 뜻

auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	float	return	typedef	
default	for	short	union	

식별자

- 1.식별자는 영문자(대소문자 알파벳), 숫자(0,9), 밑줄(_)로 구성되며, 식별자의 첫 문자로 숫자가 나올 수 없다.
- 2.프로그램 내부의 일정한 영역에서는 서로 구별되어야 한다.
- 3.키워드는 식별자로 이용할 수 없다.
- 4.식별자는 대소문자를 모두 구별한다. 예를 들어, 변수 Count,count,COUNT는 모두 다른 변수이다.
- 5.식별자의 중간에 공백(space)문자가 들어갈 수 없다.

문장과 주석

문장과 블록

프로그래밍 언어에서 컴퓨터에게 명령을 내리는 최소 단위를 문장이라하며, 문장은 마지막에 세미콜론 ;으로 종료

여러 개의 문장을 묶으면 블록

주석의 정의와 중요성

주석은 타인은 물론이거니와 자신을 위해서라도 반드시 필요하며, 주석에는 자신을 비롯한 이 소스를 보는 모든 사람이 이해할 수 있도록 도움이 되는 설명
잘 처리된 주석이란 시각적으로 정돈된 느낌을 주어야 하며, 프로그램의 내용을 적절히 설명

주석 2가지 처리 방법

한 줄 주석인 `//`은 `//`이후부터 그 줄의 마지막까지 주석으로 인식
블록 주석 `/* ... */`은 여러 줄에 걸쳐 설명을 사용할 때 이용

자료형과 변수 개요

자료형 분류

기본형, 유도형, 사용자정의형 등으로 나눌 수 있으며, 기본형은 다시 정수형, 실수형, 문자형, void로 나뉘는데, 프로그래머는 이러한 자료에 적당한 알고리즘을 적용해 프로그램을 작성

변수

변수에는 고유한 이름이 붙여지며, 물리적으로 기억장치인 메모리에 위치
변수는 선언된 자료형에 따라 변수의 저장공간 크기와 저장되는 자료값의 종류가 결정
저장되는 값에 따라 변수값은 바뀔 수 있으며 마지막에 저장된 하나의 값만 저장 유지

변수선언

컴파일러에게 프로그램에서 사용할 저장 공간인 변수를 알리는 역할이며, 프로그래머 자신에게도 선언한 변수를 사용하겠다는 약속의 의미

1. 자료형을 지정한 후 고유한 이름인 변수이름을 나열하여 표시
2. 변수이름은 관습적으로 소문자를 이용하며, 사용 목적에 알맞은 이름으로 특정한 영역에서 중복되지 않게 붙이도록 한다.
3. 변수선언도 하나의 문장이므로 세미콜론으로 종료
4. 변수선언 이후에는 정해진 변수이름으로 값을 저장하거나 값을 참조

하나의 자료형으로 여러 개 변수를 한번에 선언하려면 자료형 이후에 변수이름을 콤마로 나열

변수의 3요소

변수에서 주요 정보인 변수이름, 변수의 자료형, 변수 저장 값

변수의 이용

변수의 의미는 저장공간 자체와 저장공간에 저장된 값

정수 자료형

정수형 int

정수형의 기본 키워드는 int

short은 int보다 작거나 같고, long은 int보다 크거나 같다

[부호가 있는]을 의미하는 signed 키워드는 정수형 자료형 키워드 앞에 표시

물론 signed 키워드는 생략가능

즉 signed int와 int는 같은 자료형

정수형 저장공간

부호가 없는 정수인 unsigned int는 0과 양수만을 저장할 수 있는 정수 자료형

short는 2바이트며, int와 long은 모두 4바이트

부동소수형 3가지

부동소수형을 나타내는 키워드는 float,double,long double 세 가지

float는 4바이트이며, double과 long double은 모두 8바이트

float형 변수에 저장하면 꼭 3.14F와 같이 float형 상수로 저장

문자형 자료형

char,signed char,unsigned char 세 가지 종류

문자형 저장공간 크기는 모두 1바이트

아스키 코드

아스키 코드는 ANSI에서 제정한 정보 교환용 표준 코드로 총 127 개의 문자로 구성

오버플로와 언더플로

자료형의 범주에서 벗어난 값을 저장하면 오버플로 또는 언더플로가 발생

정수형 자료형에서 최대값+1은 오버플로로 인해 최소값

마찬가지로 최소값-1은 최대값

상수의 종류와 표현 방법

상수는 이름 없이 있는 그대로 표현한 자료값이나 이름이 있으나 정해진 하나의 값만으로 사용되는 자료값

상수는 크게 분리하면 리터럴 상수와 심볼릭 상수로 구분

리터럴 상수

달리 이름이 없이 소스에 그대로 표현해 의미가 전달되는 다양한 자료값

심볼릭 상수

리터럴 상수와 다르게 변수처럼 이름을 갖는 상수

심볼릭 상수를 표현하는 방법

const상수, 매크로 상수, 열거형 상수를 이용

문자 상수 표현

문자 상수는 문자 하나의 앞 뒤에 작은따옴표를 넣어 표현

함수 printf()에서 문자 상수를 출력하려면 다음과 같이 %c 또는 %C의 형식 제어문자가 포함되는 형식 제어문자열을 사용

이스케이프 문자

역슬래쉬 \와 문자의 조합으로 표현하는 문자를 이스케이프 문자

정수와 실수 리터럴 상수

정수형 상수는 int, unsigned int, long, unsigned long, long long, unsigned long long

이진수와 십육진수 표현방식

숫자 0을 정수 앞에 놓으면 팔진수

0x 또는 0X를 숫자 앞에 놓으면 십육진수로 인식

지수표현 방식

실수는 e 또는 E를 사용하여 10의 지수표현 방식

즉 3.14E+2는 3.14×10^2

실수형 리터럴 상수

실수형 상수도 float, double, long double의 자료형

소수는 double 유형이며, float 상수는 숫자 뒤에 f나 F를 붙임

심볼릭 const 상수

변수선언 시 자료형 또는 변수 앞에 키워드 const가 놓이면 이 변수는 심볼릭 상수
상수는 변수선언 시 반드시 초기값을 저장

열거형 상수

키워드 enum

열거형은 키워드 enum을 사용하여 정수형 상수 목록 집합을 정의하는 자료형

열거형 상수에서 목록 첫 상수의 기본값이 0이며 다음부터 1씩 증가하는 방식으로 상수값이
자동으로 부여

매크로 상수

전처리기 지시자 #define

전처리 지시자 #define은 매크로 상수를 정의하는 지시자

출력 함수 printf()

print()의 인자는 크게 형식문자열과 출력할 목록으로 구분

출력 목록의 각 항목을 형식문자열에서 %d와 같이 %로 시작하는 형식지정자 순서대로 서식
화하여 그 위치에 출력

1. 형식 지정자는 출력 내용의 자료형에 따라 %d,%i,%c,%s와 같이 %로 시작
2. 두 번째 인자부터는 형식 지정자에 맞게 콘솔로 출력할 값이 표시되는 연산식 목록이 나열
3. 형식 지정자는 두 번째부터 표시된 인자인 연산식의 수와 값의 종류에 따라 순서대로 서로
일치

출력 폭의 지정

1. 출력 필드 폭이 출력 내용의 폭보다 넓으면 정렬은 기본이 오른쪽이며, 필요하면 왼쪽 지정
가능
2. 형식지정자 %8d는 십진수를 8자리 폭에 출력
3. 지정한 출력 폭이 출력할 내용보다 넓으면 정렬은 기본적으로 오른쪽
4. 출력 폭을 지정하며 정렬을 오른쪽

함수 scanf()

형식지정자는 %d,%c,%lf,%f와 같이 %로 시작
반드시 주소연산자 &를 붙여 나열

제어문자 %f와 %lf와 %c

실수 float형 변수에 저장하려면 형식 지정자 %f를 사용
실수 double형 변수에 저장하려면 형식 지정자 %lf를 사용
문자 char형 변수에 저장하려면 제어문자 %c를 사용

연산식과 연산자 분류

변수와 다양한 리터럴 상수 그리고 함수의 호출 등으로 구성되는 식
반드시 하나의 결과값인 연산값을 가짐
산술연산자 +,-,* 기호와 같이 이미 정의된 연산을 수행하는 문자 또는 문자조합 기호
연산에 참여하는 변수나 상수를 피연산자

연산자는 연산에 참여하는 피연산자의 개수에 따라 단항, 이항, 삼항, 연산자로 나눔
삼항연산자는 조건연산자 '?'가 유일
++a처럼 연산자가 앞에 있으면 전위연산자이며,a++와 같이 연산자가 뒤에 있으면 후위 연산자

산술연산자와 부호연산자

산술연산자 +,-,*,/,%로 각각 더하기,빼기,곱하기,나누기,나머지 연산자

- 1.즉 정수끼리의 나누기 연산 결과는 소수 부분을 버린 정수
- 2.그러나 실수끼리의 연산 10.0/4.0의 결과는 정상적으로 2.5

나머지 연산식 a % b 의 결과는 a를 b로 나눈 나머지 값

부호연산자 +,-

대입연산자와 증감연산자

대입연산자는 =으로 연산자 오른쪽의 연산값을 변수에 저장하는 연산자
대입연산자의 왼쪽 부분에는 반드시 하나의 변수만이 올 수 있다.

축약 대입연산자

대입연산식 $a=a+b$ 는 중복된 a 를 생략하고 간결하게 $a+=b$ 로 쓸 수 있다. 마찬가지로 $a=a-b$ 는 간결하게 $a-=b$ 로 쓸 수 있다.

산술연산자와 대입연산자를 이어 붙인 연산자 $+=, -=, *=, /=, %=$ 을 축약 대입연산자

연산자 ++, --

증가연산자 $++$ 와 감소연산자 $--$ 는 변수값을 각각 1 증가시키고, 1 감소시키는 기능 수행

조건연산자

조건연산자는 조건에 따라 주어진 피연산자가 결과값이 되는 삼항연산자

$x?a:b$

내림변환과 올림변환

내림변환: 큰 범주의 자료형에서(double) 보다 작은 범주인 형(int)으로의 형변환 방식

올림변환: 작은 범주의 자료형(int)에서 보다 큰 범주인 형(double)으로의 형변환 방식

형변환 연산자

명시적 형변환

형변환 연산자 '(type) 피연산자'는 뒤에 나오는 피연산자의 값을 괄호에서 지정한 자료형으로 변환하는 연산자

명시적 형변환==형변환 연산자를 사용한 방식

상수나 변수의 정수값을 실수로 변환하려면 올림변환을 사용

실수의 소수부분을 없애고 정수로 사용하려면 내림변환을 사용

단한연산자인 형변환 연산자는 모든 이항연산자보다 먼저 계산

연산자 sizeof와 콤마연산자

연산자 sizeof는 연산값 또는 자료형의 저장장소의 크기를 구하는 연산자

연산자 sizeof의 결과값은 바이트 단위의 정수

연산자 sizeof는 피연산자가 int와 같은 자료형인 경우 반드시 괄호를 사용

코마연산자 ,

왼쪽과 오른쪽 연산식을 각각 순차적으로 계산하며 결과값은 가장 오른쪽에서 수행한 연산의 결과

조건에 따른 선택 if 문장

문장 if는 위에서 살펴본 조건에 따른 선택을 지원하는 구문

형태는 if (cond) stmt;

if문에서 조건식 cond가 0이 아니면 stmt를 실행하고 0이면 stmt를 실행하지 않는다

문장 if의 조건식 (cond)는 반드시 괄호가 필요하며, 참이면 실행되는 문장은 반드시 들여쓰기를 하도록 한다

조건 만족 여부에 대한 선택 if else

조건문 if (cond) stmt 1; else stmt2;는 조건 cond를 만족하면 stmt1을 실행하고, 조건 cond를 만족하지 않으면 stmt2를 실행하는 문장

반복된 조건에 따른 선택 if else if

cond1 조건식이 참이면 바로 아래의 문장 stmt1를 실행하고 종료되며, 거짓이면 다음 조건식 cond2 로 계속 이어가며, 조건식이 모두 만족되지 않으면 결국 마지막 else 다음 문장 stmt4를 실행

stmt1에서 stmt4에 이르는 여러 문장 중에서 실행되는 문장은 단 하나

중첩된 if

if 문 내부에 if문이 존재하면 중첩된 if문이라 함

Switch

주어진 연산식이 문자형 또는 정수형이라면 그 값에 따라 case의 상수값과 일치하는 부분의 문장들을 수행하는 선택 구문

switch(exp) { ...} 문은 표현식 exp 결과값 중에서 case의 값과 일치하는 항목의 문장 stmt1을 실행한 후 break를 만나 종료

default

switch 문에서 default는 생략될 수 있으며, 그 위치도 제한이 없다
default를 위치시킨 이후에 다른 case가 있다면 break를 반드시 입력

do while문

while문은 반복 전에 반복조건을 평가

do while문은 반복문체 수행 후에 반복조건을 검사

문장 `do stmt; while (cond)`는 가장 먼저 `stmt`를 실행한 이후 반복조건인 `cond`를 평가하여 0이 아니면 다시 반복문체인 `stmt`를 실행하고, 0이면 `do while` 문을 종료

```
do
    stmt;
while(cond);
next;
```

반복 횟수가 정해지지 않고 입력 받은 자료값에 따라 반복 수행의 여부를 결정하는 구문에 유용

반복문체에 특별히 분기 구문이 없는 경우, `do while` 의 문체는 적어도 한 번은 실행되는 특징

`while` 이후의 세미콜론은 반드시 필요

입력 후에 반복 검사를 진행하는 처리 과정으로 `do while` 문으로 구현이 적합

for문

`for (init; cond; inc)`

```
    stmt;

next;
```

for문과 while문의 비교

`for` 문은 주로 반복횟수를 제어하는 제어변수를 사용하며 초기화와 증감부분이 있는 반복문에 적합하다.

`while`문은 반복횟수가 정해지지 않고 특정한 조건에 따라 반복을 결정하는 구문에 적합

분기문

정해진 부분으로 바로 실행을 이용하는 기능 수행

break, continue, goto, return

1. break

반복내부에서 반복을 종료하려면 break 사용

2. continue

continue 문이 위치한 이후의 반복문체의 나머지 부분을 실행하지 않고 다음 반복을 계속 유지

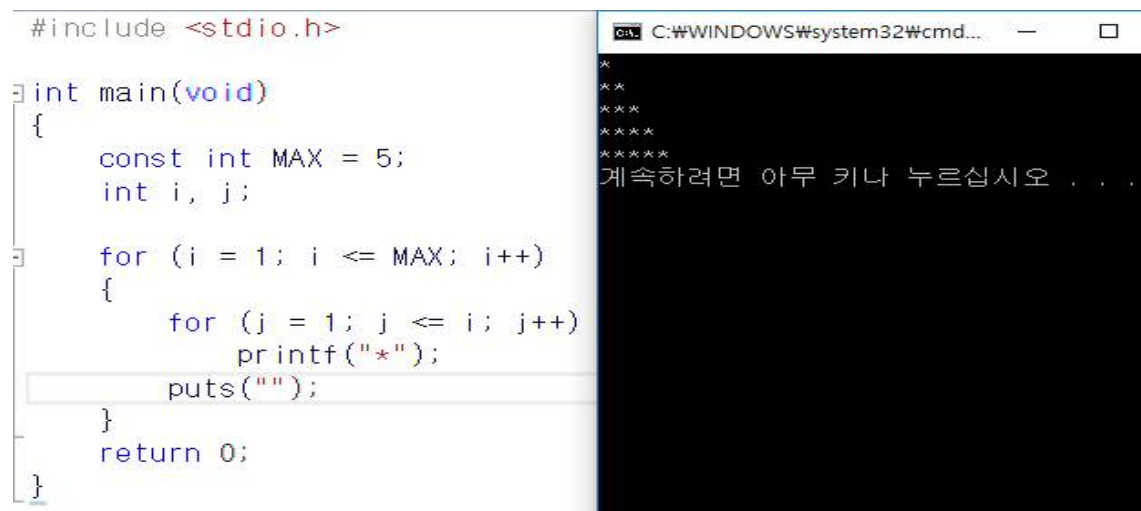
3. goto

goto 문은 레이블이 위치한 다음 문장으로 실행순서를 이동하는 문장

goto==책갈피

중첩된 for문

반복문 내부에 반복문이 또 있는 구문을 중첩된 반복문



```
#include <stdio.h>

int main(void)
{
    const int MAX = 5;
    int i, j;

    for (i = 1; i <= MAX; i++)
    {
        for (j = 1; j <= i; j++)
            printf("*");
        puts("");
    }
    return 0;
}
```

Output:

```
C:\WINDOWS\system32\cmd...
*
**
***
****
*****
계속하려면 아무 키나 누르십시오 . . .
```

주소개념

메모리 공간은 8비트인 1바이트마다 고유한 주소가 있다.

메모리 주소는 0부터 바이트마다 1씩 증가하며 저장 장소인 변수이름과 함께 기억장소를 참조하는 또 다른 방법

주소연산자 &

주소는 변수이름과 같이 저장장소를 참조하는 하나의 방법

&가 피연산자인 변수의 메모리 주소를 반환하는 주소연산자

포인터 변수 선언

포인터 변수 선언에서 자료형과 포인터 변수 이름 사이에 연산자 *를 삽입

int *p; 선언은 int 포인터 p라고 읽는다

```
int data=100;
```

```
int *p;
```

```
p=&data;
```

문장 p=&data;는 포인터 변수 p에 변수 data의 주소를 저장하는 문장

포인터 변수는 가리키는 변수의 종류에 관계없이 크기가 모두 4바이트

```
int data=10;
```

```
int *pdata=&data;
```

```
*pdata=200;
```

여러 포인터 변수선언

```
int *p1,*p2,*p3;
```

int *p1,p2,p3; // p1은 int형 포인터이나 p2,p3는 int형 변수

초기값을 대입하지 않으면 쓰레기값으로 NULL로 초기값 저장

간접참조

```
int data=100;
```

```
int *p=&data;
```

```
printf("간접참조 출력:%d\n",*p);
```

직접참조

```
*p rint=200;
printf("직접참조 출력:%d",data);
```

주소연산

int형 포인터 pi에 저장된 주소값이 100일 경우 (pi+1)은 101이 아니라 주소값 104
(pi+1)은 int형의 바이트 크기인 4만큼 증가한 주소값 104

명시적 형변환

포인터 변수는 동일한 자료형끼리만 대입이 가능하다. 만일 대입문에서 포인터의 자료형이 다르면 경고 발생

```
int value=10;
int *pi=&value;
char *pc=&value; => 포인터의 형변환 경고
```

이중 포인터

포인터 변수의 주소값을 갖는 변수
포인터의 포인터를 모두 다중 포인터

```
int l=20;
int *pi=&i;
int **dpi=&pi;
```

연산자 우선순위

우선순위	단항 연산자	설명	결합성(계산방향)
1	a++ a-- ++a --a	후위 증가, 후위 감소 전위 증가, 전위 감소	->(좌에서 우로)
2	& *	주소 간접 또는 역참조	<-(우에서 좌로)

*p++는 *(p++)으로 (*p)++와 다르다
++*p는 ++(*p)와 같다
***p는 *(++p)와 같다

포인터 상수

키워드 **const(바꿀수없음)**를 이용하는 변수 선언은 변수를 상수로 만들 듯이 포인터 변수도 포인터 상수로 만들 수 있다.

```
int I=10,j=20;
const int *pi=&i;
```

배열

배열은 여러 변수들이 같은 배열이름으로 일정한 크기의 연속된 메모리에 저장되는 구조
배열을 이용하면 변수를 일일이 선언하는 번거로움을 해소 할 수 있고,배열을 구성하는 각각의 변수를 참조하는 방법도 간편하며,반복 구문으로 쉽게 참조
배열은 한 자료유형의 저장공간인 원소를 동일한 크기로 지정된 배열크기만큼 확보한 연속된 저장공간

배열선언 구문

원소자료형 배열이름[배열크기];
배열크기는 리터럴 상수,매크로 상수 허용
변수와 const 상수로는 배열의 크기를 지정 할 수 없음

배열원소 접근

배열선언 후 배열원소에 접근하려면 배열이름 뒤에 대괄호 사이 첨자를 이용
배열에서 유효한 첨자의 범위는 0부터 (배열크기-1)까지
배열선언 후 원소에 초기값을 저장하지 않으면 쓰레기값이 저장되니 **항상 초기값을 저장**

```
int score[5];

score[0]=78;
score[1]=52;
score[2]=16;
score[3]=48;
score[4]=27;
score[5]=82; //실행오류
```

배열선언 초기화

원소자료형 배열이름[배열크기]={원소값1,원소값2,원소값3,원소값4,원소값5, ... };



배열크기는 생략 가능하며, 생략 시 원소값의 수가 배열크기가 됨

2차원 배열선언

이차원 배열선언은 2개의 대괄호가 필요

첫 번째 대괄호에는 배열의 행 크기, 두 번째는 배열의 열 크기

배열선언 시 초기값을 저장하지 않으면 반드시 행과 열의 크기는 명시

원소자료형 배열이름[배열행크기][배열열크기];

```
int td[2][3];
```

```
td[0][0]=1; td[0][1]=2; td[0][2]=3;
```

```
td[1][0]=4; td[1][1]=5; td[1][2]=6;
```

2차원 배열선언 초기화

```
int score[2][3]={{30,44,67},{87,43,56}};
```

```
int score[2][3]={30,44,67,87,43,56};
```

```
int score[][3]={30,44,67,87,43,56}; // 열의 크기는 반드시 명시
```

3차원 배열

실제로 삼차원 이상의 배열을 사용하는 경우는 드뭄

3차원 배열선언

```
int threed[2][2][3];
```

```
threed[0][0][0]=1;
```

```
threed[0][0][1]=1;
```

```
threed[0][0][2]=1;
```

```
threed[0][1][0]=1;
```

...

일차원 배열과 포인터

```
int score[]={89,98,76};
```

배열이름 score는 배열 첫 번째 원소의 주소를 나타내는 상수로 &score[0]와 간접연산자를 이용한 *score는 변수 score[0]와 같다.

배열 초기화 문장	int score[]={89,98,76};			
배열 원소값		89	98	76
배열원소 접근방법	score[i]	score[0]	score[1]	score[2]
	*(score+i)	*score	*(score+1)	*(score+2)
주소값 접근 방법	&score[i]	&score[0]	&score[1]	&score[2]
	score+i	score	score+1	score+2
실제 주소값	base + 원소크기*i	만일 4라면	8=4+1*4	12=4+2*4

이차원 배열과 포인터

```
int td[][3]={{8,5,4},{2,7,6}};
```

배열이름 td는 이차원 배열을 대표하는 이중 포인터
sizeof(td)는 배열전체의 바이트 크기를 반환

td[0][0]의 값을 20으로 수정하려면 **td=20;

포인터 배열

포인터 배열이란 주소값을 저장하는 포인터를 배열원소로 하는 배열

포인터 배열 선언

자료형 *변수이름[배열크기];

```
int *pary[5];
```

```
char *ptr[4];
```

열이 4인 이차원 배열 ary[][4]의 주소를 저장하려면 배열포인터 변수 ptr을 문장
int(*ptr)[4];로 선언

배열 크기 연산

연산자 sizeof를 이용한 식 (sizeof(배열이름) / sizeof(배열원소))

sizeof(배열이름)==배열의 전체 공간의 바이트 수

sizeof(배열원소)==배열원소 하나의 바이트 수

이차원 배열크기 계산방법

이차원 배열의 행의 수

(sizeof(x) / sizeof(x[0]))

이차원 배열의 열의 수

(sizeof(x[0] / sizeof(x[0][0])))

sizeof(x)==배열 전체의 바이트 수

sizeof(x[0])는 1행의 바이트 수이며, sizeof(x[0][0])는 첫 번째 원소의 바이트 수

함수 개념

특정한 작업을 처리하도록 작성한 프로그램 단위

함수는 라이브러리 함수와 사용자 정의 함수로 구분

사용자가 직접 개발한 함수를 사용하기 위해서는 함수선언,함수호출,함수정의 필요

절차적 프로그래밍

적절한 함수로 잘 구성된 프로그램을 모듈화 프로그램 또는 구조화된 프로그램

한번 정의된 함수는 여러 번 g hcnfdl 가능하므로 소스의 중복을 최소화하여 프로그램의 양을 줄임

함수 중심의 프로그래밍 방식

함수정의

함수머리와 함수몸체로 구성

함수헤더 { 반환형 함수이름(매개변수 목록)

{ ...
함수몸체 { 여러 문장들;
{ return (반환연산식);
}

```
int add2(int a, int b)
{
    int sum=a+b;

    return (sum);
}
```

반환형과 return

함수가 반환값이 없다면 반환형으로 void를 기술

return 문장은 함수에서 반환값을 전달하는 목지와 함께 함수의 작업 종료

함수선언과 함수호출

함수실행

정의된 함수를 실행하려면 프로그램 실행 중에 함수호출이 필요

함수원형

함수를 선언하는 문장

매개변수의 변수이름은 생략할 수 있다

함수선언으로 변수선언과 같이 함수를 호출하기 전에 반드시 선언

반환형 함수이름(매개변수 목록);

```
int add2(int a, int b);
int add2(int, int);
```

매개변수

매개변수는 함수를 호출하는 부분에서 함수몸체로 값을 전달할 목적으로 이용
함수정의에서 매개변수는 필요한 자료형과 변수명의 목록으로 나타내며 필요 없으면 키워드
void를 기술

형식매개변수

함수정의에서 기술되는 매개변수 목록의 변수
함수 내부에서만 사용할 수 있는 변수

재귀 함수

자신 함수를 호출하는 함수
함수의 호출이 계속되면 시간도 오래 걸리고 메모리의 사용도 많다

난수 라이브러리 함수

함수 rand()

특정한 나열 순서나 규칙을 가지지 않는 연속적인 임의의 수
함수 rand()의 함수원형은 헤더파일 stdlib.h
0~32767사이의 정수

함수 srand()

함수 time(NULL)은 1970년 1월 1일 이후 현재까지 경과된 시간을 초 단위로 반환하는 함수

수학과 문자 라이브러리 함수

math.h

수학 관련 함수를 사용하려면 헤더파일 math.h을 삽입

헤더파일 ctype.h

문자 관련 함수는 헤더파일 ctype.h에 매크로로 정의