프로그래밍 언어

프로그래밍 언어의 분류

- 절차적 프로그래밍 언어
  - ✓ 코드를 순서대로 읽어서 실행하는 언어
  - ✓ ALGOL : 알고리즘 개발을 만든 언어로 C 와 PASCAL의 모체
  - ✓ C: UNIX 운영체제를 개발하기 위해서 만들어진 언어
  - ✓ COBOL: 영문장과 유사한 형태로 작성하는 사무 처리용 언어
  - ✓ FORTRAN: 과학 기술 계산용 언어

- 비절차적 프로그래밍 언어
  - ✓ 코드 작성 순서와 상관없이 실행되는 언어
  - ✓ SQL이 대표적이며 객체 지향 언어들도 비절차적 언어

- 구조적 프로그래밍 언어
  - ✓ 프로그래밍을 할 때 블럭(제어문, 함수, 클래스 등)단위로 나누어서 작성이 가능한 언어
  - ✓ 유지보수가 편리해지고 재사용성이 증가
  - ✓ pascal: 교육용으로 개발된 언어
  - ✓ 현재 사용되는 대부분의 언어는 구조적 프로그래밍 언어
  - ✓ 블럭 대신에 모듈이라고도 함

프로그래밍 언어의 분류

- 객체지향 프로그래밍 언어
  - ✓ 동일한 목적을 위하여 만들어진 변수와 함수를 클래스나 인스턴스(객체)로 묶어서 사용하는 언어
  - ✓ 캡슐화, 상속, 다형성을 제공해서 재사용성을 증가시키고 유지보수를 편리하게 함
  - ✓ Smalltalk: 초창기 객체 지향 언어로 C++ 이나 Java 등에 영향을 미침
  - ✓ C++: C언어의 문법을 지원하는 객체지향 언어
  - ✓ Java: 완전환 객체 지향 언어로 반드시 클래스를 만들어서 코드를 작성해야 함
  - ✓ 그 이외에도 Objective-C, C# 등 최근에 등장한 언어들은 대부분 객체 지향의 개념을 가지고 있음
- 스크립트 언어
  - ✓ 코드를 미리 번역하지 않고 줄 단위로 읽어가면서 실행하는 언어
  - ✓ JavaScript: 웹 브라우저 안에서 동작하는 언어
  - ✓ ASP, JSP, PHP, Python, Kotlin, Swift 등이 대표적인 스크립트 언어
- 선언형 언어
  - ✓ 알고리즘을 기술하지 않고 문제만 기술하는 언어로 명령형 언어의 반대
  - ✓ HTML, CSS, XML, LISP, PROLOG, Haskell

객체 지향 용어

- 클래스: 유사한 역할을 수행하는 객체들의 모임으로 사용자 정의 자료형
- 객체: 동일한 목적을 위해 모인 데이터들과 이에 대한 연산을 가지는 것으로 클래스를 기반으로 생성하면 Instance 라고도 함
- 캡슐화(Encapsulation): 객체 지향 방법에 캡슐화는 객체 안에 데이터와 연산들을 패키지화한 것으로 정보 은폐를 통해 객체의 세부적 구현을 은폐하므로 변경 작업시에 부작용의 전파를 최소화할 수 있으며 캡슐화된 기능은 다른 클래스에서 재사용을 용이하게 한다.
- 속성(Property) : 각 객체가 가지고 있는 정보로서 객체의 성질, 분류, 식별, 수량 또는 상태 등을 표현
- 메소드(method)는 객체에 정의한 연산을 의미하며, 이것은 객체의 상태를 참조 및 변경하는 수단이며 객체가 메시지를 받으면 메소드를 수행
- 메세지: 객체들은 메시지(message)를 통해 정보를 교환하는데 객체와 객체 사이의 통신수단
- 캡슐화와 정보 은닉: 캡슐화는 데이터와 이에 대한 연산을 합한 것을 의미하며, 캡슐화의 장점은 정보 은닉(information hidden)을 의미하며 캡슐화된 객체는 외부 인터페이스만을 통하여 접근하며, 캡슐화를 하면 결합도는 낮아지고, 응집도는 높아지며, 변경 시의 부작용을 방지하며 정보 은폐를 통해 객체의 세부적 구현을 은폐하므로 변경 작업 시에 부작용의 전파를 최소화할 수 있고 캡슐화하여 처리하므로 인터페이스가 단순 해진다.
- 상속성(Inheritance): 상위 클래스의 메소드에 존재하는 모든 속성을 하위 클래스가 계승하는 것으로서, 하위 클래스는 상위 클래스의 메소드 및 모든 속성을 공유하며, 소프트웨어 재사용과 유지 보수를 증대시키며 슈퍼 클래스는 상속하는 클래스, 서브클래스는 상속받는 클래스이다.
- 다형성(Polymerphism): 동일한 메시지에 대하여 서로 다르게 반응하는 성질

Data Type

- 데이터 타입
  - ✓ 데이터를 어떻게 저장하고 얼마만큼의 메모리 크기를 할당할 것인지의 여부
  - ✓ 저장된 데이터의 종류에 따른 분류
    - Value Type(기본형, 값형): 데이터 자체를 저장하는 방식
    - Reference Type(참조형): 데이터의 참조를 저장하는 방식
  - ✓ 데이터의 개수에 따른 분류
    - Scala Type: 데이터 1개를 저장하는 방식으로 그 자체가 데이터
    - Vectory Type: 여러 개의 데이터를 묶어서 저장하는 방식으로 그 자체는 데이터들이 저장되어 있는 곳의 참조로 Array(배열 – List), Class(사용자 정의 자료형 – Object, Instance)

Data Type

- ❖ 숫자 데이터 타입
  - ✓ 정수
    - C언어
      - char(1byte)
      - unsinged char(1byte)
      - short(2byte)
      - unsigned short(2byte)
      - int(4byte)
      - unsigned int (4byte)
      - long (4byte)
      - unsigned long (4byte)
      - long long (8byte)
    - Java: byte(1byte) -> short(2byte) -> char(2byte) -> int(4byte) -> long(8bute)
    - Python: int – 메모리가 허락하는 무제한

Data Type

- 숫자 데이터 타입
  - ✓ 정수
    - 1byte는 8bit 히고 1bit는 0과 1 둘 중에 하나를 저장할 수 있음
    - 1bit가 있으면 2가지 모양을 나타낼 수 있음
    - 8bit이면 8개의 2가지 모양이 만들어지므로 256가지 모양을 만들 수 있음
    - 정수에서 양수는 0부터 시작하고 음수는 -1부터 시작
    - 양수와 음수 모두 나타내야 한다면 1byte를 가지고 -128 ~ +127까지 나타낼 수 있고 음수가 없다면 0~255까지 표현 가능
    - C 언어에서 unsigned 가 앞에 있는 자료형이 음수가 없는 자료형임
    - Java 나 Python에는 unsigned 가 없음
    - C언어에서 char는 영문과 숫자만 나타내며 Java에서는 영문 그리고 숫자 및 한글이 모두 가능
    - 문자는 코드로 변환해서 정수로 저장하며 출력을 할 때 문자로 변경해서 출력되기 때문에 정수로 취급
    - Python에는 문자 자료형이 없음

Data Type

- 숫자 데이터 타입
  - ✓ 실수
    - C언어
      - float(4byte)
      - double(8byte)
      - long double(8byte)
    - Java
      - float(4byte)
      - double(8byte)
    - Python
      - float(8byte)
    - C 와 Java에서는 실수를 여러가지로 구분하고 Python은 구분하지 않음
    - 메모리 크기가 커지면 정밀도가 높아지며 표현의 범위도 커지지만 메모리 크기가 작으면 정밀도나 표현의 범위가 작아짐
  - ✓ Python에는 복소수를 저장할 수 있는 complex가 있음

Data Type

- boolean(참, 거짓)
  - ✓ C언어: 별도의 자료형이 존재하지 않으며 0이면 거짓 그리고 0이 아닌 숫자는 참으로 간주
  - ✓ Java: false 와 true로 구분
  - ✓ Python: bool 타입으로 False 와 True로 구분
    - 0이 아닌 숫자는 True 0은 False로 간주
    - 데이터가 있으면 True 없으면 False로 간주
- 배열
  - ✓ 동일한 자료형의 데이터를 연속적으로 저장한 것으로 크기가 고정
  - ✓ 하나의 이름으로 여러 개의 데이터를 사용하기 위해서 생성
  - ✓ 초기화와 생성을 동시에 하기

    자료형 배열명 [ ] = {데이터 나열};

    int ar[] = {10,20,30};
  - ✓ 생성만 하기

    자료형 배열명[개수]; //C

    int ar[7];

    자료형 배열명[ ] = new 자료형[개수]; //Java

    int ar[] = new int[7];

Data Type

- 배열
  - ✓ 배열의 데이터 접근 : 배열이름[인덱스]
  - ✓ 배열의 데이터 수정 : 배열이름[인덱스] = 데이터;
  - ✓ 인덱스는 0부터 자신의 데이터 개수 – 1 까지
  - ✓ 데이터 개수
    - C언어 : sizeof(배열이름) / sizeof(배열이름[0])
    - Java : 배열이름.length

```
int ar[ ] = {10, 20, 40};
ar[2] = 30;
printf("%d", ar[2])); //30

ar[4] = 50; //에러 – Java는 ArrayIndexOutOfException
```

Data Type

- Literal: 프로그래밍 언어에서 고정된 값
  - ✓ 정수
    - 10진 정수: 107
    - 8진수: 0o107(Java는 o를 생략)
    - 16진수: 0x107
    - 2진수: 0b101
  - ✓ 실수
    - float 형 실수: 1.7f
    - double 형 실수: 1.7
    - 지수 형태로 입력 하는 것도 가능: 1.7E+01, 1.7E-1
  - ✓ 문자: C, Java – '문자', Python은 문자와 문자열을 구분하지 않음
  - ✓ boolean
    - ✓ Java: true, false
    - ✓ Python: True, False
  - ✓ 문자열: "문자열", Python은 '문자열'이 가능하고 3번씩 입력해서 여러 줄 문자열도 가능
  - ✓ null: null, Python에서는 None

Variable 과 Constant

- Variable(변수)
  - ✓ 데이터를 저장한 공간에 붙이는 이름으로 데이터를 나중에 다시 사용하기 위해서 생성
  - ✓ 자료형 변수명 = (초기값); 의 형태로 생성하며 C언어에서는 초기값이 없으면 Garbage 값을 가지며 Java에서는 변수가 생성되지 않고 나중에 값을 입력하면 생성되며 Python은 자료형을 입력할 필요가 없음
  - ✓ 자료형과 초기값의 Data Type은 일치해야 함
  - ✓ 변수명은 _ 그리고 영문 으로 시작하고 이후에는 _ 그리고 영문, 숫자 모두 사용 가능(Java는 한글 가능)
  - ✓ 중간에 공백은 포함할 수 없음
  - ✓ Keyword(예약어 – 프로그래밍 언어가 기능을 정해놓은 명령어)는 변수명으로 사용할 수 없음
- Constant(상수)
  - ✓ 값을 변경할 수 없도록 만드는 것
  - ✓ C언어에서는 자료형 앞에 const를 기재
  - ✓ Java언어에서는 자료형 앞에 final을 기재
  - ✓ 이름은 변수와 구분하기 위해서 모두 대문자로 하는 것이 관례

Variable 과 Constant

- 변수의 구분
  - ✓ Local Variable: 선언된 영역에서만 사용 가능한 변수
  - ✓ Class & Instance Variable: Class 나 Instance 로 사용이 제한된 변수 – Member Variable 또는 Property라고 함
  - ✓ Global Variable: 모든 영역에서 사용이 가능한 변수

- 기억 클래스
  - ✓ Auto(자동): 자신의 블럭 내에서만 사용되는 변수
  - ✓ External(외부): 블럭 외부에서 선언되서 다른 블럭에서 사용이 가능한 변수
  - ✓ Static(정적): 블록 내부에 만들더라도 생성은 1번만 이루어지며 블록이 종료되도 소멸되지 않는 변수
  - ✓ Register: CPU 내의 레지스터에 만들어지는 변수

Input & Ouput

- C언어의 Input
  - ✓ scanf(입력서식, 입력받을 참조)
    - 입력받을 참조
      - 데이터 1개를 저장할 수 있는 변수의 경우는 &변수명
      - 여러 개의 데이터를 저장할 수 있는 변수(배열)의 경우는 변수명
    - 입력 서식
      - %d: 정수(ld -> long)
      - %u: 부호없는 정수
      - %o: 8진수(lo -> long)
      - %x: 16진수(lo -> long)
      - %c: 하나의 문자
      - %f: 실수
      - %e: 지수형 실수
      - %p: 주소를 16진수로 사용
      - %s: 주소에 해당하는 위치부터 null을 만날때까지 문자로 입력받는데 공백이나 Enter를 누르면 거기까지만 입력받은
      - 서식문자 앞에 숫자를 설정하면 숫자만큼 만 입력을 받음

Input & Ouput

- C언어의 Output
  - ✓ printf(출력서식, 출력할 표현식)
    - 출력할 표현식
      - 상수, 변수
      - 연산식
      - 리턴이 있는 함수 호출
    - 출력 서식
      - %d: 정수
      - %u: 부호없는 정수
      - %o: 8진수(lo -> long)
      - %x: 16진수(lo -> long)
      - %c: 하나의 문자
      - %f: 실수
      - %e: 지수형 실수
      - %p: 주소를 16진수로 사용
      - %s: 주소에 해당하는 위치부터 null을 만날때까지 문자로 출력

Input & Ouput

- C언어의 Output
  - ✓ printf(출력서식, 출력할 표현식)
    - 출력 서식
      - %숫자d: 정수를 숫자만큼의 자리를 확보해서 오른쪽 맞춤을 해서 출력하는데 숫자가 원래 숫자의 개수보다 작으면 전체를 출력하고 음수를 사용하면 왼쪽 맞춤을 해서 출력하며 앞에 0을 추가하면 남는 자리는 0으로 채움
      - %전체자릿수.소수자릿수f: 실수를 출력할 때 소수 자릿수를 설정하면 그 아래 자리에서 반올림
      - %전체자릿수.출력할개수s: 전체 자리를 설정해서 출력할 개수만큼만 출력
    - 제어문자: ₩다음에 하나의 문자를 추가해서 특수한 기능을 하도록 해주는 문자
      - ₩n: 줄 바꿈
      - ₩t: 탭
      - ₩₩:₩
      - ₩', ₩":', "
- Java 에서의 입출력
  - ✓ 입력은 java.util.Scanner
  - ✓ 출력은 System.out.printf()를 이용해서 동일한 방식으로 가능하면 System.out.print 나 System.out.println 을 사용할 수 있음

연산자

❖ 연산자의 우선순위와 기능별 분류

연산자 명칭		연산자	결합 규칙
최우선 연산자		함수, 괄호(), 배열[], 구조체->	왼쪽→오른쪽
단항 연산자		*(참조형 변수에 저장된 값), &, -, ~, ++, --, (type), sizeof	왼쪽←오른쪽
이항 연산자	승제 연산자	*, /, %	**왼쪽→오른쪽**
	가감 연산자	+, -	
	Shift 연산자	>>, <<	
	관계 연산자	<, <=, >, >=	
	등가 연산자	==, !=	
	AND 연산자	&	
	XOR 연산자	^	
	OR 연산자	|	
	논리곱 연산자	&&	
	논리합 연산자	||	
조건 연산자		? :	왼쪽←오른쪽
할당 연산자		=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=	왼쪽←오른쪽
콤마 연산자		,	왼쪽→오른쪽

증감연산자(++,--)

- 예제

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int a = 0, b;
        b = ++a + ++a;
        // 연산자의 우선순위에 따라 a의 값이 1씩 2번 증가한 후 더하기 연산을 하므로 연산의
결과는4
        printf("%d\n", b);
        a=0;
        b = a-- + a--;
        printf("%d\n", b);
        a=0;
        b= ++a + ++a + ++a;
        printf("%d\n", b);
        printf("아무키나 누르면 종료됩니다.");
        getch();
        return 0;
}
```

```
4
0
9
```

Shift 연산자

❖ 정수를 2진수로 변경해서 비트단위 이동을 수행하는 연산자

연산자	형식	의미
>>	피 연산자 >> 숫자	피 연산자를 숫자만큼 오른쪽으로 shift 첫 번째 비트가 삽입
<<	피 연산자 << 숫자	피 연산자를 숫자만큼 왼쪽으로 shift 0이 삽입

short a = 20; a >> 2 이렇게 되면 a의 오른쪽 두 비트가 잘려나가고 왼쪽에 0이 두 비트 삽입되는데 첫번째 비트는 변경되지 않습니다.

먼저 20을 표현하면 다음과 같습니다.

00000000 00010100

a >> 2 하게 되면 다음과 같습니다.

00000000 00000101

따라서 a>>2 의 결과는 10진수 5가 됩니다.

만일 a<<2 하게 되면 반대로 왼쪽의 두 개의 비트가 잘려나가고 오른쪽 끝에 0이 두 비트 삽입됩니다.

a << 2의 결과는 다음과 같습니다.

00000000 01010000

따라서 a << 2의 결과는 10진수 80이 됩니다.

32번을 shift하게 되면 모든 데이터가 shift 되기 때문에 32이상의 shift는 32로 나눈 나머지 만큼만 shift 합니다.

Shift 연산자

- 예제

```
#include <stdio.h>
int main()
{
        int a = 20;
        printf("%d\n", a >> 2);
        printf("%d\n", a << 2);
        system("pause");
        return 0;
}
```

```
5
80
```

Shift 연산자

- 예제

```
#include <stdio.h>
int main()
{
        int a = 20;
        printf("%d\n", a << 26);
        //Overflow
        printf("%d\n", a << 27);
        //32로 나눈 나머지 만큼만 shift 하기 때문에 하지 않은 것과 동일한 효과
        printf("%d\n", a << 32);
        system("pause");
        return 0;
}
```

```
1342177280
-1610612736
20
```

논리연산자

- 정수를 2진수로 변환해서 비트 단위로 연산을 한 후 결과는 10진 정수로 리턴하는 연산자
  - ✓ 비트 AND 연산은 양 쪽의 비트가 모두 1이면 1이고 그 외는 0을 리턴하는 연산
  - ✓ 비트 OR 연산은 양 쪽의 비트가 모두 0이면 0이고 그 외는 1을 리턴하는 연산
  - ✓ 비트 XOR 연산은 양 쪽의 비트가 동일하며 1이고 다르면 0을 리턴하는 연산
  - ✓ 비트 NOT 연산은 0이면 1 1이면 0으로 반전시키는 연산
  - ✓ 모든 비트를 각각의 비트 단위로 논리 연산한 후 이 결과를 정수로 변환하여 리턴
  - ✓ 컴퓨터에서는 모든 수나 문자는 이진수로 저장

short a = 20; 라고 변수를 초기화하게 되면 실제로 컴퓨터에 저장된 형태는 다음과 같습니다.
00000000 00010100
short b = 12; 이렇게 초기화하면 변수 b 는 다음과 같이 저장되게 됩니다.
00000000 00001100
a & b의 결과는 다음과 같습니다.
00000000 00000100 연산의 결과는 10진수 4가 됩니다.
a | b의 결과는 다음과 같습니다.
00000000 00011100 따라서 연산의 결과는 10진수 28이 됩니다.
a ^ b의 결과는 다음과 같습니다.
00000000 00011000 따라서 연산의 결과는 10진수 24가 됩니다.

논리연산자

- 예제

```
#include <stdio.h>
int main()
{
        printf("%d\n", 20 & 12);
        printf("%d\n", 20 | 12);
        printf("%d\n", 20 ^ 12);
        system("pause");
        return 0;
}
```

```
4
28
24
```

논리연산자

- && 연산이나 ||연산은 피 연산자로 true 또는 false 형태의 데이터가 올 수 있으며 결과가 1 또는 0으로 리턴
- &&: 왼쪽의 결과가 참이면 오른쪽을 조사해서 참이라면 1을 리턴하게 되고 거짓이라면 0을 리턴하게 되며 왼쪽이 거짓이라면 오른쪽은 검사하지 않고 0을 리턴합니다.
- ||: 왼쪽이 참이면 무조건 1을 리턴하고 오른쪽은 조사하지 않으며 만일 왼쪽이 거짓이라면 오른쪽을 검사하여 참이면 1을 리턴하고 아니면 0을 리턴합니다.
- &&이나 || 연산자의 한쪽에 상수도 가능합니다.
- 상수를 이용하면 0이 아닌 양수는 모두 true로 간주합니다.

논리연산자

- 예제

```
#include <stdio.h>
int main()
{
        int a = 10;
        printf("상수를 이용한 결과: %d\n", a > 10 && 5);
        printf("결과: %d \t a의 값: %d\n", a >= 10 && ++a, a);
        a = 10;
        printf("결과: %d \t a의 값: %d\n", a <= 10 && ++a, a);
        system("pause");
        return 0;
}
```

```
상수를 이용한 결과: 0
결과: 1         a의 값: 11
결과: 1         a의 값: 11
```

조건 연산자

- 조건연산자는 피 연산자를 세 개를 가지는 연산자입니다.
- 삼항 연산자라고도 합니다.
- 사용형식

  수식 ? 문장1: 문장2
- 수식이 참이면 문장1을 수행하고 거짓이면 문장2를 수행하게 됩니다.
- 수식은 되도록이면 괄호로 구분해주는 것이 좋습니다.

조건 연산자

- 예제

```
#include <stdio.h>
int main()
{
        int a, b;
        a = 5;
        b = 3;
        printf("큰 값= %d\n", (a > b) ? a : b);
        printf("작은값= %d\n", (a < b) ? a : b);
        system("pause");
        return 0;
}
```

```
큰 값 = 5
작은값 = 3
```

할당 연산자

- 할당 연산자는 대입 연산자라고도 합니다.
- 할당 연산자는 오른쪽의 피 연산자의 값을 왼쪽의 변수에 대입하는 연산자를 의미합니다.
- 할당 연산자의 왼쪽에는 반드시 메모리 공간(변수)이 와야 하며 오른쪽에는 메모리 공간 또는 상수가 올 수 있으며 오른쪽에 변수가 오는 경우에는 변수가 저장하고 있는 값입니다.
- C언어에서의 =의 의미는 우리가 흔히 수학에서 표현하는 equel이 아니라 할당(assignment)의 의미

연 산 자	형 식	내 용
=	a=b	a = b
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
&=	a &= b	a &= b
^=	a ^= b	a = a ^ b
|=	a |= b	a = a | b
<<=	a <<= b	a = a << b
>>=	a >>= b	a = a >> b

할당 연산자

❖ 실제로 C언어에서 할당 연산자로 프로그램을 더 효율적으로 작성할 수 있는데, 이것은 대입 연산자의 식 자체가 어떤 값(연산자의 좌측 피 연산자에 대입된 값)을 가진다는 것에 기인하게 됩니다.

```
a = 0;
b = 0;
c = 0;
```

위 세 개의 문장은 a = b = c = 0; 으로 압축될 수 있습니다.

a += 1; → a = a + 1; 등의 표현도 가능

❖ 이들 할당 연산자가 프로그램 내에서 특정 수식과 함께 활용 될 때 결합 순서에 주의해야 하는데 할당 연산자가 다른 연산자보다 연산순위가 낮다는 것만 주의하여 프로그램을 작성하면 올바른 값을 얻을 수 있을 것입니다.

a *= 3 + b 와 a = a * 3 + b는 다릅니다.

첫 번째 문장은 3+b를 먼저 한 후에 a를 곱해서 대입하게 되며

뒤의 문장은 먼저 a 에 3을 곱한 후에 b를 더하게 됩니다.

할당 연산자

- 예제

```
#include <stdio.h>
int main()
{
        int i = 1, j = 2;
        int s = 0;
        s = 100;
        printf("%d\t", s);
        s += j;
        printf("%d\t", s);
        s -= j;
        printf("%d\t", s);
        s *= j;
        printf("%d\t", s);
        s /= 10;
        printf("%d\t", s);
        s %= 7;
        printf("%d\n", s);
        system("pause");
        return 0;
}
```

```
100     102     100     200     20      6
```

연습문제

- ++변수 와 변수++의 차이를 설명하시오.
- (이항연산자)& 와 && 그리고 | 와 || 의 차이를 설명하시오.

제어문
(Control Statement)

제어문은 실행의 흐름을 변경하여 프로그램의 실행을 프로그래머가
임의대로 바꿀 수 있게 해주고 특정한 블럭을 반복수행을 하게도 해주는 명령어

1. 조건문(분기문)

- 조건문은 조건에 따라서 수행해야 하는 동작이 다를 때 사용하는 블럭
- 조건문에는 if와 if – else, switch 문이 있습니다.
- if 문은 조건 식을 판단하여 참 또는 거짓의 논리적인 결과에 따라 그 수행을 다르게 하고자 하는 경우에 사용하는 제어문
- 단순 if 문 형식

```
if(조건식)
{
    블럭 1;
}
블럭 2;
```

- 단순 if 문은 조건식이 참이면 블럭1을 수행하고 블럭2로 넘어가게 되고 거짓이면 블럭1을 수행하지 않고 블럭2로 넘어가게 됩니다.
- 만일 조건식이 참인 경우 수행해야 하는 블럭이 1개의 문장이라면 { }를 생략 할 수 있으며 여러 개의 문장 인 경우에는 반드시 { } 에 기재해주어야 합니다.

1. 조건문(분기문)

- 예제

```
#include <stdio.h>
int main() {
		int kor,eng;
		printf("국어점수를 입력하시오(0-100):");
		scanf("%d",&kor);
		printf("영어점수를 입력하시오(0-100):");
		scanf("%d",&eng);
		if (kor<eng)  printf("영어점수가 더 높습니다.\n");
		if (kor>eng)  printf("국어점수가 더 높습니다.\n");
		if (kor==eng)  printf("국어점수와 영어점수가 같습니다.\n");
		if (((kor+eng)/2)>= 60)
				printf("당신은 평균이 60점 이상이므로 합격입니다.\n");
		if (((kor+eng)/2)< 60)
				printf("당신은 평균이 60점이 안되므로 불합격입니다.\n");
		return 0;
}
```

```
국어 점수를 입력하시오: 80
영어 점수를 입력하시오: 90
영어 점수가 더 높습니다.
당신은 평균이 60점 이상이므로 합격입니다.
```

1. 조건문(분기문)

- if-else문: 조건이 참인 경우와 조건이 거짓인 경우에 대해 수행해야 하는 내용이 다를 때 사용하는 제어문
- if – else 문의 사용형식

```
if(조건식)
{
    블럭 1;
}
else
{
    블럭 2;
}
다음블럭;
```

1. 조건문(분기문)

- 예제

```
#include <stdio.h>
int main() {
		int kor,eng;
		printf("국어점수를 입력하시오(0-100):");
		scanf("%d",&kor);
		printf("영어점수를 입력하시오(0-100):");
		scanf("%d",&eng);
		if (kor<eng)
				printf("영어점수가더높습니다.\n");
		else
		{
				if (kor>eng)
						printf("국어점수가 더 높습니다.\n");
				else
						printf("국어점수와 영어점수가 같습니다.\n");
		}
		if (((kor+eng)/2)>= 60)
				printf("당신은 평균이 60점 이상이므로 합격입니다.\n");
		else
				printf("당신은 평균이 60점이 안되므로 불합격입니다.\n");

		return 0;
}
```

1. 조건문(분기문)

- 다중 if-else문: 조건식이 여러 개 가 있는 경우에 사용하는 제어문
- else 안에서 다른 조건을 비교하는 경우에 사용
- 다중 iff – else 문의 사용형식.

```
if(조건식1)
    블럭 1;
else if(조건식2)
    블럭 2;
else
    블럭 n + 1;
다음 블럭;
```

- 다중 if-else문의 경우 조건식1이 만족되면 블럭 1을 수행하고 조건식1을 만족하지 않고 조건식2를 만족시킬 경우 블럭 2를 수행하게 됩니다.
- 이와 같은 과정을 반복하여 조건식n을 만족하는 경우 블럭 n을 수행하고 모든 조건을 만족하지 않으면 블럭 n + 1을 수행하고 다음 블럭을 수행하게 됩니다.
- else if의 개수는 제한이 없으면 모든 조건을 만족하지 않으면 else 내부의 블록을 수행하게 되는데 else는 생략이 가능합니다.

1. 조건문(분기문)

- 예제

```
#include <stdio.h>
int main()
{
   int jumsu ;
   printf("점수를입력하세요\n");
   scanf("%d",&jumsu);
   if (jumsu >=90)
                printf("당신의 학점은 A입니다\n");
   else if (jumsu >=80)
                printf("당신의 학점은 B입니다\n");
   else if (jumsu >=70 )
                printf("당신의 학점은 C입니다\n");
   else if (jumsu >=60)
                printf("당신의 학점은 D입니다\n");
   else
      printf("당신의 학점은 F입니다\n");
   return 0;
}
```

1. 조건문(분기문)

- switch: 다중 if-else와 비슷하지만 여러 개의 조건 중에서 특정한 값을 선택하기 쉽게 한 제어문
- switch문에 사용된 내용은 정수형이거나 문자형(문자도 정수)
- switch문의 일반적인 형식
- switch 문의 사용형식

```
switch(수식 또는 변수)
{
    case 값 1:
        문장 1;
        break;
    case 값 2:
        문장 2;
        break;
    case 값 n:
        문장 n;
        break;
    default:
        문장 n + 1;
        break;
}
```

1. 조건문(분기문)

- switch 문은 수식 또는 변수의 값을 평가하고 그 결과값에 따라 case 레이블 문을 하나 하나 차례로 비교해 나가면서 수식의 결과값과 case 레이블이 일치하는 레이블의 문장을 수행하게 됩니다.
- default는 위에서 일치하는 값이 하나도 없을 때 수행하는 문장을 기술합니다.
- case 레이블 문에 break문이 있을 때는 switch – case 의 블록을 빠져 나가 제어가 끝나게 되고 만일 break문이 없을 경우에는 수식의 결과값과 case 레이블이 일치하는 레이블의 문장을 수행하고 계속 그 다음의 case 레이블을 수행하며 break를 만나면 종료

1. 조건문(분기문)

- 예제

```
#include <stdio.h>
int main()
{
   int a, b;
   char operand;
   a = 15;  b = 5;
   printf("사칙 연산자 중 아무거나 입력하세요:");
   scanf("%c", &operand);
   switch (operand)
   {
                case '+':
                printf("a + b = %d 입니다", a + b);
                break;
                case '-':
                printf("a - b = %d 입니다", a - b);
                break;
                case '*':
                printf("a * b = %d 입니다", a * b);
                break;
```

1. 조건문(분기문)

- 예제

```
            case '/':
               printf("a / b = %d 입니다", a / b);
               break;
            default:
               printf("잘못된연산자를입력하셨습니다\n");
   }
   return 0;
}
```

```
사칙연산자중 아무거나 입력하세요: -
a - b = 10입니다.
```

위의 프로그램은 a 와 b의 두 변수에 15와 5를 넣어서 그 값의 사칙연산을 수행하는 프로그램입니다.
키보드로부터 입력되는 연산자를 operand에 넣고 operand를 switch - case문으로 비교하여
각각의 연산을 수행하여 결과를 출력하는 프로그램입니다.
만일 위의 case에 없는 문자를 입력하게 되면 default 문이 수행되게 됩니다.

2. 반복문(Loop, Iteration)

- 프로그램을 작성하다 보면 어떤 일련의 명령들을 반복 수행하게 되는 경우가 있는데 매번 반복 명령들을 작성해야 한다면 프로그램이 매우 길어지고 복잡하게 될 것입니다.
- 이런 문제들을 해결하기 위해 사용하는 명령문들을 반복문이라고 하며 이러한 반복문에는 for, while, do, do-while 이 있습니다.
- for문
  - ✓ for 문의 사용형식
    - ○ 반복문장이 하나인 경우

```
for(처음 한 번만 수행되는 식; 수행여부판별식; 두번째 부터 수행할 식)
    반복문장;
```

    - ○ 반복문장이 두개 이상인 경우

```
for(처음 한 번만 수행되는 식; 수행여부판별식; 두번째 부터 수행할 식)
{
    반복문장1;
    반복문장2;

    반복문장n;
}
```

2. 반복문(Loop, Iteration)

- 처음 한번만 수행할 식은은 반복문을 시작하기 전에 제어변수를 초기화 하기 위한 용도로 주로 사용되며 조건판별식은 반복문을 종료할 것인지 계속할 것인지를 판정하게 되는 조건을 의미하며 두번째 부터 수행되는 식은 제어변수를 증가시키거나 감소하는 형태로 주로 작성
- 위 식에서 초기식이나 조건식 또는 증감식은 생략해도 되지만 괄호는 생략해서는 안됩니다.
- 반복문장이 하나가 아니라 여러 개 인 경우 { } 안에 기재하면 됩니다.
- for 문 안에 다른 for 문이 존재할 수 도 있는데 이를 다중 for 문이라고 합니다.
- 처음 한번만 수행되는 내용이나 두번째부터 수행되는 식은 2개 이상의 문장을 작성할 수 있는데 이 경우에는 ,로 구분해서 작성

2. 반복문(Loop, Iteration)

- 예제

```
#include <stdio.h>
int main()
{
        printf("Hello World\n");
        printf("Hello World\n");
        printf("Hello World\n");
        printf("Hello World\n");
        printf("Hello World\n");
        printf("Hello World\n");
        printf("Hello World\n");
        printf("Hello World\n");
        printf("Hello World\n");
        printf("Hello World\n");
        return 0;
}
```

```
#include <stdio.h>
int main()
{
        int i;
        for(i=0; i<10; i++)
                printf("Hello World\n");
        return 0;
}
```

2. 반복문(Loop, Iteration)

- 예제

```
#include <stdio.h>
int main()
{
    int i, sum=0;
    for(i=1; i<=100; i++)
                        sum = sum + i;
    printf("1부터까지의합= %d\n", sum);
            //for 문의조건식을변경
            sum = 0;
            for(i=0; i!=101; i++)
                        sum = sum + i;
    printf("1부터까지의합= %d\n", sum);
            return 0;
}
```

```
1부터 100까지의 합 = 5050
```

2. 반복문(Loop, Iteration)

- while: for문과 비슷한 동작을 수행하는 명령으로 조건식이 참이면 문장을 반복 수행하게 됩니다.
- while 문 의 사용형식
  - 반복문장이 하나인 경우

```
while(조건식)
    반복문장;
```

  - 반복문장이 두 개 이상인 경우

```
while(조건식)
{
    반복문장1;
    반복문장2;
    ?
    ?
    반복문장n;
}
```

- while 문은 조건식이 거짓이 될 때까지 명령문들을 반복 수행하게 됩니다.

2. 반복문(Loop, Iteration)

- 예제

```
#include <stdio.h>
int main()
{
    int n=0, sum=0;
    while(sum < 100)
    sum += n++;
    printf("n = %d \tsum = %d\n", n, sum);
    return 0;
}
```

```
n = 15    sum = 105
```

2. 반복문(Loop, Iteration)

- do – while: do-while문은 while문과 비슷하지만 while문처럼 조건식 판정을 반복문장이 시작하기 전에 판정하는 것이 아니라 반복 문장들이 다 끝난 후 조건식을 판정합니다.
- 따라서 while을 사용하는 경우 한번도 반복을 하지 않을 수 도 있지만 do-while을 사용하게 되면 일단 한번은 수행을 하게 됩니다.
- do - while 문 의 사용형식

```
do
{
    반복문장1;
    ?
    반복문장n;
} while(조건식)
```

2. 반복문(Loop, Iteration)

- 예제

```
#include <stdio.h>
int main()
{
    int n=0, sum=0;
    do
   {
        sum += n++;
    }
    while(sum <100);
    printf("n = %d sum = %d\n", n, sum);
}
```

```
n = 15    sum = 105
```

3. 기타 제어문

- break문: break문은 반복문이나 switch문에서 사용되어 프로그램 수행 중에 break문이 소속되어 있는 영역을 빠져 나오게 하는 역할을 하는 제어문
- continue문: continue문은 for, while과 같이 반복문 내에서 사용하며 continue문은 continue문 다음에 오는 부분은 무시하고 다음 반복을 새로 시작하게 하는 제어문
- goto문: goto문은 프로그램 수행 중에 goto문을 만나면 무조건 goto문이 지시하는 장소로 분기하게 하는 제어문
  - goto문의 사용형식

```
goto 이동할 곳;
이동할 곳:

이동할 곳:
goto 이동할 곳;
```

3. 기타 제어문

- 예제(1+2+. +n 의 합이 50 미만이 되는 최대의 n 값 찾기)

```
#include <stdio.h>
int main()
{
    int i=0, sum=0;
    while(1)
    {
        sum += ++i;
        printf("i = %d sum = %d\n", i, sum);
        if(sum > 50)
        {
            sum -= i--;
            break;
        }
    }
    printf("\n합이 50이 되지않는 최대값은%d 입니다.\n", i);
                return 0;
}
```

3. 기타 제어문

- 예제(1부터 100까지의 수 중에서 3의 배수가 아닌 수의 합을 구하는 프로그램)

```
#include <stdio.h>
int main()
{
    int i, sum=0;
    for(i=1; i<=100; i++)
    {
        if(i%3 == 0)
        {
                continue;
         }
        sum += i;
    }
    printf("sum = %d\n", sum);
                return 0;
}
```

3. 기타 제어문

- 예제

```
#include <stdio.h>
int main()
{
        int i = 0;
go : printf(" %d ",i);
        i++;
        if (i <10)
        {
                goto go;
        }
        return 0;
}
```

4. 제어문의 중첩

- 제어문은 중첩해서 사용이 가능
- 안쪽 블록이 끝나면 바깥 블록으로 제어권 이동

3. 기타 제어문

- 예제

```
#include <stdio.h>
int main()
{
        int i, j;
        int k = 0;

        for(i=0; i<=4; i++)
        {
                for(j=0; j<=4; j++)
                {
                        k++;
                        printf("%3d", k);
                }
                printf("\n");
        }
        return 0;
}
```

```
  1  2  3  4  5
  6  7  8  9 10
 11 12 13 14 15
 16 17 18 19 20
 21 22 23 24 25
```

3. 기타 제어문

- 예제

```
#include <stdio.h>
int main()
{
        int i, j;
        int k = 0;

        for(i=0; i<=4; i++)
        {
                for(j=0; j<=i; j++)
                {
                        k++;

        printf("%3d", k);
                }
                printf("\n");
        }
        return 0;
}
```

1				
2	3			
4	5	6		
7	8	9	10	
11	12	13	14	15

3. 기타 제어문

- 예제

```
#include <stdio.h>
int main()
{
        int i, j;
        int k = 0;
        for(i=0; i<=4; i++)
        {
                for(j=0; j<=4-i; j++)
                {
                        k++;

        printf("%3d", k);
                }
                printf("\n");
        }
        return 0;
}
```

1	2	3	4	5
6	7	8	9	
10	11	12		
13	14			
15				

3. 기타 제어문

- 예제

```
#include <stdio.h>
int main()
{
        int i, j;
        int k = 0;
        for(i=0; i<=4; i++)
        {
                for(j=0; j<=4-i; j++)
                {
                   k++;
                   printf("%3d", k);
                }
                printf("\n");
        }
        return 0;
}
```

1	2	3	4	5
6	7	8	9	
10	11	12		
13	14			
15				

3. 기타 제어문

- 예제

```
#include <stdio.h>
int main()
{
    int i, j;
    int k = 0;
    for(i=0; i<=4; i++){
                if(i<3){
                   for(j=0; j<=i; j++){
                                k++;
                                printf("%3d", k);
                                }
                }
                else{
                  for(j=0; j<=4-i; j++) {
                                k++;
                                printf("%3d", k);
                                }
               }
              printf("\n");
    }
    return 0;
}
```

1				
2	3			
4	5	6		
7	8			
9				

4. 제어문의 중첩

- 예제(교번 처리)

```
#include <stdio.h>
int main()
{
   int num;
   int odd = 0;
   int even = 0;
   int flag = 0;
   for(num=1;num<=100;num++)
   {
                if (flag == 0)
                {
                        odd = odd + num;
                        flag = 1;
                }
                else
                {
                        even = even + num;
                        flag = 0;
                }
   }
  printf("짝수합= %d 홀수합= %d\n",odd,even);
  return 0;
}
```

5. 연습문제

- 2부터 100까지 소수의 개수와 합을 구하는 프로그램을 작성하시오.
  - ✓ 소수는 자신과 1로만 나누어지는 수를 의미
  - ✓ 2부터 자신의 절반까지 나누어서 나머지가 모두 0이 아니면 소수

- 입력받은 숫자까지의 피보나치 수열의 값을 출력해주는 코드를 작성하시오.
  - ✓ 피보나치 수열은 1,1,2,3,5,8,13...
  - ✓ 첫번째와 두번째 수열의 값은 1
  - ✓ 세번째부터는 앞의 2개 항의 합

6. 제어문

- Java에서는 goto는 예약어이지만 사용할 수 없고 반복문에 레이블을 만들어서 break 나 continue 다음에 레이블을 붙여서 레이블이 있는 곳으로 제어권을 옮길 수 있음
- Python에서는 if, while, for 그리고 break, continue가 있음
  - ✓ 제어문의 조건을 작성할 때 ( )를 하지 않음
  - ✓ 블록을 만드는 방법으로 { } 대신에 들여쓰기를 이용
  - ✓ for의 사용 형식이 다름
    - range
      - range([시작값], 종료값, [간격])를 이용해서 일정한 패턴의 수열을 만들 수 있습니다.
      - 숫자를 하나만 대입하면 종료 값만 있는 것으로 간주하고 시작값은 0 간격은 1로 설정
      - 2개의 숫자를 대입하면 시작 값과 종료 값만 있는 것으로 간주하고 간격은 1로 설정

    for 임시변수 in 데이터모임:

        반복수행할 내용

    - 데이터 모임의 내용을 임시변수에 하나씩 대입하면 들여쓰기 부분을 반복 수행
    - 임시변수에 대입되는 것이라서

    for I in range(1,101,1): 이면 i가 100에서 종료됩니다.

배열과 문자열

1. 배열(Array)

- 배열은 동일한 데이터 타입의 데이터들이 연속적으로 저장되는 자료구조입니다.
- 선언한 후 사용하며 배열을 선언할 때는 배열의 차원 및 크기가 명시되어야 합니다.
- 1차원 배열
  - ✓ 1차원 배열의 선언 형식

    자료형 배열명[배열의 크기]
  - ✓ 자료형은 배열 요소들의 자료형이며 배열을 구성하는 각 자료를 배열 요소라 합니다.
  - ✓ 배열 요소의 자료형은 char, int, double 등이며 기타 파생되는 자료형도 가능
  - ✓ [ ] 의 숫자는 배열의 크기
  - ✓ 배열의 크기에는 변수를 사용할 수 없으며 배열 요소를 접근할 때는 변수 사용 가능(생성 시 변수를 사용해도 되는 컴파일러가 있음)
  - ✓ 정수 1차원 배열의 선언

    int array[5]
  - ✓ 정수형으로 array라는 배열 이름으로 int 타입 4개를 선언
  - ✓ array는 이 배열이 시작되는 곳의 주소를 가리키고 있습니다.
  - ✓ int는 32bit 시스템에서 4byte이므로 array의 크기는 20 byte 이고 array의 시작은 array[0]의 주소입니다.
  - ✓ C언어에서는 배열의 인덱스가 0에서 시작
  - ✓ array[0] 은 array에서부터 0번째에 해당하는 값이라는 의미입니다.
  - ✓ array[1]의 의미는 array로부터 자료형 만큼 한번 이동한 곳의 값이라는 의미입니다.

1. 배열(Array)

- 1차원 배열의 초기화
  - ✓ 자료형 배열명[배열의 크기] = {배열 요소의 값 나열};
    - 선언과 동시에 초기화하는 것으로 배열의 크기를 생략 가능
    - 만일 배열의 초기화 개수가 배열의 크기보다 크면 에러를 발생시키고 배열의 크기보다 작으면 int형은 0 float형은 0.0 char 형은 NULL로 초기화 합니다.
    - 정수 배열의 경우 int nai[10] = {0,} 처럼 선언하게 되면 모든 배열의 요소가 0값으로 초기화됩니다.
  - ✓ 선언한 후 별도로 초기화

  자료형 배열명[크기];

  배열명[0] = 값1; 배열명[1] = 값2;
- C언어는 배열의 범위를 점검하지 않습니다.
- 배열을 선언할 때 그 크기를 지정하도록 되어 있으므로 컴파일러는 배열 요소의 끝 번호를 알고 있지만 크기를 점검하지 않으므로 아래와 같은 문장은 유효한 문장으로 인식됩니다.

  int ar[5];

  ar[8]=1234;
- ar의 크기가 5밖에 안 되는데 ar[8]이라는 존재하지 않는 배열 요소에 어떤 값을 쓰려고 했는데 이런 코드도 C언어에서는 에러를 발생시키지 않습니다.
- 컴파일러가 배열의 범위를 점검하지 않고 단순히 ar에서부터 4*7에 해당하는 만큼 이동한 곳의 값이라고 생각하기 때문입니다.
- sizeof(배열명)은 배열이 할당받은 전체 메모리 크기를 리턴

1. 배열(Array)

❖ 예제

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	int kor[5] = { 100,80,56,75,80 };
	int eng[5];
	int mat[5] = { 98,80,55,88 };
	int i;
	eng[0] = 88;
	eng[1] = 90;
	eng[2] = 85;
	eng[3] = 99;
	eng[4] = 78;

	printf("국어\t 영어\t 수학\n");
	printf("-----------------------\n");
	for (i = 0; i < 5; i++)
	{
		printf("%d\t %d\t %d\n", kor[i], eng[i], mat[i]);
	}
	system("pause");
	return 0;
}
```

1. 배열(Array)

```
국어    영어    수학
------------------------
100     88      98
80      90      80
56      85      55
75      99      88
80      78      0
```

- 위의 프로그램에서 kor 배열은 선언과 동시에 초기화 한 경우이며 eng 배열은 선언을 하고 난 후 초기화 한 경우 입니다.
- mat 배열은 5개로 선언한 후 4개의 초기값 만 준 경우 입니다.
- 이 경우 이 배열의 자료형이 정수형이므로 초기값이 없는 데이터는 0의 값으로 초기화
- 배열을 선언과 동시에 초기화하는 경우에는 선언 시 크기를 입력하지 않아도 됩니다.
- 배열 요소의 개수는 sizeof(배열명)/sizeof(배열명[0])으로 구할 수 있습니다.

1. 배열(Array)

❖ 예제

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int a = 10;
        int ar[5] = { 0, };
        int b = 20;
        ar[7] = 100;
        ar[-3] = 111;
        printf("a:%d\n", a);
        printf("b:%d\n", b);
        system("pause");
        return 0;
}
```

```
a:100
b:111
```

1. 배열(Array)

- 다차원 배열
  - ✓ 차원이 2개 이상인 배열
  - ✓ 다차원 배열도 일차원 배열과 메모리에 구현되는 방법은 동일
  - ✓ 기억장소가 행 열 구조형태로 만들어지므로 우선순위 및 배열 요소 크기의 추가가 필요
- 이차원 배열의 형식
  - ✓ 자료형 배열명[배열의 크기] [배열의 크기]
- 이차원 배열의 선언 예
  - ✓ int array[2][3]
  - ✓ 2행 3열 크기의 2차원 배열이 만들어 지며 6개의 int 기억장소를 가지게 됩니다.
  - ✓ array[0][0] array[0][1] array[0][2] array[1][0] array[1][1] array[1][2]
  - ✓ C언어에서는 행 우선 순위로 배열을 제어합니다.
- 이차원 배열의 초기화
  - ✓ 자료형 배열명[배열의 크기] [배열의 크기] = {값1, 값2, ...}
  - ✓ 자료형 배열명[크기][크기] = {{값1, 값2, ...} {값3, 값4, ...} }
- 다차원 배열의 경우 선언하면서 초기화하는 경우 맨 앞쪽의 크기는 생략할 수 있지만 그 뒤의 크기는 생략할 수 없습니다.

1. 배열(Array)

❖ 예제

```
#include<stdlib.h>
#include<stdio.h>
int main() {
        int score[5][3] = { { 85,  60,  70},        { 90,  95,  80}, {75,  80, 100}, { 80,
70,  95},{100,  65,  80}, };
        int t_kor = 0, t_eng = 0, t_mat = 0;
        int sum[5] = { 0, };
        double avg[5];
        int  i, j;
        for (i = 0; i < 5; i++) {
                t_kor += score[i][0];
                t_eng += score[i][1];
                t_mat += score[i][2];
        }
        printf("\n국어합계\t영어합계\t수학합계");
        printf("\n%3d\t\t%3d\t\t%3d\n", t_kor, t_eng, t_mat);
        for (i = 0; i < 5; i++) {
                for (j = 0; j < 3; j++) {
                        sum[i] = sum[i] + score[i][j];
                        avg[i] = (double)sum[i] / 3;
                }
        }
```

1. 배열(Array)

❖ 예제

```
        for (i = 0; i < 5; i++)
                printf("\n%2d번 학생 합계=> %d   평균=> %5.2f", i + 1, sum[i], avg[i]);
        printf("\n");
        system("pause");
        return 0;
}
```

```
국어합계      영어합계       수학합계
 430          370           425

 1번학생합계=> 215  평균=> 71.67
 2번학생합계=> 265  평균=> 88.33
 3번학생합계=> 255  평균=> 85.00
 4번학생합계=> 245  평균=> 81.67
 5번학생합계=> 245  평균=> 81.67
```

1. 배열(Array)

❖ 예제(5*5 마방진)

마방진은 가로, 세로, 대각선의 합이 항상 같은 행렬입니다.

만드는 원리
1.첫 째줄 가운데 자리가 첫 번째 숫자가 대입
2.오른쪽 대각선 방향으로 **1**씩 증가하면서 대입되며 시작하는 숫자를 만나면 행 번호를 증가시켜서 대입
3.열 번호는 **1**씩 증가하고 열의 끝을 만나면 시작으로 이동
4.행 번호는 **1**씩 감소하고 행의 시작을 만나면 끝으로 이동

1. 배열(Array)

❖ 예제(5*5 마방진)

```
#include<stdlib.h>
#include <stdio.h>
int main() {
        int ma[5][5];
        int i = 0, j = 2, k = 1;
        for (k = 1; k <= 25; k++) {
                ma[i][j] = k;
                if (k % 5 == 0)
                        i = i + 1;
                else {
                        i = i - 1;
                        j = j + 1;
                        if (i < 0)
                                i = 4;
                        if (j > 4)
                                j = 0;
                }
        }
        for (i = 0; i < 5; i++) {
                for (j = 0; j < 5; j++)
                        printf("%3d", ma[i][j]);
                printf("\n");
        }
        system("pause");
        return 0;
}
```

1. 배열(Array)

- 문자 배열
  - ✓ 문자 배열은 문자의 집합으로 하나의 문자열이 될 수 있습니다.
  - ✓ 문자 배열의 초기화는 2가지 방법이 있습니다.
  - ✓ 문자 단위로 초기화
    - char 배열명[크기] = {'문자' .....};
    - 크기만큼 문자 삽입이 가능합니다.
    - 크기만큼 문자를 삽입한 경우 이 때 %c를 이용해서 문자 단위로 출력은 가능하지만 %s로는 정확한 출력 결과를 예측할 수 없습니다.
    - %s는 문자열을 한 묶음으로 출력하는 것이 아니고 NULL을 만날 때까지 출력하므로 NULL이 포함되지 않은 상태로 저장된 데이터를 출력할 때는 결과를 예측할 수 없습니다.

1. 배열(Array)

❖ 예제

```
#include<stdlib.h>
#include<stdio.h>
int main()
{
	char ch[5] = { 'H', 'e', 'l', 'l', 'o' };
	int i;
	printf("문자별로출력\n");
	for (i = 0; i < 5; i++)
	{
		printf("%c", ch[i]);
	}
	printf("\n");

	printf("문자열로출력\n");
	printf("%s\n", ch);
	system("pause");
	return 0;
}
```

```
문자별로출력
Hello
문자열로출력
Hello徼徼徼??몘?
```

1. 배열(Array)

- 문자열 단위로 초기화
  - ✓ char 배열명[크기] = "문자열";
  - ✓ 문자열 단위로 초기화 할 때는 위와 같이 할 수 있습니다.
  - ✓ 역시 마찬가지로 문자열의 길이는 크기만큼 될 수 있지만 이렇게 하면 %s를 이용해서는 출력을 하면 안됩니다.
  - ✓ 만일 문자열의 길이를 알 수 없는 경우 크기를 지정하지 않으면 문자열의 사이즈보다 하나 더 많은 공간을 배열로 설정해서 NULL까지 저장해주므로 그렇게 하는 것이 좋습니다.
  - ✓ 문자열을 직접 대입하는 것은 생성 시 한번만 가능합니다.
  - ✓ 이후에는 문자단위로 초기화 하는 것만 가능합니다.

1. 배열(Array)

- 2차원 문자 배열
  - ✓ 각 행이 하나의 문자열을 저장
  - ✓ char str[5][10]이라고 선언하면 9글자 짜리 문자열을 5개 저장할 수 있게 됩니다.
  - ✓ 2차원 문자 배열을 초기화하는 경우에는 행 단위로 초기화하는 것이 편리합니다.
  - ✓ apple, mellon, banana 3 개의 과일이름을 초기화하는 경우라면

  char fruit[3][7] = { “apple”, “mellon”, “banana”};

  로 선언한 후

```
for(i=0; i<3; i++)
{
        printf("fruit[%d] 내용 %s₩n", i,fruit[i]);
}
```

  로 출력하면 됩니다.

1. 배열(Array)

❖ 예제

```
#include<stdlib.h>
#include<stdio.h>
int main()
{
	char ch[] = "Hello";
	int i;
	printf("문자 별로 출력\n");
	for (i = 0; i < 5; i++)
	{
		printf("%c", ch[i]);
	}
	printf("\n");

	printf("문자열로 출력\n");
	printf("%s\n", ch);
	system("pause");
	return 0;
}
```

```
문자별로출력
Hello
문자열로출력
Hello
```

1. 배열(Array)

❖ 예제

```
#include<stdlib.h>
#include <stdio.h>
int main()
{
	char fruit[3][7] = { "apple", "mellon", "banana" };
	int i;
	for (i = 0; i < 3; i++)
	{
		printf("fruit[%d] 내용%s\n", i, fruit[i]);
	}
	system("pause");
	return 0;
}
```

입력을 받을 때는 **fruit[행 번호]**로 입력 받으면 됩니다.
2차원 배열을 이용해서 문자열을 생성하는 경우 문자열의 길이가 일정하지 않으면 낭비가 발생합니다.
이러한 이유로 **2**차원 문자배열은 포인터 배열을 사용하는 경우가 많습니다.

2. 문자열 처리

- Java에서는 String 이라는 클래스를 이용
- Python은 문자와 문자열을 구분하지 않고 str로 처리

3. 기본 알고리즘

- 최대, 최소 알고리즘
  - ✓ 최대값을 찾을 때는 최대값을 처음 아주 작은 수나 배열의 첫 번째 값이라 하고 모든 수와 비교해서 최대값보다 큰 수를 만나면 그 수를 최대값에 대입해주면 됩니다.
  - ✓ 최소값을 찾을 때는 최소값을 처음 아주 큰 수나 배열의 첫 번째 값이라 하고 모든 수와 비교해서 최소값보다 작은 수를 만나면 그 수를 최대값에 대입해주면 됩니다.
  - ✓ 만일 최대값 또는 최소값을 가진 번호를 저장하고 싶다면 최대값 또는 최소값을 구할 때 그 번호를 다른 변수에 저장하면 됩니다.
  - ✓ 만일 문자의 최대값이나 최소값을 구한 다면 문자 변수는 ASCII 값을 가지고 있으므로 직접 비교가 가능합니다.
  - ✓ 문자열인 경우는 strcmp 함수를 이용해야 합니다.

3. 기본 알고리즘

❖ 예제(1부터 100까지 3의 배수의 합계와 개수와 평균)

```
#include<stdlib.h>
#include<stdio.h>
int main()
{
	int i;
	int sum = 0, cnt = 0;
	float avg;

	for (i = 1; i <= 100; i++)
	{
		if (i % 3 == 0)
		{
			sum = sum + i;
			cnt = cnt + 1;
		}
	}
	avg = (float)(sum / cnt);
	printf("1부터 100까지의 3의 배수의 합 = %d \n", sum);
	printf("1부터 100까지의 3의 배수의 개수 = %d \n", cnt);
	printf("1부터 100까지의 3의 배수의 평균 = %4.2f \n", avg);
	system("pause");
	return 0;
}
```

3. 기본 알고리즘

❖ 예제(5개의 정수를 입력받아서 최대값과 최소값 구하기)

```
#include<stdlib.h>
#include <stdio.h>
int main()
{
	int test[5];
	int max, min, i;
	printf("다섯개의숫자를입력하세요\n");
	for (i = 0; i < 5; i++){
		printf("숫자를입력하세요:");
		scanf_s("%d", &test[i]);
	}
	max = 0;
	min = test[0];
	for (i = 0; i < 5; i++){
		if (max < test[i])
			max = test[i];
		if (min > test[i])
			min = test[i];
	}
	printf("----------------------\n");
	printf("최대값= %d \n", max);
	printf("최소값= %d \n", min);
	system("pause");
	return 0;
}
```

3. 기본 알고리즘

❖ 예제(5개의 정수를 입력받아서 최대값과 최소값 및 인덱스 구하기)

```
#include<stdlib.h>
#include <stdio.h>
int main()
{
        int test[5];
        int max, min, i;
        int maxnum, minnum;
        printf("다섯개의숫자를입력하세요\n");
        for (i = 0; i < 5; i++)
        {
                printf("숫자를입력하세요:");
                scanf_s("%d", &test[i]);
        }
```

3. 기본 알고리즘

❖ 예제(5개의 정수를 입력받아서 최대값과 최소값 및 인덱스 구하기)

```
        max = 0;
        min = test[0];
        for (i = 0; i < 5; i++)
        {
                if (max < test[i])
                {
                        max = test[i];
                        maxnum = i + 1;
                }
                if (min > test[i])
                {
                        min = test[i];
                        minnum = i + 1;
                }
        }
        printf("---------------------\n");
        printf("%d번째숫자가최대값= %d \n", maxnum, max);
        printf("%d번째숫자가최소값= %d \n", minnum, min);
        system("pause");
        return 0;
}
```

3. 기본 알고리즘

- 정렬은 데이터를 순서대로 나열하는 것
- 선택 정렬
  - ✓ 정렬방식은 작은 것부터 큰 순서대로 나열하는 오름차순 정렬과 큰 것에서 작은 순서대로 나열하는 내림차순 정렬이 있고 정렬을 하는 방법은 여러 가지가 있습니다.
  - ✓ 선택정렬은 첫 번째 자리부터 마지막에서 두 번째 자리까지 자신보다 뒤에 있는 모든 자리들과 비교해서 다음 자료가 작으면 2개의 요소의 자리를 변경해주면 됩니다.
  - ✓ ex)초기 상태 50 40 10 20 30
  - ✓ 1Pass 10 50 40 20 30
  - ✓ 2Pass 10 20 50 40 30
  - ✓ 3Pass 10 20 30 50 40
  - ✓ 4Pass 10 20 30 40 50
    - 1번째 자리를 기준으로 2,3,4,5 번째 자리와 비교
    - 1번째 자리는 제외하고2번째 자리를 기준으로 3,4,5 번째 자리와 비교
    - 3번째 자리를 기준으로 4,5 번째 자리와 비교
    - 4번째 자리를 기준으로 5번째 자리와 비교

3. 기본 알고리즘

- 2개의 요소 자리 변경(SWAP)

  a 와 b 의 swap

  temp = a;

  a = b;

  b = temp;

3. 기본 알고리즘

❖ 예제(5개의 정수를 입력받아서 선택 정렬을 이용해서 오름차순 정렬)

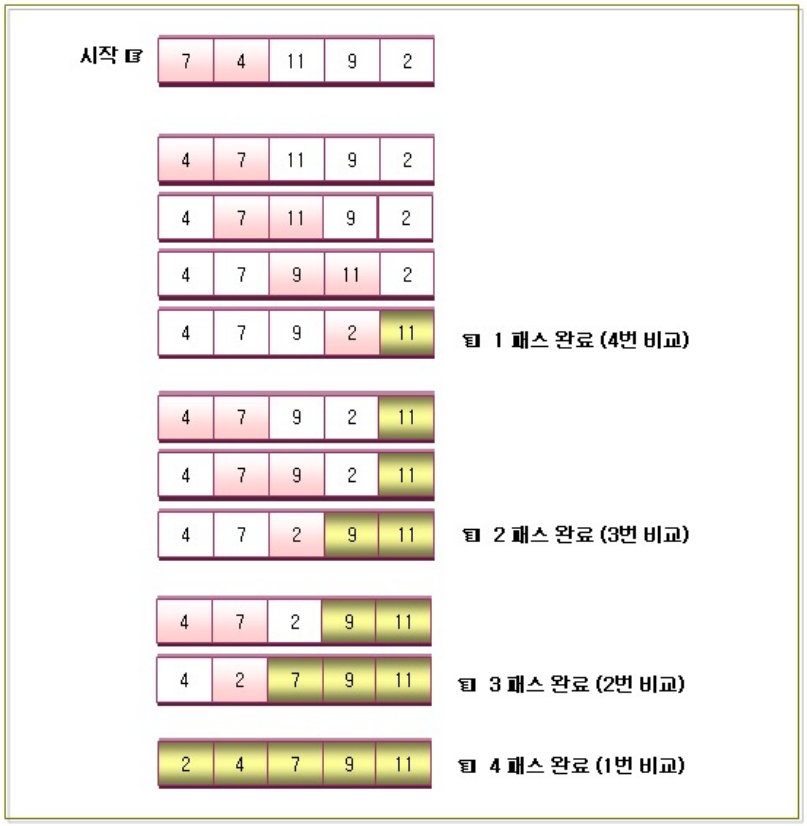
```
#include <stdlib.h>
#include <stdio.h>
int main()
{
	int test[5];
	int temp, i, j;
	printf("다섯 개의 숫자를 입력하세요\n");
	for (i = 0; i < 5; i++)
	{
		printf("숫자를 입력하세요:");
		scanf_s("%d", &test[i]);
	}
	printf("정렬 전 출력\n");
	for (i = 0; i < 5; i++)
	{
		printf("%d\t", test[i]);
	}
	printf("\n");
```

3. 기본 알고리즘

```
//선택정렬
        for (i = 0; i < 4; i++)
        {
                for (j = i + 1; j < 5; j++)
                {
                        if (test[i] > test[j])
                        {
                                temp = test[i];
                                test[i] = test[j];
                                test[j] = temp;
                        }
                }
        }
        printf("정렬 후 출력\n");
        for (i = 0; i < 5; i++)
                printf("%d\t", test[i]);
        printf("\n");
        printf("----------------------\n");
        system("pause");
        return 0;
}
```

3. 기본 알고리즘

- 버블 정렬은 n개의 데이터가 있을 때 1부터 n-1 번째 자료까지 n-1번 동안 다음 자료와 비교해가면서 정렬하는 방법입니다.
- 버블 정렬의 효과를 높이기 위해서는 비교 시 횟수만큼 빼면서 정렬하면 성능을 높일 수 있고 flag처리 등을 이용해서 자리 바꿈이 일어나지 않을 때 멈추게 하면 성능을 더욱 향상 시킬 수 있습니다.



3. 기본 알고리즘

❖ 예제(5개의 정수를 입력받아서 버블 정렬을 이용해서 오름차순 정렬)

```
#include <stdio.h>
#include <stdilib.h>
int main()
{
	int test[5];
	int temp, i, j;
	int cnt = 0;
	int flag;
	printf("다섯개의 숫자를 입력하세요\n");

	for(i=0;i<5;i++)
	{
		printf("숫자를 입력하세요:");
		scanf_s("%d",&test[i]);
	}

	printf("정렬 전 출력 \n");
	for(i=0;i<5;i++)
	{
		printf("%d\t", test[i]);
	}
	printf("\n");
	printf("----------------------\n");
```

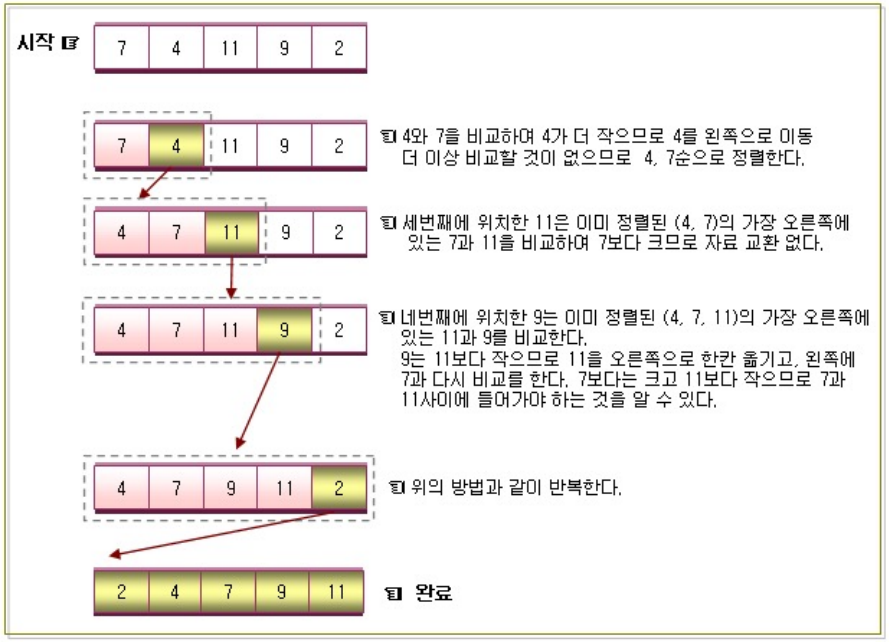
3. 기본 알고리즘

❖ 예제(5개의 정수를 입력받아서 버블 정렬을 이용해서 오름차순 정렬)

```
        for(i=0;i<4;i++)
        {
                flag = 0;
                for(j=0;j<4;j++)
                {
                        cnt=cnt+1;
                        if(test[j] > test[j+1])
                        {
                                temp = test[j];
                                test[j] = test[j+1];
                                test[j+1] = temp;
                                flag = 1;
                        }
                }
                if (flag == 0) break;
        }
        printf("정렬 후 출력 \n");
        for (i=0;i<5;i++)
        {
                printf("%d\t",test[i]);
        }
                printf("\n");
                printf("총 비교횟수 = %d번\n", cnt);
        system("pause");
    return 0;
}
```

3. 기본 알고리즘

- 삽입 정렬: 가장 왼쪽에 있는 첫 번째 데이터를 이미 정렬된 상태로 가정하고 나머지 자료들을 정렬하는 방법입니다.
- 두 번째 값을 기준으로 첫 번째 값을 비교하여 값에 따라 순서대로 나열하며 세 번째 값을 기준으로 두 번째 값과 첫 번째 값을 비교하여 값에 따라 순서대로 나열합니다.
- 위와 같은 방법으로 n - 1개의 값과 비교하여 삽입될 적당한 위치를 찾아 삽입합니다.
- 적은 비교와 많은 교환이 필요한 방법이므로 소량의 자료를 처리하는데 가장 유리한다.



3. 기본 알고리즘

❖ 예제(5개의 정수를 입력받아서 버블 정렬을 이용해서 오름차순 정렬)

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        int data[]={7,4,11,9,2};
        int i,j;
        int temp;
        printf("정렬전의데이터\n");
        for(i=0; i<5; i++)
                printf("%d\t", data[i]);
        printf("\n");

        for (i=1; i<sizeof(data)/sizeof(data[0]); i++) {
                temp = data[i];
                for (j=i; j>0; j--) {
                        if (data[j-1] > temp) {
                                data[j] = data[j-1];
                        }
                        else {
                                break;
                        }
                }
                data[j]=temp;
        }
```

3. 기본 알고리즘

❖ 예제(5개의 정수를 입력받아서 버블 정렬을 이용해서 오름차순 정렬)

```
        printf("정렬후의데이터\n");
        for(i=0; i<5; i++)
                printf("%d\t", data[i]);
        printf("\n");
        system("pause");
        return 0;
}
```

3. 기본 알고리즘

❖ Merge Sort: 2개의 정렬된 데이터를 비교해서 하나의 정렬된 데이터를 만드는 정렬

```
#include <stdlib.h>
#include <stdio.h>
int main()
{
        int data1[] = { 2,4,6,8,10 };
        int data2[] = { 1,3,5,7,9 };
        int data3[10];
        int index1, index2;
        int temp1, temp2;
        int sw = 0;
        int i;
        index1 = 0;
        index2 = 0;
        temp1 = data1[index1];
        temp2 = data2[index2];
```

3. 기본 알고리즘

```
for (i = 0; i < 10; i++) {
                    if (temp1 < temp2) {
                                data3[i] = temp1;
                                index1 = index1 + 1;
                                temp1 = data1[index1];
                                if (index1 == 5 && sw == 0) {
                                            sw = 1;
                                            temp1 = 99999999;
                                }
                                else if (index1 == 5 && sw == 1) {
                                            break;
                                }
                    }
                    else {
                                data3[i] = data2[index2];
                                index2 = index2 + 1;
                                temp2 = data2[index2];
                                if (index2 == 5 && sw == 0) {
                                            sw = 1;
                                            temp2 = 99999999;
                                }
                                else if (index2 == 5 && sw == 1) {
                                            break;
                                }
                    }
          }
```

3. 기본 알고리즘

```
printf("정렬후의데이터\n");
        for (i = 0; i < 10; i++)
                printf("%4d", data3[i]);
        printf("\n");
        system("pause");
        return 0;
}
```

3. 기본 알고리즘

- 순위 알고리즘: 순위를 구할 때는 모든 사람의 점수가 1등이라 하고 자신을 포함한 모든 사람들과 비교하여 자신의 점수보다 높은 점수를 만나면 순위를 1씩 증가 시켜 주면 됩니다.

```
#include <stdio.h>
#include <stdio.h>
int main()
{
        int test[5];
        int rank[5], i, j;
        printf("다섯개의점수를입력하세요\n");
        for(i=0;i<5;i++)
        {
                printf("점수를입력하세요:");
                scanf_s("%d",&test[i]);
        }
```

3. 기본 알고리즘

```
for(i=0;i<5;i++)
        {
                rank[i] = 1;
                for(j=0;j<5;j++)
                {
                        if(test[i] < test[j])
                        {
                                rank[i]++;
                        }
                }
        }

printf("번호\t 점수\t 순위\n");
        for (i=0;i<5;i++)
        {
                printf("%d\t %d\t %d등\n",i+1,test[i], rank[i]);
        }
    system("pause");
    return 0;
}
```

3. 기본 알고리즘

- 순차 검색
  - ✓ 순차 검색은 데이터가 정렬되어 있지 않은 경우 첫 번째 데이터부터 마지막 데이터까지 모두 검색해서 찾는 방법입니다.
  - ✓ 우리가 일반적으로 정리되지 않은 서랍에서 물건을 찾는 경우와 동일합니다.
  - ✓ 평균 비교 횟수가 데이터가 있을 확률을 0.5라고 한다면
  - ✓ 0.5 * n + 0.5 * n / 2 입니다.
  - ✓ 데이터가 많은 경우 매우 비 효율적인 방법이 됩니다.

3. 기본 알고리즘

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int ar[]={23,47,19,63,57,26,75,73,82,89,47,11};
    int i,num;
    int key,index=0;
    num=sizeof(ar)/sizeof(ar[0]);
    printf("찾고자하는값을2자리로입력하세요:");
            scanf("%d",&key);
            for (i=0;i<num;i++) {
        if (ar[i] == key) {
                                index = i+1;
        }
    }

            if (index == 0)
            {
        printf("찾는값이없습니다.\n");
    }

            else
            {
        printf("찾는값은%d번째에있습니다.\n",index);
    }
    system("pause");
    return 0;
}
```

3. 기본 알고리즘

- 이분 검색
  - ✓ 이분 검색은 데이터가 정렬되어 있는 경우 중앙 값과 비교해서 작으면 왼쪽으로 크면 오른쪽으로 이동해서 검색하는 방법입니다.

  - ✓ low = 0, high = n(데이터의 개수)-1로 초기화
  - ✓ 무한 반복 문에서

  low > high 이면 데이터가 없는 것이므로 break;

  low > high 가 아니면 middle = (low + high)/2를 해서 middle값을 찾는다.

  검색 값과 data[middle]과 비교해서 같으면 찾은 것이므로 출력하고 break;

  검색 값이 중앙값보다 크다면 low = middle +1

  작다면 high = middle -1을 해서 loop를 돌리면 됩니다.

3. 기본 알고리즘

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int data[10]={11, 16, 21, 26, 35, 39, 47, 52, 57, 72};
        int k, cnt =0;
        int low = 0;
        int high = 9;
        int middle;
   printf("검색값: ");
   scanf("%d", &k);
        while(1)
        {
                if(low >= high)
                {
                        printf("검색데이터가없습니다\n");
                        break;
                }
            middle=(low+high)/2;
                cnt++;
      if(data[middle]==k)
                {
                        printf("%d번째: %d 검색횟수= %d번\n", middle+1,
data[middle],cnt);
                        break;
      }
```

3. 기본 알고리즘

```
                if(k>data[middle])
                {
                        low = middle +1;
                }
                else
                {
                        high = middle -1;
                }
        }
    system("pause");
    return 0;
}
```

4. 연습문제

❖ 다섯 개의 숫자를 입력 받아서 7에 가장 가까운 값을 찾는 프로그램을 작성하시오.

❖ 아래와 같은 숫자를 저장하는 이차원 배열을 만들고 출력하시오.

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

포인터

1. 포인터

- 모든 변수는 메모리 내에 저장장소를 할당 받아서 사용합니다.
- 모든 변수의 저장과 참조는 변수의 값이 저장된 참조(reference - address)를 알아야 가능
- 변수자체를 참조할 때 그 변수가 저장된 메모리의 주소는 알 필요가 없었지만 실제로 컴퓨터는 변수를 참조할 때 그 변수가 저장되어 있는 주소를 먼저 찾아내고 그 다음 그 주소가 가리키는 내용을 참조하게 됩니다.
- 메모리의 주소를 저장하거나 사용하기 위한 변수가 포인터입니다.
- 포인터(pointer)란 주소를 관리하기 위한 자료형입니다.
- 포인터는 실제 값을 사용하는 것이 어렵거나 바람직하지 못한 경우 사용됩니다.
- 포인터를 사용하는 이유
  - ✓ 동적으로 메모리를 관리하기 위해서
  - ✓ 1MB 이상의 연속된 저장 공간을 사용하고자 하는 경우
  - ✓ 배열이나 문자열을 편리하게 사용하기 위하여
  - ✓ 복잡한 자료 구조를 효율적으로 처리하기 위해
  - ✓ 호출하는 함수의 매개 변수의 값을 호출당하는 함수에서 변경하기 위하여

1. 포인터

❖ 예제

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	int a = 5;
	int *ap;
	ap = &a;
	printf("a의 값:%d\n", a);
	printf("a의 주소:%p\n", &a);
	//a는일반 변수이므로 *을 사용하게 되면 참조할 수 있는 자리가 없기 때문에
	//에러가 발생하게 됩니다.
	//printf("a가 가리키는 곳의 값:%d\n",*a);
	printf("ap의 값:%p\n", ap);
	printf("ap의 실제주소:%p\n", &ap);
	printf("ap가 가리키는 곳의 값:%d\n", *ap);
	*ap = *ap + 5;
	printf("a의 값:%d\n", a);
	printf("ap가 가리키는 곳의 값:%d\n", *ap);
	system("pause");
	return 0;
}
```

1. 포인터

❖ 예제

정수형 변수 **a**는 **5**로 초기화되고 정수형 포인터 변수 **ap**는 **a**의 시작 주소가 대입됩니다.
따라서 주소 **ap**가 가리키는 내용인 ***ap**는 **5**가 될 것입니다.
그 다음 문장인 ***ap = *ap + 5;** 는 **ap**가 가리키는 주소에 있는 값에 **5**를 더하라는 것이므로 결과적으로 **a**에 **5**를 더해주는 결과를 가져오게 됩니다.
출력되는 주소는 사용하는 컴퓨터 기종이나 컴파일러에 따라 다를 수 있습니다.
주소를 출력할 때 **%d**를 사용하여 출력하면 **10**진수로 **%p**를 사용하여 출력하면 **16**진수로 출력됩니다.
*의 의미는 메모리 참조를 한 번 더 하라는 의미입니다.

2. 포인터 변수

- 메모리 상의 위치 값을 담아둘 용도의 변수를 포인터 변수라 부르며 주소 변수라고도 합니다.
- 포인터 변수의 선언형식
  - ✓ 자료형 *포인터변수명;
  - ✓ 포인터 변수 선언이나 포인터 변수의 값을 가져올 때 사용되는 연산자(' * ')는 단항 연산자이며 포인터 변수를 선언할 때 사용할 수 있고 선언된 이후에 사용되면 변수가 가리키는 장소에 저장된 내용을 의미합니다.
  - ✓ 포인터 변수 자체의 자료형은 항상 unsigned int이며 포인터 변수 선언 시에 사용한 자료형은 포인터 변수가 가리켜야 할 대상의 자료형
  - ✓ 자료형 *변수명 으로 선언해도 되고 자료형* 변수명 으로 선언해도 됩니다.
- 포인터 변수에 변수의 주소 대입
  - ✓ 포인터변수 = &변수명
  - ✓ 포인터변수 = 배열명 또는 주소변수(포인터변수)
- 배열 이름은 &연산자를 붙이지 않아도 됩니다.
- 포인터변수에 다른 변수의 시작 주소 값을 저장할 경우에는 그 변수의 자료형과 포인터 변수가 가리켜야 하는 자료형이 반드시 일치 해야 합니다.
- 포인터 변수가 가리켜야 하는 자료형이 다르면 읽는 방법과 읽어야 하는 메모리 크기가 다르기 때문입니다.

2. 포인터 변수

❖ 예제

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	char array[6] = "Hello";
	char *ap;
	ap = array;
	printf("%c\n", *ap);
	//printf("%s\n", *ap);
	//printf("%c\n", ap);
	printf("%s\n", ap);
	system("pause");
	return 0;
}
```

위의 프로그램에서 ***ap**는 **array** 배열의 시작 주소의 값 즉 **array[0]**을 가리키므로 하나의 문자만을 출력할 수 있습니다.
이 때는 **%s**를 사용해서 출력할 수 없습니다.
%s는 주소가 반드시 와야 하기 때문입니다.
%s는 매칭이 되는 주소부터 **NULL**을 만날 때까지 출력합니다.

2. 포인터 변수

- 포인터 변수의 연산
  - ✓ 포인터 변수도 숫자를 저장하고 있으므로 산술연산이 가능
  - ✓ 포인터 변수끼리 덧셈은 할 수 없습니다.
  - ✓ 뺄셈은 가능한데 포인터 변수 사이의 뺄셈은 주소의 차이가 얼마인지 알아낼 수 있습니다.
  - ✓ 이 때 주의할 점은 주소끼리 뺄셈한 결과를 리턴하는 것이 아니고 자신이 가리키는 자료형의 크기 개수를 리턴해줍니다.
  - ✓ 포인터에 정수를 더하거나 뺄 수 있습니다.
  - ✓ 증감연산자인 ++, --도 사용 가능합니다.
  - ✓ 자료형 * ptr 일 때 ptr + I 는 ptr + (i*sizeof(자료형)) 이 됩니다.
  - ✓ 포인터 변수끼리 대입할 수 있습니다.
  - ✓ 정수형 포인터 p1, p2가 있을 때 p1=p2 대입식으로 p2가 기억하고 있는 번지를 p1에 대입할 수 있습니다.
  - ✓ 만약 두 포인터의 자료형이 다를 경우는 캐스트 연산자로 자료형을 강제로 맞추어야 합니다.
  - ✓ 이 경우 결과에 대해서는 프로그래머가 책임져야 합니다.
  - ✓ 포인터 변수와 실수와의 연산은 허용되지 않습니다.
  - ✓ 포인터 변수에 곱셈이나 나눗셈을 할 수 없습니다.
  - ✓ 포인터 변수끼리 비교는 가능합니다.

2. 포인터 변수

❖ 예제

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	int n1 = 100;
	int n2 = 1000;
	int *ap;
	int *bp;
	ap = &n1;
	bp = &n2;
	printf("n1의 주소: %p\n", ap);
	printf("n2의 주소: %p\n", bp);
	// 포인터 변수가 가리키는 거리를 아래 문장이 출력함
	printf("n1과 n2의 주소 차이: %d\n", ap - bp);

	//printf("n1과 n2의 주소의 합: %d\n",ap+bp);
	//포인터 변수끼리 더할수는 없습니다.

	printf("%d\n", ap > bp);
	//ap의 주소가 bp의 주소보다 큼
	system("pause");
	return 0;
}
```

2. 포인터 변수

❖ 예제

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	int n = 1234;
	int *ap1;
	double f = 3.14;
	double *ap2;
	ap1 = &n;
	ap2 = &f;
	printf("정수:%d\n", *ap1);
	printf("실수:%f\n", *ap2);
	ap1 = (int *)&f;
	printf("ap2로 읽은 f번지의 값: %f\n", *ap2);
	printf("ap1로 읽은 f번지의 값: %f\n", *ap1);
	printf("ap2로 읽은 f번지의 주소: %p\n", ap2);
	printf("ap1로 읽은 f번지의 주소: %p\n", ap1);
	system("pause");
	return 0;
}
```

2. 포인터 변수

❖ 예제

```
정수 = 1234
실수 = 3.140000
ap2로 읽은 f번지의 값 3.140000
ap1로 읽은 f번지의 값 이상한 값
ap2로 읽은 f번지의 값 1245040
ap1로 읽은 f번지의 값 1245040
```

위의 예처럼 형 변환을 이용하여 서로 다른 자료형을 지시하게 만든 포인터 변수에 다른 자료형의 주소를 대입시킬 수는 있지만 값 자체를 정확하게 대입할 수는 없습니다.
따라서 위와 같은 경우는 가급적 피하는 것이 좋습니다.

2. 배열과 포인터

- 1차원 배열에서는 배열 이름 자체가 배열이 할당받은 메모리 공간의 시작 주소를 의미합니다.
- 1차원 배열에서는 배열 이름에 1씩 증가시키면 배열의 자료형 크기 만큼씩의 주소가 증가하게 됩니다.
- 포인터 변수와 배열의 이름이 다른 점은 포인터 변수는 다른 데이터의 주소를 기억할 수 있지만 배열 변수는 처음 할당받은 공간의 주소만 기억해야 하며 스택의 공간을 할당받기 때문에 메모리 사용에 한계가 있습니다.

2. 배열과 포인터

❖ 예제

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int i;
        int ar[3] = { 1, 3, 5 };
        int *ap = ar;
        for (i = 0; i<3; i++)
        {
                printf("ar[%d] = %d  ar+%d = %p \n", i, ar[i], i, ar + i);
                printf("ap[%d] = %d  ap+%d = %p \n", i, ap[i], i, ap + i);
        }
        system("pause");
        return 0;
}
```

ar의 시작주소가 **1245036**이라면 **ar+1**의 값은 **1245037**이 되어야 할 것 같은데 **1245040**이 됩니다.
그 이유는 **ar**가 정수 형이므로 **ar + 1**은 정수 형 **1**개가 저장되는 **4**바이트를 더해야 실제 메모리의 주소가 나오게 됩니다.
만일 **short**형이었다면 **2**바이트를 더해야 했을 것이고 **char** 이었다면 **1**바이트를 더하면 되었을 것입니다.
배열이름**+1** 은 배열의 시작주소에 **1**을 더하는 것이 아니라 배열의 시작주소에 배열의 자료형만큼을 더하게 되어 배열 다음 요소를 가리키게 됩니다.
이처럼 일 차원 배열과 포인터는 거의 동일하게 사용합니다.

2. 배열과 포인터

❖ 예제

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int a[3] = { 10,20,30 };
        int *ap;
        ap = a;
        printf("*ap = %d  \n", *ap);
        *ap++;
        printf("*ap = %d  \n", *ap);

        ap = a;
        (*ap)++;
        printf("*ap = %d  \n", *ap);
        system("pause");
        return 0;
}
```

```
*ap = 10
*ap = 20
*ap = 11
```

2. 배열과 포인터

❖ 예제

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	int array1[3] = { 1,3,5 };
	int array2[3];
	int *ap;
	//array2 = array1;
	//위의 문장은 성립할 수 없음 배열의 주소는 변경 불가
	ap = array1;
	printf("ap가 가리키는 주소: %p\n", ap);
	printf("array1이 할당받은 주소: %p\n", array1);
	ap = array2;
	printf("ap가 가리키는 주소: %p\n", ap);
	printf("array2 가할당받은 주소: %p\n", array2);
	printf("array1의 진짜 주소: %p\n", &array1);
	printf("ap의 진짜 주소: %p\n", &ap);
	system("pause");
	return 0;
}
```

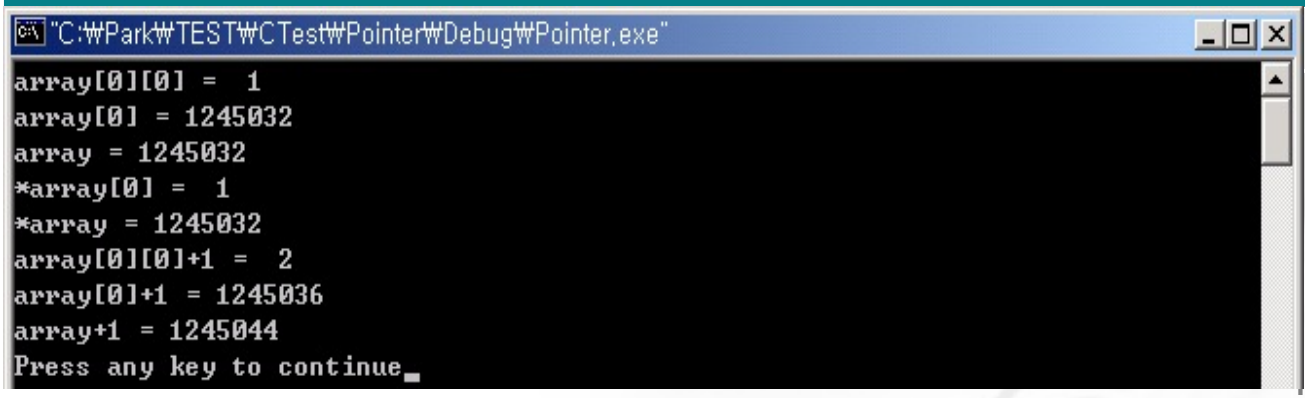
2. 배열과 포인터

- 2차원 배열과 포인터
  - ✓ 2차원 배열도 1차원 배열과 마찬가지로 연속적으로 메모리에 저장되게 됩니다.
  - ✓ 배열명[행][열]은 배열의 요소에 해당하는 값이 출력됩니다.
  - ✓ 배열명[행]은 열이 생략됐는데 열을 생략하면 1개가 아니라 그 생략된 열의 모든 구성요소를 의미하므로 첫 번째 요소의 주소가 됩니다.
  - ✓ 2차원 배열에서 배열명은 첫 번째 행의 주소가 됩니다.
  - ✓ 따라서 배열명 + 1은 하나의 열을 증가시키는 것이 아니고 하나의 행을 증가시키는 효과를 가져오게 됩니다.
  - ✓ 이처럼 다차원 배열에서는 하나의 요소번호라도 생략하면 항상 생략된 요소번호의 [0]의 주소를 가리키게 됩니다.
  - ✓ 이는 1개를 가리킬 때는 메모리를 직접 참조해도 되지만 여러 개를 가지고 있는 경우 모든 변수는 하나만 가리킬 수 있으므로 첫 번째 요소 한 개만 가리킬 수 있기 때문입니다.

2. 배열과 포인터

❖ 예제

```
#include <stdlib.h>
#include <stdio.h>
int main()
{
        int array[2][3] = { { 1,3,5 },{ 7,9,11 } };
        printf("array[0][0] = %d\n", array[0][0]);
        printf("array[0] = %p\n", array[0]);
        printf("array = %p\n", array);
        printf("*array[0] = %d\n", *array[0]);
        printf("*array = %p\n", *array);
        printf("array[0][0]+1 = %d\n", array[0][0] + 1);
        printf("array[0]+1 = %p\n", array[0] + 1);
        printf("array+1 = %p\n", array + 1);
        system("pause");
        return 0;
}
```



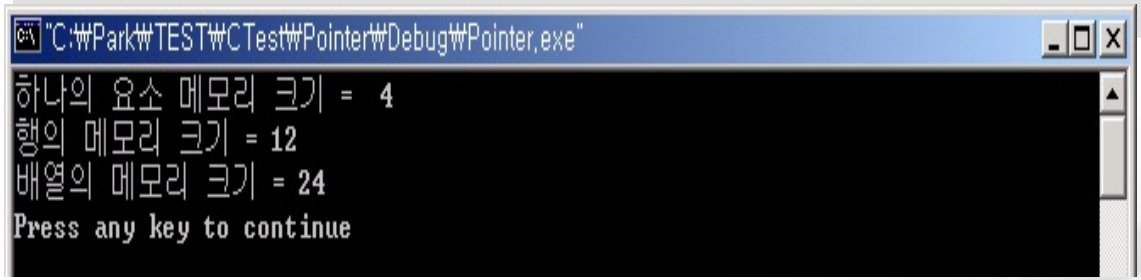
2. 배열과 포인터

- 2차원 배열에서 array[0][0] 은 첫 번째 행 첫 번째 열의 값을 의미하므로 1 이 출력됩니다.
- array[0] 은 첫 번째 행의 시작 주소를 의미하게 되므로 1234032 가 출력됩니다.
- *array[0] 은 첫 번째 행의 시작 주소에 저장된 값을 출력하게 되므로 1이 출력됩니다.
- array은 배열의 시작 주소가 되므로 1234032이 출력됩니다.
- *array은 배열의 첫 번째 시작 주소가 되므로 1234032이 출력됩니다.
- array[0][1] 은 첫 번째 행 두 번째 열의 값을 의미하므로 3이 출력됩니다.
- array[0]+1 이 경우는 첫 번째 행의 시작 주소에 하나의 요소 크기만큼을 더한 주소가 출력됩니다.
- *array[0]+1 이 경우는 *array[0] 값에 1을 더한 것이므로 2가 출력됩니다.
- array + 1 은 하나의 행 크기만큼의 주소를 더해주게 됩니다.
- *array + 1 이 경우는 하나의 요소 크기만큼의 주소를 더해주게 됩니다.
- 이차원 배열에서는 배열 명으로 요소의 값을 출력하고 싶다면 **을 해주면 됩니다.
- 2차원 배열에서 메모리 사이즈는 배열의 이름이 메모리 전체의 사이즈가 되며 행의 크기는 열 번호를 생략한 크기가 되며 하나의 요소 크기는 행과 열 요소를 모두 표현해서 크기를 구하면 됩니다.

2. 배열과 포인터

❖ 예제

```
#include <stdlib.h>
#include <stdio.h>
int main()
{
        int array[2][3] = { { 1,3,5 },{ 7,9,11 } };
        printf("하나의 요소 메모리 크기= %d\n", sizeof(array[0][0]));
        printf("행의 메모리크기= %d\n", sizeof(array[0]));
        printf("배열의 메모리크기= %d\n", sizeof(array));
        system("pause");
        return 0;
}
```



2. 배열과 포인터

❖ 예제

```
#include <stdlib.h>
#include <stdio.h>
int main()
{
	int array[2][3] = { { 1,3,5 },{ 7,9,11 } };
	printf("%2d\n", array[0][0]);
	printf("%2d\n", *array[0]);
	printf("%2d\n", **array);

	printf("%2d\n", array[0][1]);
	printf("%2d\n", *(array[0] + 1));
	printf("%2d\n", *(*array + 1));

	system("pause");
	return 0;
}
```

```
1
1
1
3
3
3
```

5. 동적 메모리 할당

- 동적 할당(Dynamic Allocation)이란 프로그램을 작성할 때(Compile Time 또는 Design Time) 메모리 필요량을 지정하는 정적 할당과는 달리 실행 중에(Run Time) 필요한 만큼 메모리를 할당하는 기법입니다.
- 동적 공간 할당 또는 동적 공간 확보라고도 하며 실행 시 메모리 공간을 해제 하는 것을 동적 메모리 해제 라고 합니다.
- 동적 메모리 할당은 메모리를 효율적으로 사용하기 위해서 사용합니다.
- 학생 수를 입력 받아서 학생 수만큼의 배열을 생성하고자 하는 경우 아래와 같이 입력하면 에러가 발생합니다.

```
int Num;
printf("학생수를 입력해 주세요. ");
scanf("%d",&Num);
int Score[Num];
```

- 배열의 크기는 상수로만 지정할 수 있기 때문입니다.
- Visual C++ 에서는 에러를 발생시키지만 UNIX에서는 운영체제 종류에 따라 에러를 발생시키지 않을 수 있습니다.
- 배열은 스택에 생성되므로 기본적으로 1MB이상의 메모리 공간을 사용할 수 없지만 동적으로 메모리를 할당받는 경우는 힙을 사용하므로 크기는 4GB정도 까지 설정할 수 있습니다.
- 즉 사이즈가 큰 변수를 만들 때도 동적 메모리 할당을 사용해야 합니다.

함수(Function)

특정한 일들을 처리할 수 있는 부분적인 프로그램을 만들기 위해서 C언어에서는 함수를 사용하게 됩니다. printf(), scanf()등도 일종의 함수로서 VC++가 제공하는 라이브러리에 있는 함수이며 이러한 함수를 Maker Function 이라고 부릅니다.

1. 함수

- 자주 사용하는 기능을 하나의 이름으로 미리 정의해두고 그 이름을 호출하는 것으로 기능을 사용하도록 해주는 것을 함수라고 합니다.
- 함수를 사용하는 이유
  ① 프로그램 코드(code)의 불필요한 반복을 피하기 위한 수단을 제공
  ② 프로그램을 보다 체계적이고 간결하게 작성하기 위한 수단을 제공
  ③ 대형프로그램을 여러 개의 부분으로 나누어서 프로그램하고 컴파일을 가능하게 합니다.

- 함수의 종류
  - 컴파일러가 제공하는 함수 – Maker Function
  - 사용자가 정의해서 사용하는 사용자 정의 함수가 있습니다.

1. 함수

- 컴파일러가 제공하는 함수를 사용하기 위해서는 그 함수가 정의된 헤더파일을 소스 파일에 포함시켜야 하고 이 구문이 #include 구문입니다.
- 사용자 정의 함수를 사용하기 위해서는 함수를 먼저 정의되어야 하며 그 형식은 다음과 같습니다.

```
결과형 함수명(매개변수 선언 및 리스트) // 결과형을 제외한 이 부분을 함수 원형이라고 합니다.
{
        함수내부의 변수 선언;
        명령문 나열;
}
```

  - ✓ 함수의 결과형은 함수가 반환하는 값의 자료형으로 함수가 반환하는 값이 없을 경우 결과형은 void
  - ✓ 결과형이 생략된 경우는 int형으로 간주
  - ✓ 함수에 데이터를 전달하고자 하는 경우에는 매개변수를 통해서 가능
  - ✓ 함수 안에서 선언된 변수들은 로컬변수(local variable)로서 이 함수 내에서만 사용할 수 있으며 처음에 값을 대입하지 않으면 사용할 수 없거나 의미 없는 값을 가지고 있습니다.
  - ✓ 함수는 호출 될 때 자신만의 스택을 할당받아서 사용하고 호출이 끝나면 그 스택을 반납하기 때문입니다.(스택의 크기는 1MB로 제한)
  - ✓ 함수의 결과를 전달하고자 하는 경우 리턴을 이용해서 결과를 전달
- 함수의 호출: 함수는 공유 영역에 코드로 존재하며 함수를 호출하면 함수 내부로 제어권이 이동됩니다.
  - ✓ 함수명(매개변수)

1. 함수

❖ **매개 변수가 없는 함수**

- ✓ 함수명 다음의 ( ) 안에 아무것도 없는 함수
- ✓ 함수 정의 형식

```
결과형 함수명()
{
   함수 내용
}
```

```
결과형 함수명();
명령문..

자료형 함수명()
{
   함수 내용
}
```

- ✓ 함수의 원형선언과 동시에 함수 내용을 입력해도 되고 나중에 함수의 내용을 작성해도 됩니다.
- ✓ 매개 변수 란에 매개 변수가 없음을 나타내기 위하여 void 라고 적어도 됩니다.

1. 함수

❖ 예제(+를 14번 출력하고 줄 바꿈을 해주는 함수를 생성해서 호출하는 예제)

```
#include <stdio.h>
void disp()
{
    int i;
    for(i=1; i<=14; i++)
    {
                        printf("+");
    }
    printf("\n");
}
int main()
{
    disp();
    printf("+ C Language +\n");
    disp();
            return 0;
}
```

1. 함수

❖ 예제(앞의 예제를 함수를 선언만 하고 나중에 구현한 예제)

```
#include <stdio.h>
void disp();
int main()
{
    disp();
    printf("+ C Language +\n");
    disp();
    return 0;
}
void disp()
{
    int i;
    for(i=1; i<=14; i++)
    {
                        printf("+");
    }
    printf("\n");
}
```

1. 함수

❖ **매개 변수가 있는 함수**

- ✓ 함수명 다음의 ( ) 안에 만드는 변수를 매개변수라 합니다.
- ✓ 매개변수(Parameter) 또는 인수(Argument)라고 하며 호출하는 함수에서 호출하는 함수에게 넘겨주는 데이터입니다.
- ✓ 함수 정의 형식

```
자료형 함수명(자료형 변수1, 자료형 변수2, ...)
{
    함수의 내용
}
```

- ✓ 이 경우에는 함수를 호출할 때 함수명(값1, 값2, ...)으로 호출하게 됩니다.

1. 함수

❖ 예제(앞의 함수를 +의 개수를 넘겨받는 형태로 수정한 예제)

```
#include <stdio.h>
void disp(int );
int main()
{
    int i;
    printf("+를몇개출력하실건가요:");
    scanf("%d", &i);
    disp(i);
    printf("+ C Language +\n");
    disp(i);
    return 0;
}
void disp(int n)
{
    int i;
    for(i=1; i<=n; i++)
            printf("+");
    printf("\n");
}
```

2. 매개변수 전달방법

- 함수에 사용되는 매개변수를 전달하는 방법은 두 가지가 있습니다.
- 그 하나는 값을 전달(call by value)하는 방법이고 다른 하나는 값이 저장되어 있는 번지(call by reference)를 전달하는 방법입니다.
- call by value
  - ✓ 함수를 호출할 때 매개변수에 바로 사용할 수 있는 값을 전달하는 방식
- call by reference
  - ✓ 매개 변수의 값이 다른 데이터의 주소를 전달하는 방식으로 포인터(주소 변수) 형의 매개 변수 전달 방식이라고 합니다.
  - ✓ 이때 매개 변수가 포인터 변수가 되고 호출 시 주소를 전달하게 되므로 주소에 해당하는 데이터의 변경이 호출하는 함수에서 전달한 데이터에 영향을 주게됩니다.

2. 매개변수 전달방법

❖ 예제(call by value)

```
#include <stdio.h>
void increment(int);
int main()
{
        int score = 10;
        printf("Before calling score = %d\n", score);
        increment(score);
        printf("After  calling score = %d\n", score);
        return 0;
}
void increment(int n)
{
   n += 10;
}
```

2. 매개변수 전달방법

❖ 예제(call by reference)

```
#include <stdio.h>
void increment(int *);
int main()
{
        int score = 10;
        printf("Before calling score = %d\n", score);
   increment(&score);
   printf("After  calling score = %d\n", score);
        return 0;
}
void increment(int *n)
{
   *n += 10;
}
```

2. 매개변수 전달방법

❖ 예제(call by value를 이용한 swap)

```
#include<stdio.h>
void swap(int , int );
int main( )
{
        int a=10, b=20;
        printf("main에서swap 함수를 호출하기 전의 값\n");
        printf("a = %d  b =%d \n", a, b);
        swap(a, b);
        printf("main에서swap 함수를 호출한 후의 값\n");
        printf("a = %d  b =%d \n", a, b);
        return 0;
}
void swap(int x, int y)
{
        int temp;
        printf("swap 함수에서 실행 전의 값\n");
        printf("x = %d  y =%d \n", x, y);

        temp=x;
        x=y;
        y=temp;

        printf("swap 함수에서 실행 후의 값\n");
        printf("x = %d  y =%d \n", x, y);
}
```

2. 매개변수 전달방법

❖ 예제(call by reference를 이용한 swap)

```
#include<stdio.h>
void swap(int * , int *);
int main( )
{
        int a=10, b=20;
   printf("main에서 swap 함수를 호출하기 전의 값\n");
        printf("a = %d  b =%d \n", a, b);
        swap(&a, &b);
        printf("main에서 swap 함수를 호출한 후의 값\n");
        printf("a = %d  b =%d \n", a, b);
        return 0;
}
void swap(int *x, int *y)
{
        int temp;
        printf("swap 함수 실행전의 값\n");
        printf("x = %d  y =%d \n", *x, *y);
      temp=*x;
        *x=*y;
        *y=temp;
        printf("swap 함수 실행후의 값\n");
        printf("x = %d  y =%d \n", *x, *y);
}
```

3. return

- return: 함수의 수행을 종료하고 함수를 호출한 곳으로 제어권을 이동하는 예약어
- return 뒤에 데이터가 기재되어 있으면 그 결과 값을 함수의 수행결과로 돌려줍니다.
- 함수가 반드시 return 값이 있는 것은 아니며 return 값이 없는 함수의 결과형은 void로 기재합니다.
- 함수 내의 제어가 return 문을 만나면 프로그램의 어느 부분에 있던지 상관없이 해당하는 값을 return하고 제어를 그 함수를 호출한 함수로 옮기게 되는 것입니다.

3. return

❖ 예제

```
#include <stdio.h>
int add(int , int );
int main()
{
        int a,b;
        printf("첫번째 숫자를 입력하세요:");
        scanf("%d", &a);
        printf("두번째 숫자를 입력하세요:");
        scanf("%d", &b);
        printf("두 수의 합= %d\n", add(a,b));
        return 0;
}
int add(int x, int y)
{
        return(x+y);
}
```

4. 재귀함수

- 재귀(recursion)는 자기 자신을 다시 호출하여 사용하는 것
- 재귀함수는 호출할 때마다 스택을 생성해서 메모리를 상당히 많이 소모하며 처리속도 또한 느리게 되므로 가급적 쓰지 않는 것을 기본으로 합니다.
- 하지만 수열을 구하는 프로그램 등에서 쓰이면 프로그램의 형태가 간결해지며 그 프로그램을 이해하기 쉬워 질 수 있습니다.

4. 재귀함수

❖ 예제(1부터 n까지의 합을 구하는 프로그램)

```
#include <stdio.h>
int sum (int n)
{

        int hab;
        if (n == 1)
                return(1);
        else
                 hab = n + sum(n-1);
        return(hab);
}

int main()
{
        int a, b;
        printf("합을 구할 수를 입력하세요:");
        scanf("%d", &a);
        b = sum(a);
        printf("1부터 %d까지의 합은 %d입니다\n",a,b);
        return 0;
}
```

4. 재귀함수

❖ 예제(피보나치 수열의 값과 합을 알아보는 프로그램)

```
#include<stdio.h>
int fibonacci(int n);
int main()
{
	int i;
	int x;
	int sum = 0;
	printf("알고싶은 피보나치 수열의 값을 입력하세요:");
	scanf("%d",&x);
	for(i=1; i<=x; i++)
	{
		printf("피보나치 수열의 값: %d\n", fibonacci(i));
		sum = sum+fibonacci(i);
	}
	printf("%d번째까지 피보나치 수열의 합: %d\n", x,sum);
	return 0;
}
int fibonacci(int n)
{
	if(n==1||n==2)
		return 1;
	else
		return fibonacci(n-1) + fibonacci(n-2);
}
```

4. 재귀함수

❖ 예제(하노이의 탑)

```
#include<stdio.h>
void hanoi(int n, int a, int b, int c);
int main()
{
        int x;
        printf("몇개의 고리를 옮기실건가요:");
        scanf("%d", &x);
        hanoi(x, 'a', 'b', 'c');
        return 0;
}

void hanoi(int n, int a, int b, int c)
{
        if(n==1)
                printf("%c에서 %d의 원반을 %c로 이동한다. \n", a, n, c);
        else
        {
                hanoi(n-1, a, c, b);
            printf("%c에서 %d의 원반을 %c로 이동한다.\n", a, n, c);
                hanoi(n-1, b, a, c);
        }
}
```

Quick Sort

❖Quick Sort

❶ 왼쪽 끝에서 오른쪽으로 움직이면서 크기를 비교하여 피봇보다 크거나 같은 원소를 찾아 L로 표시한다. 단, L은 R과 만나면 더 이상 오른쪽으로 이동하지 못하고 멈춘다.

❷ 오른쪽 끝에서 왼쪽으로 움직이면서 피봇보다 작은 원소를 찾아 R로 표시한다. 단, R은 L과 만나면 더 이상 왼쪽으로 이동하지 못하고 멈춘다.

❸-ⓐ ❶과 ❷에서 찾은 L 원소와 R 원소가 있는 경우, 서로 교환하고 L과 R의 현재 위치에서 ❶과 ❷ 작업을 다시 수행한다.

❸-ⓑ ❶~❷를 수행하면서 L과 R이 같은 원소에서 만나 멈춘 경우, 피봇과 R의 원소를 서로 교환한다. 교환된 자리를 피봇 위치로 확정하고 현재 단계의 퀵 정렬을 끝낸다.

❹ 피봇의 확정된 위치를 기준으로 만들어진 새로운 왼쪽 부분집합과 오른쪽 부분집합에 대해서 ❶~❸의 퀵 정렬을 순환적으로 반복 수행하는데, 모든 부분집합의 크기가 1 이하가 되면 전체 퀵 정렬을 종료한다.

Quick Sort

❖Quick Sort

- ✓ 정렬하지 않은 {69, 10, 30, 2, 16, 8, 31, 22} 자료를 퀵 정렬 방법으로 정렬하는 과정 (가운데 원소를 피봇으로 정함)
  - 1단계
    - 원소 2를 피봇으로 선택하고 퀵 정렬을 시작
    - L은 정렬 범위의 왼쪽 끝에서 오른쪽으로 움직이면서 피봇보다 크거나 같은 원소를 찾고, R은 정렬 범위의 오른쪽 끝에서 왼쪽으로 움직이면서 피봇보다 작은 원소를 찾음. L은 원소 69를 찾았지만, R은 피봇보다 작은 원소를 찾지 못한 상태로 원소 69에서 L과 만남. L과 R이 만나 더 이상 진행할 수 없는 상태가 되었으므로
    - 원소 69를 피봇과 자리를 교환하고 피봇 원소 2의 위치를 확정

Quick Sort

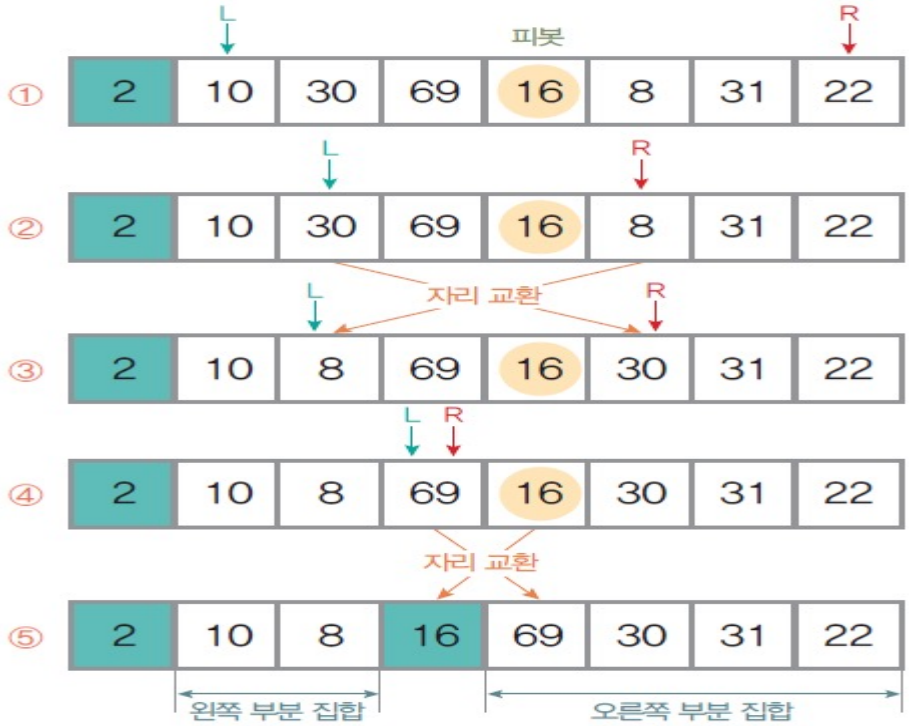
❖Quick Sort

- ✓ 정렬하지 않은 {69, 10, 30, 2, 16, 8, 31, 22} 자료를 퀵 정렬 방법으로 정렬하는 과정 (가운데 원소를 피봇으로 정함)
  - 2단계: 위치가 확정된 피봇 2의 왼쪽 부분집합은 공집합이므로 퀵 정렬을 수행하지 않고, 오른쪽 부분집합에 대해서 퀵 정렬을 수행
    - 오른쪽 부분집합의 원소가 일곱 개이므로 가운데 있는 원소 16을 피봇으로 선택하고 퀵 정렬을 시작.
    - L은 오른쪽으로 움직이면서 피봇보다 크거나 같은 원소인 30을 찾고 R은 왼쪽으로 움직이면서 피봇보다 작은 원소인 8을 찾음.
    - L이 찾은 30과 R이 찾은 8을 서로 자리를 교환한 후 현재 위치에서 L은 다시 오른쪽으로 움직이면서 피봇보다 크거나 같은 원소69를 찾고 R은 피봇보다 작은 원소를 찾음
    - R이 원소 69에서 L과 만나 더 이상 진행할 수 없는 상태가 되었으므로,
    - 원소 69를 피봇과 교환하고 피봇 원소 16의 위치를 확정

Quick Sort

❖Quick Sort

- ✓ 정렬하지 않은 {69, 10, 30, 2, 16, 8, 31, 22} 자료를 퀵 정렬 방법으로 정렬하는 과정 (가운데 원소를 피봇으로 정함)
  - 2단계: 위치가 확정된 피봇 2의 왼쪽 부분집합은 공집합이므로 퀵 정렬을 수행하지 않고, 오른쪽 부분집합에 대해서 퀵 정렬을 수행



Quick Sort

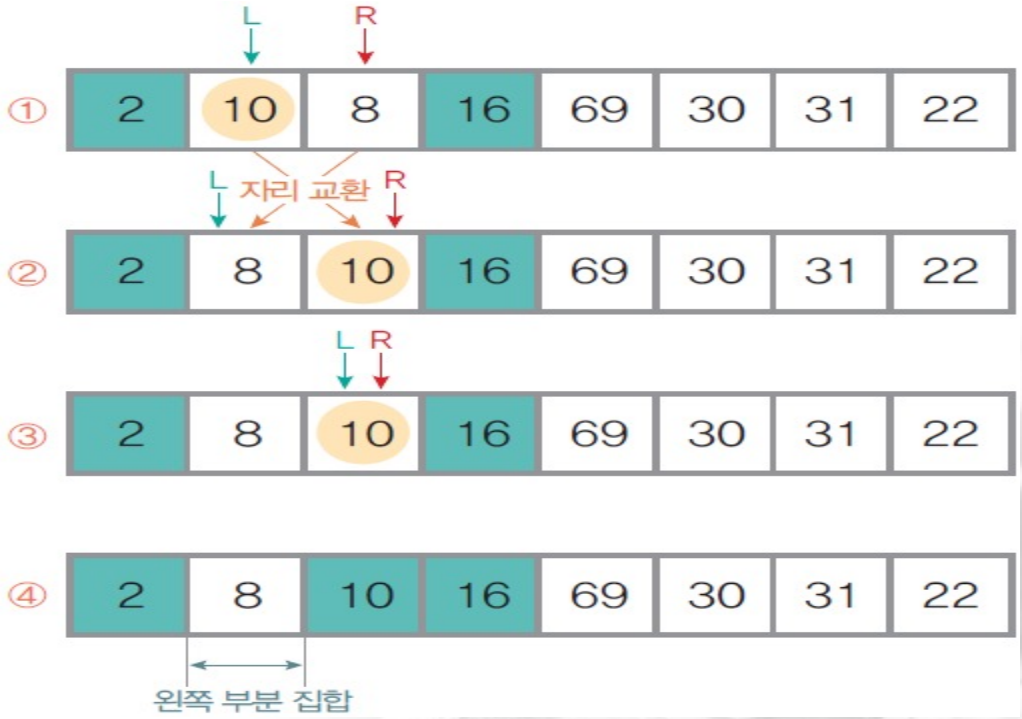
❖Quick Sort

- ✓ 정렬하지 않은 {69, 10, 30, 2, 16, 8, 31, 22} 자료를 퀵 정렬 방법으로 정렬하는 과정 (가운데 원소를 피봇으로 정함)
  - 3단계: 위치가 확정된 피봇 16을 기준으로 새로 생긴 왼쪽 부분집합에서 퀵 정렬을 수행
    - 원소 10을 피봇으로 선택하고 L은 오른쪽으로 움직이면서 피봇보다 크거나 같은 원소를, R은 왼쪽으로 움직이면서 피봇보다 작은 원소를 찾음
    - L이 찾은 10과 R이 찾은 8을 서로 교환
    - 현재 위치에서 L은 다시 오른쪽으로 움직이다가 원소 10에서 R과 만나서 멈추게 됨. 더 이상 진행할 수 없는 상태가 되었으므로,
    - R의 원소와 피봇을 교환하고 피봇 원소 10의 위치를 확정한다(이 경우에는 R과 피봇 위치가 같았으므로, 자리를 교환하기 전과 후의 상태가 같음)

Quick Sort

❖Quick Sort

- ✓ 정렬하지 않은 {69, 10, 30, 2, 16, 8, 31, 22} 자료를 퀵 정렬 방법으로 정렬하는 과정 (가운데 원소를 피봇으로 정함)
  - 3단계: 위치가 확정된 피봇 16을 기준으로 새로 생긴 왼쪽 부분집합에서 퀵 정렬을 수행



Quick Sort

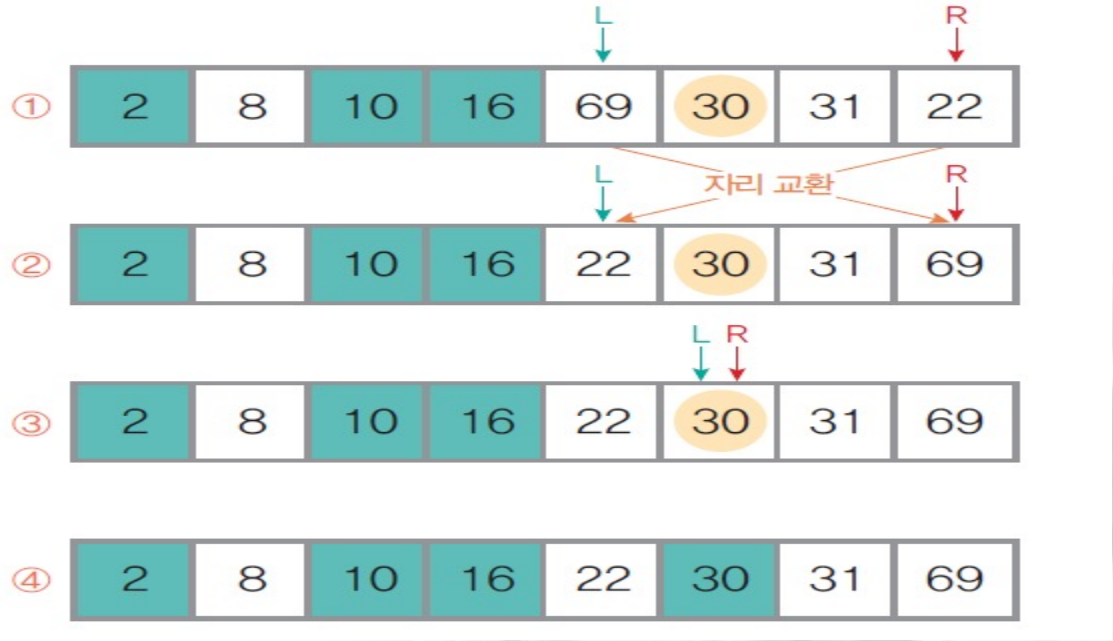
❖Quick Sort

- ✓ 정렬하지 않은 {69, 10, 30, 2, 16, 8, 31, 22} 자료를 퀵 정렬 방법으로 정렬하는 과정 (가운데 원소를 피봇으로 정함)
  - 4단계: 피봇 10의 왼쪽 부분집합은 원소가 한 개이므로 퀵 정렬을 수행하지 않고, 오른쪽 부분집합은 공집합이므로 역시 퀵 정렬을 수행하지 않고 [2]에서 피봇이었던 원소 16의 오른쪽 부분집합에 대해서 퀵 정렬을 수행
    - 오른쪽 부분집합의 원소가 네 개이므로 가운데 있는 원소 30을 피봇으로 선택. L은 오른쪽으로 이동하면서 피봇보다 크거나 같은 원소를 찾고 R은 왼쪽으로 움직이면서 피봇보다 작은 원소를 찾음
    - L이 찾은 69와 R이 찾은 22를 서로 교환. 현재 위치에서 다시 L은 피봇보다 크거나 같은 원소를 찾아 오른쪽으로 움직이고 R은 피봇보다 작은 원소를 찾아 왼쪽으로 움직이다가
    - 원소 30에서 L과 R이 만나 멈춤. 더 이상 진행할 수 없음
    - R의 원소와 피봇을 교환하고 피봇 원소30의 위치가 확정 (이 경우에 R과 피봇 위치가 같았으므로, 자리를 교환하기 전과 후의 상태가 같음).

Quick Sort

❖Quick Sort

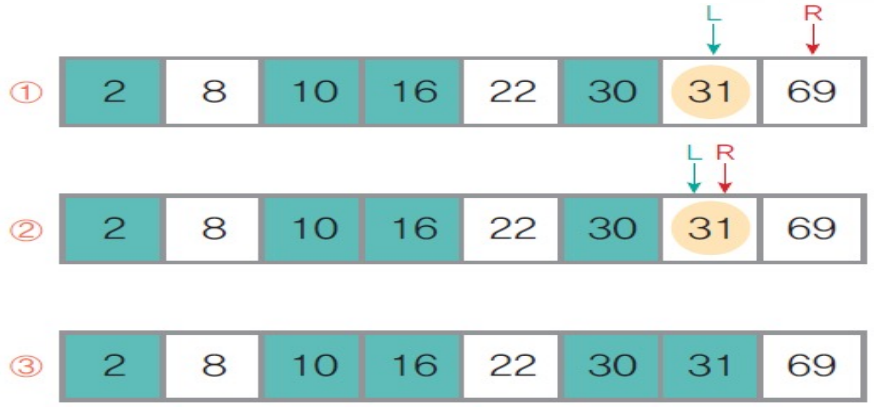
- ✓ 정렬하지 않은 {69, 10, 30, 2, 16, 8, 31, 22} 자료를 퀵 정렬 방법으로 정렬하는 과정 (가운데 원소를 피봇으로 정함)
  - 4단계: 피봇 10의 왼쪽 부분집합은 원소가 한 개이므로 퀵 정렬을 수행하지 않고, 오른쪽 부분집합은 공집합이므로 역시 퀵 정렬을 수행하지 않고 [2]에서 피봇이었던 원소 16의 오른쪽 부분집합에 대해서 퀵 정렬을 수행



Quick Sort

❖Quick Sort

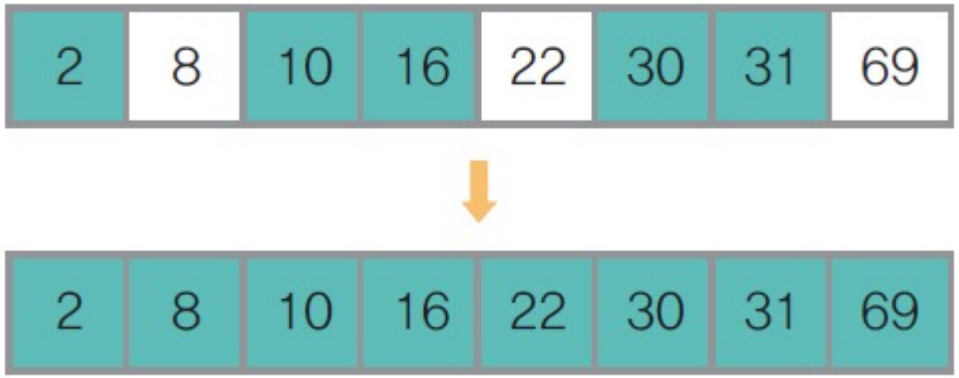
- ✓ 정렬하지 않은 {69, 10, 30, 2, 16, 8, 31, 22} 자료를 퀵 정렬 방법으로 정렬하는 과정 (가운데 원소를 피봇으로 정함)
  - 5단계: 피봇 30의 왼쪽 부분집합의 원소가 한 개이므로 퀵 정렬을 수행하지 않고, 오른쪽 부분집합에 대해서 퀵 정렬을 수행
    - 피봇은 31을 선택하고 L은 오른쪽으로 움직이면서 피봇보다 크거나 같은 원소를 찾고 R은 왼쪽으로 움직이면서 피봇보다 작은 원소를 찾음
    - L과 R이 원소 31에서 만나 더 이상 진행하지 못하는 상태가 되어
    - 원소 31을 피봇과 교환하여 위치를 확정(이 경우에는 R과 피봇 위치가 같았으므로, 자리를 교환하기 전과 후의 상태가 같음)



Quick Sort

❖Quick Sort

- ✓ 정렬하지 않은 {69, 10, 30, 2, 16, 8, 31, 22} 자료를 퀵 정렬 방법으로 정렬하는 과정 (가운데 원소를 피봇으로 정함)
  - 6단계: 피봇 31의 오른쪽 부분집합의 원소가 한 개이므로 퀵 정렬을 수행하지 않지 않고 모든 부분 집합의 크기가 1 이하이므로 전체 퀵 정렬을 종료



Quick Sort

❖ 예제(quick sort)

```
#include <stdio.h>
//left는 quick 정렬을 수행할 왼쪽의 인덱스
//right는 quick 정렬을 수행할 오른쪽의 인덱스
//data는 정렬할 배열
//len은 전체 데이터 개수
void quickSort(int left, int right, int data[], int len) {
    int i;
    int temp;
    // 정렬 결과 출력
    for (i=0; i<len; i++) {
        temp = data[i];
        printf("%d\t",temp);
    }
    printf("\n");
```

Quick Sort

❖ 예제(quick sort)

```
// 가장 왼쪽의 데이터를 pivot으로 설정
int pivot = left;
// 피봇의 위치를 j에 대입
int j = pivot;
// 피봇과 비교할 데이터의 위치의 초기값을 설정
i = left + 1;
    if (left < right) {
            for (; i <= right; i=i+1) {
                if (data[i] < data[pivot]) {
                    j = j + 1;
                    int temp = data[j];
                    data[j] = data[i];
                    data[i] = temp;
                }
            }

            int temp = data[left];
            data[left] = data[j];
            data[j] = temp;

            pivot = j;
```

Quick Sort

❖ 예제(quick sort)

```
quickSort(left, pivot - 1, data, len);
            quickSort(pivot + 1, right, data, len);
        }

    }
```

Quick Sort

❖ 예제(quick sort)

```
int main(int argc, const char * argv[]) {
   int list[] = { 69, 10, 30, 2, 14, 8, 31, 22, 16 };
    int i, temp;
    int len = sizeof(list) / sizeof(list[0]);
    // 퀵 정렬 수행(left: 배열의 시작 = 0, right: 배열의 끝 = 8)
    quickSort(0, len - 1, list, len);

    // 정렬 결과 출력
    printf("정렬 후\n");
    for (i=0; i<len; i++) {
        temp = list[i];
        printf("%d\t",temp);
    }
    printf("\n");

    return 0;
}
```

Quick Sort

❖ 예제(quick sort)

69	10	30	2	14	8	31	22	16
16	10	30	2	14	8	31	22	69
8	10	2	14	16	30	31	22	69
2	8	10	14	16	30	31	22	69
2	8	10	14	16	30	31	22	69
2	8	10	14	16	30	31	22	69
2	8	10	14	16	30	31	22	69
2	8	10	14	16	30	31	22	69
2	8	10	14	16	22	30	31	69
2	8	10	14	16	22	30	31	69
2	8	10	14	16	22	30	31	69
정렬 후								
2	8	10	14	16	22	30	31	69

5. 함수 포인터

- 함수 포인터(Pointer to Function)란 함수를 가리키는 포인터입니다.
- 다른 언어에서는 delegate란 표현을 사용하기도 합니다.
- 포인터 변수는 메모리의 번지를 저장하는 변수인데 함수도 메모리에 존재하며 시작 번지가 있으므로 포인터 변수로 가리킬 수 있습니다.
- 일반적인 포인터는 변수가 저장되어 있는 번지를 가리키지만 함수 포인터는 함수의 시작 번지를 가리킨다는 점에서 다릅니다.
- 함수 포인터의 선언 형식

  결과형 (*변수명)(매개변수 리스트);

  int func(int a);

  정수형 매개변수를 하나 취하고 정수형을 리턴하는 func라는 함수의 원형이 있습니다.

  이런 함수를 가리킬 수 있는 함수 포인터 pf를 선언하는 절차는 다음과 같습니다.

  ① int pf(int a); // 함수명을 변수명으로 바꾼다.

  ② int *pf(int a); // 변수명 앞에 *를 붙인다.

  ③ int (*pf)(int); // 변수를 괄호로 묶어 버리면 됩니다.

  매개 변수의 이름은 생략 가능합니다.

5. 함수 포인터

- 함수 포인터에 함수 대입
  - ✓ 함수포인터 = 함수명
  - ✓ 예) 위의 경우
  - ✓ pf = func;
  - ✓ 이처럼 함수이름을 포인터 변수에 직접 대입할 수 있습니다.
  - ✓ 그 이유는 함수명만 사용하면 함수의 시작 주소를 가리키는 주소 상수이기 입니다.

- 함수 포인터의 호출
  - ✓ (*함수포인터)(매개변수);
  - ✓ (함수포인터)(매개변수);
  - ✓ 위 두 줄은 모두 동일한 문장으로 취급합니다.
  - ✓ 어떻게 입력하더라도 결과는 같습니다.

5. 함수 포인터

❖ 예제(함수 포인터)

```
#include <stdio.h>
int sum(int a)
{
    int i, sum=0;
            for(i=1; i<=a; i++)
            {
                    sum = sum + i;
            }
            return sum;
}
int main()
{
    int n;
            int (*pf)(int a);
            printf("하나의 양수를 입력하세요");
            scanf("%d", &n);
            pf=sum;
      printf("1부터%d 까지의합= %d\n",n, (*pf)(n));
            printf("1부터%d 까지의합= %d\n",n, (pf)(n));
            return 0;
}
```

5. 함수 포인터

- 함수 포인터를 사용하는 이유
  - ✓ 함수 포인터를 이용하면 동일한 모양의 함수를 호출할 때 하나의 문장으로 처리가 가능
  - ✓ 함수의 원형이 동일한 경우라면 함수 포인터를 이용해서 여러 개의 함수에 접근 가능
  - ✓ 윈도우 프로그래밍에서 스레드 처리 함수 같은 경우는 사용자가 어떤 함수를 스레드 함수로 사용할 지 알 수 없기 때문에 미리 함수의 원형을 설정해두고 함수 명만을 등록하도록 하는 형태를 취하게 됩니다.
  - ✓ 함수를 다른 함수의 인수로 전달 할 수 있기 때문입니다.

5. 함수 포인터

❖ 예제(함수 포인터)

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int add(int a, int b)
{
        return a+b;
}
int sub(int a, int b)
{
        return a-b;
}
int main()
{
        int x,y; char menu;
        int (*pf)(int a, int b);
        printf("첫번째 연산에 사용할 정수 값을 입력하세요=>");
        scanf("%d", &x);
        printf("두번째 연산에 사용할 정수 값을 입력하세요=>");
        scanf("%d", &y);
```

5. 함수 포인터

❖ 예제(함수 포인터)

```
        while(1)
        {       printf("1. 두수의합2. 두수의차\n");
                menu = getch();
                if (menu =='1')
                        pf=add; break;
                else if(menu == '2')
                        pf=sub; break;
                else
                {
                        printf("잘못된 메뉴 선택입니다\n");
                        continue;
                }
        }
    printf("결과는%d입니다.\n",(*pf)(x,y));
        return 0;
}
```

클래스와 메소드

- 자바는 완전한 객체지향 언어
- 자바는 모든 코드를 클래스 안에 작성해야 함
- 클래스는 바로 사용이 가능한 경우도 있지만 인스턴스를 만들어서 사용하는 것이 일반적
- 변수는 데이터를 저장하는 것이 목적이고 함수나 메소드는 일을 수행하는 것이 목적
- 자바에서 어떤 작업을 수행시키고자 하면 클래스나 인스턴스를 만들어서 메소드를 호출하면 됨

```
System.out.println( “자바에서 작업 수행” );
//System은 클래스 이름이고 out은 클래스 안에 속한 인스턴스이고 println이 메소드

String str = new String( “자바의 문자열 인스턴스 만들기” ); //인스턴스 생성
System.out.println(str.substring(0, 5)); //인스턴스가 호출한 메소드의 결과를 출력하기
```

Python

- 특징
  - ✓ Python은 데이터의 자료형을 데이터를 대입할 때 결정하기 때문에 변수를 만들 때 자료형을 기재할 필요가 없음
  - ✓ 줄 단위로 번역해서 실행하기 때문에 하나의 명령어를 구분하기 위한 ;이 필요없지만 한 줄에 2개의 명령어를 작성하는 경우에는 명령어를 구분하기 위해서 ;을 입력
  - ✓ 하나의 블럭을 만들고자 할 때는 : 을 하고 들여쓰기를 이용해서 코드를 작성
  - ✓ 들여쓰기가 같으면 동일한 코드 블록
  - ✓ 코드 블록은 제어문, 함수(메소드), 클래스
- 문자열 입력은 input('메시지')
  - ✓ 문자열로만 리턴하기 때문에 숫자로 변환할 때는 int(input('메시지')) 의 형태로 변환
- 출력은 print
  - ✓ 메시지나 데이터를 ,로 구분해서 대입해서 출력
  - ✓ '서식' % 데이터의 형태로 입력해서 서식을 적용해서 출력하는 것도 가능

Python

- ❖ Python은 문자열을 문자 데이터의 모임으로 취급
- ❖ 배열이 없고 list가 존재
- ❖ 파이썬의 list에서 주의할 점은 인덱스의 음수가 허용된다는 점인데 음수는 뒤에서부터 찾아옵니다.

```
msg = “Hello Python”
print(msg[1]) //e 가 출력
print(msg[-1]) //n 출력
```

- ❖ 슬라이싱 : 인덱스에 범위를 지정해서 데이터를 잘라내는 것
  - ✓ [시작위치:종료위치:간격]
  - ✓ [시작위치:종료위치] -> 간격은 1
  - ✓ [시작위치:] -> 시작위치부터 끝까지
  - ✓ [:종료위치] -> 처음부터 종류위치까지
  - ✓ [::간격] -> 처음부터 끝까지 간격을 가지고 추출
  - ✓ [::], [:] -> 처음부터 끝까지 1간격으로

Python

- Python의 Collection
  - ✓ list : [데이터 나열]
  - ✓ set : {데이터 나열}
  - ✓ map : {key:value, key:value…}

라이브러리 & 프레임워크

- 라이브러리 나 프레임워크는 프로그램을 개발할 때 많이 사용될 것 같은 함수나 클래스를 미리 만들어서 제공하는 것
- 유사한 개념으로 솔루션이 있는데 솔루션은 특정한 작업을 편리하게 만들기 위한 것으로 유사한 애플리케이션을 여러 번 만드는 경우 그 애플리케이션에 특화된 형태로 제공
- 프로그래밍 언어가 제공하는 라이브러리를 표준 라이브러리라고 하고 외부 개발자들이 만들어서 제공하는 라이브러리는 외부 라이브러리 또는 3rd Party 라이브러리라고도 합니다.
- 이러한 라이브러리 제공 때문에 Java 나 Python, R 등이 프로그래밍하기 편리하다고 함
- C언어에서 많이 사용하는 표준 라이브러리
  - ✓ stdio.h: 입출력 라이브러리
  - ✓ stdlib.h: 표준 라이브러리로 컴파일러마다 제공되는 것이 다름
  - ✓ string.h: 문자열 라이브러리
  - ✓ memory.h: 메모리 관련 라이브러리

라이브러리 & 프레임워크

- Java 언어에서 많이 사용하는 패키지
  - ✓ java.lang: 표준 패키지
  - ✓ java.util: 자료구조 패키지
  - ✓ java.io: 입출력 패키지
  - ✓ java.net: 네트워크 패키지
  - ✓ java.sql: 데이터베이스 사용 패키지

예외(Exception)

❖컴파일 에러(error): .java 파일을 .class 파일로 만드는 과정에서 발생하는 오류로 .java 파일에 오류가 있거나 jvm이 인식할 수 없는 클래스 사용으로 인한 오류

❖예외(exception): 컴파일은 성공적으로 수행되어 클래스가 만들어 졌지만 실행 도중 외부 요인이나 잘못된 입력 등으로 발생하는 오류

❖논리적 에러: 컴파일 이나 런타임 시에 에러가 발생하지는 않지만 정상적으로 실행이 되었지만 의도하지 않은 결과가 나오는 경우

❖단언(assert): 개발자가 의도적으로 특정 조건을 만족했을 때 만 프로그램이 정상적으로 동작하도록 만드는 것

❖컴파일 에러가 발생하면 에러를 수정

❖컴파일 에러는 eclipse 나 IntelliJ, NetBeans와 같은 IDE를 사용하면 명확하게 표시를 해주므로 비교적 찾기 쉬움

❖런타임 에러나 논리적 에러는 프로그램 외부의 문제인지 논리적인 오류인지 여러 가지 상황을 고려해야 하므로 에러의 원인을 찾기 어려운 경우가 많음

예외(Exception)

❖예외: 문법적으로는 이상이 없어서 컴파일 시점에는 아무런 문제가 없지만 프로그램 실행 중에 발생하는 예기치 않은 사건으로 인해 프로그램이 중단되는 런타임 에러

❖예외가 발생하는 경우
- ✓ 정수를 0으로 나누는 경우
- ✓ 배열의 첨자가 범위를 벗어나거나 음수를 사용하는 경우
- ✓ 부적절한 형 변환이 일어나는 경우
- ✓ 입출력을 위한 파일이 없는 경우 등

❖예외처리의 용도
- ✓ 정상 종료
- ✓ 예외내용 기록
- ✓ 예외 발생 시 무시하고 계속 실행
- ✓ 정상적인 값으로 변경해서 실행

Java 예외 관련 클래스

❖Exception 클래스의 주요 하위 클래스들

NoSuchMethodException	메소드가 존재하지 않을 때
ClassNotFoundException	클래스가 존재하지 않을 때
CloneNotSupportedException	객체의 복제가 지원되지 않는 상황에서 복제를 시도하고자 하는 경우
IllegalAccessException	클래스에 대한 부정적인 접근
InstantiationException	추상클래스나 인터페이스로부터 객체를 생성하고자 하는 경우
InterruptedException	스레드가 인터럽트 되었을 때
RuntimeException	실행시간에 예외가 발생한 경우
IOException	입출력과 관련된 예외 처리 EOFException,FileNotFoundException,InterruptedException

Java 예외 관련 클래스

❖RuntimeException 클래스의 주요 하위 클래스: 프로그래머의 실수로 발생하는 예외로 소스 코드를 수정하면 해결되는 경우가 많음

ArithmeticException	0으로 나누는 등의 산술적인 예외
NegativeArraySizeException	배열의 크기를 지정할 때 음수의 사용
NullPointerException	null인 참조형 변수가 메소드나 멤버 변수에 접근하고자 하는 경우
ArrayIndexOutOfBoundsException	배열에서 인덱스 범위를 초과해서 사용하는 경우
NumberFormatException	숫자로 변경이 되지 않는 데이터를 숫자로 변경했을 때 발생
ClassCastException	객체의 타입을 변경이 불가능한 타입으로 변경하는 경우
SecurityException	보안을 이유로 메소드를 수행할 수 없을 때

예외 처리

❖예외 처리 방법 중 예외가 발생한 메소드 내에서 직접 처리하는 방법: try, catch, finally 블록 사용(finally를 사용하는 경우 catch 생략 가능) – if 문 처럼 동작

```
try {
..... // try 블록: 예외가 발생할 가능성이 있는 문장을 지정
}
catch(예외타입1 매개변수1) {
  .....  // 예외 처리 블록1: 예외의 종류에 따라 처리하는 처리
}
catch(예외타입N 매개변수N) {
  ..... // 예외 처리 블록 N
}
finally{
  ......  // finally 블록: 예외의 발생여부와 상관없이 무조건 수행되는 블록
        // 생략가능
}
```