18 프로그램언어 기초

Section 1. 프로그램 언어 특성

1. 프로그래밍 언어의 유형 분류

⑴ 개발 편의성 측면에 따른 분류

① 저급언어(Low-Level Language)

- 컴퓨터가 직접 이해할 수 있는 언어
- 실행속도는 빠르나 기계마다 기계어가 상이하여 호환성이 없고, 유지관리가 어렵다.

② 고급언어(High-Level Language)

- 인간이 이해할 수 있는 소스코드로 되어 있는 언어
- 실행을 위해서 저급언어로 번역하는 과정이 필요하다.

⑵ 실행 및 구현 방식에 따른 분류

① 명령형 언어(Imperative Language)

- 컴퓨터가 동작해야 할 절차를 통해 프로그래밍의 상태를 변경시키는 구문에 중점을 둔 방식
- FORTRAN, C 등

② 함수형 언어(Functional Language)

- 자료 처리를 수학적 함수의 계산으로 취급하고 상태와 가변 데이터를 멀리하는 프로그래밍 패러다임 중 하나다.
- LISP, Scala 등

③ 논리형 언어(Logic Language)

- 논리 문장을 이용하여 프로그램을 표현하고 조건이 만족되면 연관된 규칙이 실행되는 방식
- PROLOG 등

④ 객체지향언어(Object-Oriented Language)

- 객체 간의 메시지 통신을 이용하여 동작하는 방식
- JAVA, C++ 등

⑶ 빌드(Build) 방식에 따른 분류

① 컴파일 언어(Compile Language)

- 소스코드를 목적 코드로 변환하여 실행하는 방식
- 소스코드를 컴퓨터가 이해할 수 있는 언어로 번역을 하는 방식
- C, C++ 등

② 인터프리터 언어(Interpreter Language)

- 소스코드를 한 줄씩 실행하는 방식
- 소스코드를 컴퓨터가 이해할 수 있는 언어로 통역을 하는 방식
- BASIC, Python 등

③ 바이트 코드 언어(Byte Code Language)

- 컴파일을 통해 가상머신이 번역할 수 있는 Byte Code로 변환되고, 가상머신이 기계어로 번역하는 방식
- JAVA, Scala 등

2. 절차적 프로그래밍 언어

(1) 절차적 프로그래밍 언어 개념

- 일련의 처리 절차를 정해진 문법에 따라 순서대로 기술하는 언어
- Procedure(루틴, 서브루틴, 메소드, 함수)를 이용하여 작성하는 프로그래밍 스타일
- 순차적인 처리를 중요시 여기며, 프로그램 전체가 유기적으로 연결되도록 만드는 프로그래밍 기법이다.

(2) 절차적 프로그래밍 언어 장/단점

① 장점

- 함수를 통해 코드의 재활용성이 높아진다.
- 컴퓨터의 처리 구조와 유사하며 실행속도가 빠르다.

② 단점

- 프로그램 분석이 어려움
- 유지보수나 코드의 수정이 어려움

(2) 절차적 언어의 종류

① C 언어

- 1972년 데니스 리치에 의해 개발
- 운영체제를 작성하기 위한 시스템 프로그래밍 언어로 개발
- 이식성이 뛰어남

② COBOL

- 금융 및 인적 자원과 같은 비즈니스 컴퓨터 프로그램을 위해 설계됨
- 4개의 DIVISION으로 구성

③ FORTRAN

- 과학 계산용으로 주로 사용되는 언어

④ BASIC

- 교육용으로 개발되어 언어의 문법이 쉽다.

3. 객체지향 프로그래밍 언어

⑴ 객체지향 프로그래밍 언어 개념

- 만들고자 하는 소프트웨어의 구조를 객체로 만들고, 객체들끼리 상호작용 하도록 만드는 프로그래밍 언어
- 객체들을 적절히 조립하고, 연결하여 소프트웨어가 동작하도록 만드는 것

⑵ 객체지향 프로그래밍 언어 특징

① 캡슐화(Encapsulation)

- 데이터와 연산을 하나로 묶어 캡슐처럼 만드는 것
- 캡슐화된 객체의 세부 내용이 외부에 은폐(정보 은닉)되어, 변경이 발생할 때 오류의 파급효과가 적다.
- 캡슐화된 객체들은 재사용이 용이하다.

② 정보은닉(Information Hiding)

- 다른 객체에게 자신의 정보를 숨기고 자신의 연산만을 통하여 접근을 허용하는 것

③ 상속(Inheritance)

- 상위 클래스의 데이터와 연산을 하위 클래스가 물려받는 것
- private 한 요소 외의 모든 내용을 하위 클래스가 사용할 수 있다.
- extends 라는 키워드를 사용한다.

④ 다형성(Polymorphism)

- 메시지를 통해 연산을 수행할 때, 같은 이름을 가진 여러 메서드 중 특정 메서드가 호출되는 것
- 오버로딩과 오버라이딩이 있다.

⑤ 추상화(Abstarction)

- 불필요한 부분을 생략하고 객체의 속성 중 가장 중요한 것에만 중점을 두어 개략화하는 것

⑶ 객체지향 프로그래밍 언어 장/단점

① 장점

- 재 사용성
- 생산성 향상
- 자연적인 모델링
- 유지보수의 우수성

② 단점

- 개발 속도가 느려진다.
- 실행 속도가 대체적으로 느리다.
- 코딩 난이도 상승

⑷ 객체지향 프로그래밍 언어의 종류

- JAVA
  - 객체지향언어의 대표적인 언어
- 시뮬라67
  - 최초의 객체지향언어

- 스몰토크(Smalltalk)
  - 최초로 GUI를 제공하는 언어
- 오브젝티브-C(Objective-C)
  - 애플의 운영체제인 iOS에서 사용되는 언어
- C++
  - 객체지향성이 더해진 C 언어의 확장형
- 파이썬(Python)
  - 플랫폼 독립적이며 인터프리터식, 객체지향적, 동적 타이핑(dynamically typed) 대화형 언어

4. 스크립트 언어

(1) 스크립트 언어 개념

- 응용 소프트웨어를 제어하는 컴퓨터 프로그래밍 언어
- 다른 응용 프로그램에 삽입되어서 동작하는 프로그래밍 언어
- 최종사용자가 응용 프로그램의 동작을 사용자의 요구에 맞게 수행할 수 있도록 해준다.

(2) 스크립트 언어의 종류

- 자바스크립트(JavaScript)
  - HTML 문서 내에 삽입되어 사용되며, 동적인 인터렉티브한 페이지를 만들 수 있게 한다.
- JSP
  - 자바 서버 페이지(JavaServer Pages)의 약자
  - HTML 페이지 안에 자바(Java) 코드를 직접 삽입하여 웹 서버에서 동적으로 웹 페이지를 생성하여 웹브라우저가 표현할 수 있도록 전달해 주는 스크립트 프로그래밍 언어
- PHP
  - HTML 문서 안에 포함하여 작동하는 서버 측 스크립트 프로그래밍 언어
  - 리눅스 운영체제에 아파치(Apache) 웹서버를 설치하고 마이에스큐엘(MySQL) DB 환경에서 PHP 프로그래밍 언어가 주로 사용된다.
- ASP
  - 마이크로소프트의 윈도우 서버에서 운영되는 스크립트 방식의 웹 프로그래밍 언어
- Perl
  - 유닉스 계열의 운영체제에서 사용하는 스크립트 프로그래밍 언어
- Python
  - 네덜란드 암스텔담의 귀도 반 로썸(Guido van Rossum)이 개발한 객체 지향 스크립트 프로그래밍 언어
  - 별도의 컴파일 과정이 필요없다.
  - 리눅스나 윈도우 등 특정 플랫폼에 독립적이다.
- VBScript
  - 마이크로소프트 Visual Basic(비주얼베이직) 기반의 스크립트 프로그래밍 언어

5. 선언형 언어

(1) 선언형 언어 개념

- 프로그램이 수행해야 하는 문제를 기술하는 언어
- 목표를 명시하고 알고리즘은 명시하지 않는다.
- 가독성이나 재사용성이 좋고, 오류가 적음

(2) 선언형 언어의 종류

- 하스켈(Haskell)
  - 함수형 프로그래밍 언어
- HTML
  - 웹 콘텐츠의 의미와 구조를 정의할 때 사용
- SQL
  - 데이터를 관리하기 위해 설계된 특수 목적의 프로그래밍 언어

Section 2. C언어

1. 변수

⑴ 변수의 개념

- 값을 저장해 놓는 기억공간
- 변수에 저장된 데이터의 값은 언제든지 변경될 수 있다.
- 변수마다 정해진 자료형이 있고, 할당된 값을 가지고 있다.

⑵ 변수명 작성 규칙

- 모든 변수는 사용되기 전에 선언한다.
- 영문자 또는 언더바(_)로만 시작할 수 있다.
- 중간에 숫자와 언더바(_)를 사용할 수 있다.
- 중간에 공백이 올 수 없다.
- 언더바(_) 이외의 특수문자는 사용할 수 없다.
- 대소문자를 구분한다.
- 예약어(void, int, char, for 등)는 변수로 사용할 수 없다.

⑶ C언어 예약어

구분	종류
자료형	char, int, float, double, enum, void, struct, union, short, long, signed, unsigned 등
기억분류	auto, register, static, extern
제어문	if, else, for, while, do, switch, case, default, break, continue, return, goto
기타	sizeof, const, volatile

⑷ 자료형(Data Type)

- 효율적인 메모리 사용을 위하여 여러 종류의 자료형 존재
- C언어 자료형

종류	데이터 타입	크기	허용 범위
문자형	char	1Byte	-128~127
	unsigned char	1Byte	0~255
정수형	short	2byte	-32,768~32,767
	int	4Byte	-2,147,483,648~2,147,483,647
	long	4Byte	-2,147,483,648~2,147,483,647
	long long	8Byte	
실수형	float	4Byte	
	double	8Byte	
	long double	8Byte	

- JAVA언어 자료형

종류	데이터 타입	크기	허용 범위
논리형	boolean	1bit	true 또는 false
문자형	char	2Byte	0~65,535
정수형	byte	1byte	-128~127
	short	2Byte	-32,768~32,767
	int	4Byte	-2,147,483,648~2,147,483,647
	long	8Byte	
실수형	float	4Byte	
	double	8Byte	

(5) 변수의 선언

- 자료형 변수명 = 값;
  - int cnt = 10; // cnt 변수를 정수형으로 선언하고 초기값 10을 대입
  - int cnt; // cnt 변수를 정수형으로 선언하고, 초기값이 없을 때, 쓰레기 값이 대입
  - char a = 'C'; // a 변수를 문자형으로 선언하고, 초기값 C를 대입
- 잘못된 변수명의 예
  - 43User; // 첫 문자는 언더바나 영문으로만 시작 가능
  - my Data; // 중간에 공백 삽입
  - continue; // 예약어는 사용할 수 없다.

2. 연산자

(1) 산술 연산자

연산자	기능	산술 연산식	결과
+	더하기	10 + 5	15
-	빼기	10 - 3	7
*	곱하기	3 * 7	21
/	나누기	7 / 3	2
%	나머지	7 % 3	1
++	1증가		
--	1감소		

(2) 관계 연산자

연산자	기능	관계 연산식	결과
>	크다	10 > 1	1, true
>=	크거나 같다	10 >= 10	1, true
<	작다	10 < 1	0, false
<=	작거나 같다	10 <= 10	1, true
==	같다	10 == 5	0, false
!=	같지 않다	10 != 5	1, true

(3) 논리 연산자

연산자	기능	관계 연산식	결과
&&	AND	10 && 0	0, false
||	OR	10 || 0	1, true
!	NOT	!1	0, false

(4) 비트 연산자

연산자	기능	비트 연산식	결과
&	비트 AND	10 & 7	2
|	비트 OR	10 | 7	15
~	비트 not	~10	-11
^	비트 XOR	10 ^ 7	13
<<	좌 비트 이동	10 << 2	40
>>	우 비트 이동	10 >> 2	2

(5) 삼항 연산자

연산자	기능	삼항 연산식	결과
? :	3항 연산	10 > 3 ? 10 : 3	10

(6) 대입 연산자

연산자	기능	비트 연산식	결과
+=	덧셈 후 대입	a += 10;	
-=	뺄셈 후 대입	a -= 10;	
*=	곱셈 후 대입	a *= 10;	
/=	나눗셈 후 대입	a /= 10;	
%=	나머지 후 대입	a %= 10;	

(7) 연산자 우선순위

우선순위	분류	종류
1	단항 연산자	++, --, !
2	산술 연산자	*, /, +, -
3	시프트 연산자	<<, >>
4	관계 연산자	>, <, >=, <=, ==, !=
5	비트 연산자	&, ^, |
6	논리 연산자	&&, ||
7	삼항 연산자	? :
8	대입 연산자	=, +=, -=, *=, /=, %=

3. 입출력 함수

(1) printf() / scanf()

- 가장 많이 사용하는 표준 입출력함수로, <stdio.h> 헤더파일에 정의되어 있다.
- 사용 예

```
#include<stdio.h>
int main() {
    int sum;
    printf("input : ");
    scanf("%d", &sum);
    printf("%d ₩n", sum);
    return 0;
}
```

- 출력 변환 문자

구분	설명	사용 예	출력 값
%d	10진수	printf("%d", 10);	10
%o	8진수	printf("%o", 10);	12
%x	16진수	printf("%x", 10);	a
%f	실수	printf("%f", 1.3);	1.300000
%c	문자 1개	printf("%c", 'A');	A
%s	문자열	printf("%s", "abcde");	abcde

(2) getchar() / putchar()

- 단일 문자를 입출력하는 버퍼형 입출력 함수
- 사용 예

```
#include<stdio.h>
int main() {
    int i;
    i = getchar();
    printf("%d", i);   // 'A'를 입력했을 때, 65 출력
    putchar('A'); // A 출력
    putchar(65); // A 출력
}
```

⑶ gets() / puts()

- 문자열을 입출력 하는 입출력 함수
- 인자로 주소값을 입력 받는다.
- 사용 예

```
#include<stdio.h>
int main() {
	char str[20];
	gets(str);
	puts(str);
}
```

4. 제어문

(1) if 문

- 조건식의 결과에 따라 중괄호{ } 에 묶어놓은 블록의 실행 여부가 결정되는 조건문
- 조건식에는 true 또는 false 값을 산출 할 수 있는 boolean 변수가 온다.
- C언어에서는 0은 false(거짓), 0이 아닌 나머지는 모두 true(참)로 인식한다.
- 중괄호 블록이 없으면 다음에 있는 한 줄의 명령만 실행한다.
- 문법

```
// 단순 if 문
if( 조건식 ){
    참일 때의 명령문
}

// if~else 문
if( 조건식 ){
    참일 때의 명령문
}
else{
    거짓일 때의 명령문
}

// 다중 if 문
if( 조건식1 ){
    조건식 1이 참일 때 명령문
}
else if( 조건식2 ){
    조건식 2가 참일 때 명령문
}
else{
    모든 조건에 만족하지 않을 때의 명령문
}
```

- 예제

```
int main(){
     int score = 65;
     if( score >= 60 ){
          printf("합격 하셨습니다.");
     }
     else{
          printf("아쉽게도 불합격 하셨습니다.");
     }
}
```

(2) switch 문

- 비교할 변수가 어떤 값을 가지냐에 따라 실행문을 선택한다.
- 해당 실행문 수행 후 break 문을 만날 때까지 모든 실행문을 실행한다.
- 문법

```
switch(변수){
    case 값1 :
        명령문;
        break;
    case 값2 :
        명령문;
        break;
    default :
        명령문;
}
```

- 예제

```
char ch = 'A';
switch(ch){
    case 'A' :
        printf("ch의 값은 A입니다.");
        break;
    case 'B' :
        printf("ch의 값은 B입니다.");
        break;
    default :
        printf("ch의 값은 A과B가 아닌 다른 문자입니다.");
}
```

5. 반복문

(1) for 문

- 반복의 횟수가 정해진 반복문
- 자체적으로 초기식, 조건식, 증감식을 모두 포함하고 있는 반복문
- 문법

```
for (초기식; 조건식; 증감식)
{
    조건식의 결과가 참인 동안 반복할 명령문;
}
```

- 예제

```
int i;
int sum = 0;
for( i = 0; i < 5; i++ )
{
    printf("%d", i); // 0, 1, 2, 3, 4 출력
    sum += i;
}
printf("₩n i = %d", i); // i=5 출력
printf("₩n sum= %d", sum); // sum=10 출력
```

(2) while 문

- 특정 조건을 만족할 때까지 계속해서 주어진 명령문을 반복 실행
- 문법

```
while (조건식)
{
    조건식의 결과가 참인 동안 반복할 실행문;
}
```

- 예제

```
int i = 1;
int sum = 0;
while( i < 5 )
{
    printf("%d", i); // 1, 2, 3, 4 출력
    sum += i;
    i++;
}
printf("₩n i = %d", i); // i=5 출력
printf("₩n sum= %d", sum); // sum=10 출력
```

⑶ do~while 문

- 무조건 한번 반복을 실행한 후 조건식을 검사하여 조건에 만족하는 동안 반복 실행
- 문법

```
do {
    조건식의 결과가 참인 동안 반복할 실행문;
} while (조건식);
```

- 예제

```
int i = 4;
int sum = 0;
do
{
    printf("%d", i); // 4 출력
    sum += i;
    i++;
} while( i < 5 );
printf("₩n i = %d", i); // i=5 출력
printf("₩n sum= %d", sum); // sum=4 출력
```

⑷ continue 문

- continue 문을 만나게 되면 더 이상 아래의 명령문들을 실행하지 않고, 반복의 다음 처리로 이동
- for 문에서 예제

```
int i;
int sum = 0;
for( i = 0; i < 5; i++ )
{
    if( i % 2 == 0 ){
        continue;
    }
    printf("%d", i); // 1, 3 출력
    sum += i;
}
printf("₩n i = %d", i); // i=5 출력
printf("₩n sum= %d", sum); // sum=4 출력
```

- while 문에서 예제

```
int i = 1;
int sum = 0;
while( i < 5 )
{
    i++;
    if( i % 2 == 0 ){
        continue;
    }
    printf("%d", i); // 3, 5 출력
    sum += i;
}
printf("₩n i = %d", i); // i=5 출력
printf("₩n sum= %d", sum); // sum=8 출력
```

(5) break 문

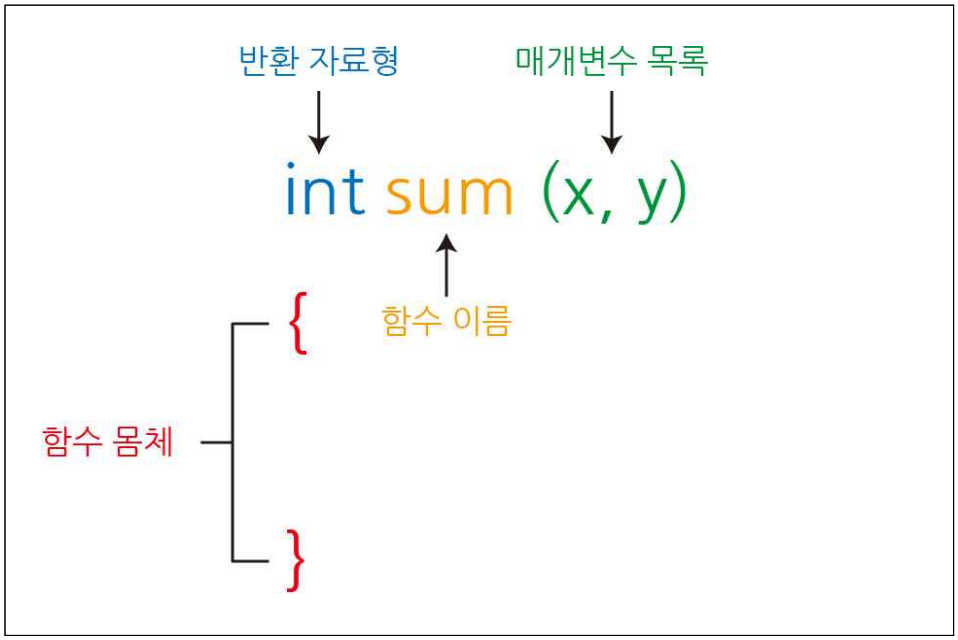
- break 문을 만나게 되면 반복문을 종료하고 그 이후의 명령문을 실행
- 예제

```
int i;
int sum = 0;
for( i = 1; i < 5; i++ )
{
    if( i % 2 == 0 ){
        break;
    }
    printf("%d", i); // 1 출력
    sum += i;
}
printf("₩n i = %d", i); // i=2 출력
printf("₩n sum= %d", sum); // sum=1 출력
```

6. 함수

(1) 함수의 개념

- 하나의 특별한 목적의 작업을 수행하기 위해 독립적으로 설계된 프로그램 코드의 집합
- 문법



- 예제

```
int add(int x, int y)
{
    int sum = x+y;
    return sum;
}

int main(void)
{
    printf("%d", add(5, 3)); // 8 출력
    printf("%d", add(2, 3)); // 5 출력
}
```

⑵ 변수의 유효범위

① 지역 변수(local variable)

- 블록 내에서 선언되고, 블록이 종료되면 메모리에서 사라짐
- 메모리상의 스택(stack)영역에 저장되며, 초기화 하지 않으면 쓰레기값으로 초기화 됨
- 예제

```
int main(void)
{
    int i = 10;
    int sum = 0;
    if( i >= 10 )
    {
        int sum = 0;
        sum+=i;
        printf("%d", sum); // 10 출력
    }
    printf("%d", sum); // 0 출력
}
```

② 전역 변수(global variable)

- 프로그램의 어디에서나 접근할 수 있도록, 함수의 외부에 선언된 변수
- 프로그램이 종료되면 메모리에서 사라짐
- 메모리 상의 데이터(data)영역에 저장되며, 초기화 하지 않으면 0으로 초기화 됨
- 예제

```
int cnt;
int add(int x, int y)
{
    cnt+=1;
    return x+y;
}
int main(void)
{
    printf("%d₩n", add(3, 5) );  // 8 출력
    printf("%d₩n", add(2, 3) );  // 5 출력
    printf("%d₩n", add(8, 3) );  // 11 출력
    printf("%d₩n", cnt );  // 3 출력
}
```

③ 정적 변수(static variable)

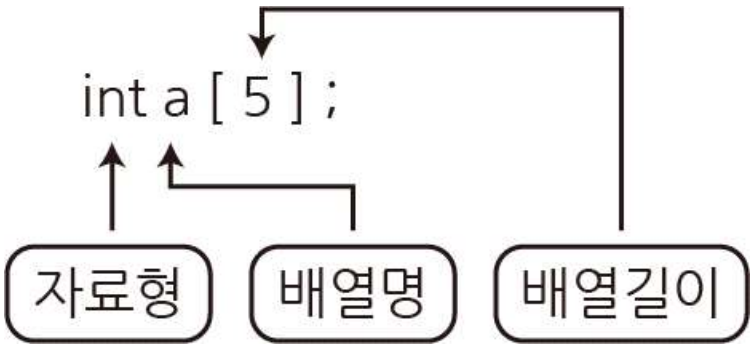
- static 키워드로 선언한 변수
- 단 한번만 초기화 되고, 프로그램이 종료되면 메모리에서 사라짐
- 지역 변수와 전역 변수의 특징을 모두 가진다.
- 예제

```
static int cnt;
int add(int x, int y)
{
    static int cnt = 0;
    cnt+=1;
    printf("%d₩n", cnt);  // 1, 2, 3 출력
    return x+y;
}
int main(void)
{
    printf("%d₩n", add(3, 5) );  // 8 출력
    printf("%d₩n", add(2, 3) );  // 5 출력
    printf("%d₩n", add(8, 3) );  // 11 출력
}
```

7. 배열과 포인터

(1) 1차원 배열

- 1차원 배열의 개념
  - 같은 자료형의 변수를 연속적으로 묶어 놓은 저장공간
  - 메모리상의 물리적 위치에도 연속적으로 저장됨
- 배열의 선언



- 예제

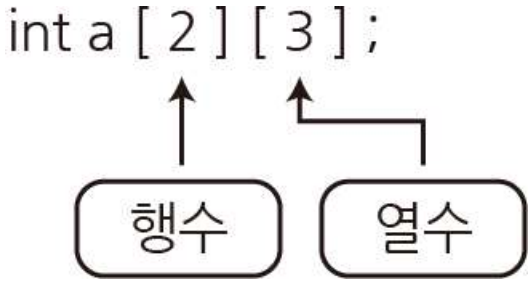
```
int i;
int sum;
int jumsu[3];

jumsu[0] = 80;
jumsu[1] = 75;
jumsu[2] = 90;

for( i = 0; i < sizeof(jumsu) / sizeof(int); i++ )
{
    sum+= jumsu[i];
}

printf("%d", sum);  // 245 출력
```

⑵ 2차원 배열

- 2차원 배열의 개념

 - 같은 자료형의 변수를 행과 열의 연속적인 공간으로 묶어 놓은 저장 공간

- 배열의 선언



a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]

- 예제

```
int arr[2][3];

for( int i = 0; i < 2; i++ )
{
    for( int j = 0; j < 3; j++ )
    {
        arr[i][j] = i * j;
    }
}

for( int i = 0; i < 2; i++ )
{
    for( int j = 0; j < 3; j++ )
    {
        printf("%d₩t", arr[i][j]);
    }
    printf("₩n");
}
/* 출력 값 */
0    0    0
0    1    2
```

⑶ 포인터

- 포인터의 개념
  - 메모리의 주소값을 저장하는 변수, 포인터 변수라고도 한다.
- 포인터 관련 연산자
  - & (주소 연산자) : 변수의 주소값을 알 수 있는 연산자
  - * (참조 연산자) : 변수의 주소값을 알 수 있는 연산자
- 단순 포인터 예제

```
int num = 3;
int *ptr = &num;
printf("%d", num);  // num 변수에 대입된 3 출력
printf("%d", ptr);  // num 변수의 주소값을 출력
printf("%d", *ptr); // ptr이 가리키고 있는 주소인 num 변수의 값 출력
```

8. 구조체

⑴ 구조체

- 구조체의 개념
  - 여러 변수들을 모아서, 하나의 객체를 구성할 때 사용하는 사용자 정의 타입 객체
- 구조체의 구조

```
struct 구조체명
{
    멤버변수1;
    멤버변수2;
};
```

- 기본 사용법

```
struct person
{
    char *name;
    int age;
};
struct person user1;
user1.name = "hungjik";
user1.age = 43;
printf("%s", user1.name);
printf("%d", user1.age);
```

- 구조체 정의와 함께 사용

```
struct person
{
    char *name;
    int age;
} test;
test.name = “hungjik”;
test.age = 43;
printf(“%s”, test.name);
printf(“%d”, test.age);
```

9. C언어 표준 라이브러리

헤더파일	기능	관련 함수
stdio.h	표준 입출력	printf(), scanf(), getchar(), fopen(), fseek(), 등
string.h	메모리와 문자열의 처리	strcat(), strcmp(), strcpy(), strlen(), 등
ctype.h	문자 검사 및 변환	isalnum(), isalpha(), isgraph(), tolower(), 등
math.h	삼각함수, 지수 절대값 함수 등 수학 함수	sin(), cos(), tan(), log(), exp(), fmod(), sqrt(), abs(), 등
stdlib.h	메모리 동적 할당, 가상 난수 발생, 문자열 변환	malloc(),free(), rand(), strtod(), atof(), 등
malloc.h	메모리 할당에 관한 함수	calloc(), malloc(), free(), 등
time.h	날짜와 시간 그리고 내부 클락	clock(), ctime(), asctime(), ftime(), time(), getdata(), 등

Section 3. JAVA 언어

1. JAVA 언어

⑴ JAVA의 역사

- 자바는 1980년대 초, 미국 Sun 사의 Jmmes Gosling이 가전 제품에 이용할 목적으로 개발
- 인터넷이 확산되면서 90년대부터 관심을 받기 시작함

⑵ JAVA의 특징

- 이식성이 높은 언어
- 객체 지향 언어
- 메모리를 자동으로 관리
- 다양한 어플리케이션 개발
- 멀티쓰레드 구현
- 동적로딩 지원
- 오픈 소스 라이브러리가 풍부

2. JAVA 기본 구성

⑴ 클래스(Class)

- 객체가 가지고 있는 속성과 연산의 구조를 정의한다.
- 속성은 멤버변수로, 연산은 메서드로 정의한다.

⑵ 멤버 변수(Member Variable)

- 객체의 속성을 정의한다.
- 해당 객체가 가지고 있는 고유한 값이다.

⑶ 메서드(Method)

- 특정 작업을 수행하기 위한 명령문의 집합
- 멤버변수의 값이나, 상태를 변경할 수 있는 명령의 집합

⑷ 접근지정자

- 클래스의 멤버 변수와, 메서드를 외부에서 접근할 수 있는 범위를 지정한다.
- 종류

종류	접근범위	클래스	패키지	상속	전체
public	접근 제한 없음	O	O	O	O
protected	동일 패키지와 상속 받은 클래스	O	O	O	
default	동일 패키지	O	O		
private	동일 클래스	O			

(5) 인스턴스

- 클래스를 통해서 실제로 구현된 구체적인 실체
- 실제로 메모리에 할당된 상태

(6) 객체 정의와 생성

- 현실 세계를 객체로 표현한다.
- 예제

```
public class Person{
    String name = "홍길동";
    int age = 40;

    public void setName(String n)
    {
        name = n;
    }
    public void setAge(int i){
        age = I;
    }

    public static void main(String[] args){
        Person p1 = new Person();  // p1 이라는 인스턴스 생성

        p1.set_name("홍길동");
        p1.set_age(40);
    }
}
```

3. 객체지향 특징

(1) 생성자

- 인스턴스가 만들어질 때, 초기값 등을 지정하는 역할을 한다.
- 자바에서 생성자는 해당 클래스의 이름과 같아야 한다.
- 생성자는 인자를 다르게 하여 여러 개를 가질 수 있다.
- 예제

```
public class Person {
    String name;
    int age;

    public Person(){
        this.name = “사람”;
        this.age = 1;
    }

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void toPrint(){
        System.out.println( this.name + “님의 나이는 “ + this.age + “살 입니다.” );
    }

    public static void main(String[] args)
    {
        Person p1 = new Person();
        p1.toPrint();  // 사람님의 나이는 1살입니다.

        Person p2 = new Person(“이흥직”, 40);
        p2.toPrint();  // 이흥직님의 나이는 40살입니다.
    }
}
```

⑵ 예외처리(Exception Handling)

- 프로그램 실행 중에 발생하는 예외를 처리해주는 방법
- 예외 상황이 발생했을 때 프로그램이 자연스럽게 종료 될 수 있도록 처리한다.
- 예외처리 사용

```
 public static void main(String[] args) {
        try{
            코드실행
        }
        catch(ArithmeticException e){
            산술계산 오류발생시 실행
        }
        catch(Exception e){
            모든 오류 발생시 실행
        }
        finally{
            모든 오류처리가 종료된 후 실행
        }

    }
}
```

⑶ 상속(Inheritance)

- 부모클래스의 멤버를 자식클래스에서 그대로 쓸 수 있다.
- 클래스를 재사용할 수 있기 때문에 효율적이고, 개발기간을 단축시킨다.
- 부모의 private 한 멤버는 상속 받을 수 없다.
- extends 라는 키워드를 사용한다.
- 예제

```
class Parent {
    String name;
    int age;
    public void set_name(String param_n){
        name = param_n;
    }
    public void set_age(int param_i){
        age = param_i;
    }
}
class Child extends Parent{
    int height;
    public void set_height(int param_h){
        height = param_h;
    }

    public static void main(String[] args){
        Child c = new Child();
        c.set_name("홍길동");
        c.set_age(40);
        c.set_height(170);

        System.out.print(c.name);   // 홍길동
        System.out.print(c.age); // 40
        System.out.print(c.height);    // 170
    }
}
```

⑷ 메서드 오버라이딩(Method Overriding)

- 상속 관계에서 부모 클래스에서 정의된 메서드를 다시 재정의 한다.
- 추상 클래스나 인터페이스를 상속 받을 때 사용되는 개념
- 부모 메서드의 이름, 리턴 타입, 매개변수의 개수와 유형이 완전히 동일해야 한다.
- 예제

```
class Parent {
    public void set_name(String param_n){
        System.out.println("부모의 이름 변경");
    }
    public void set_age(int param_i){
         System.out.println("부모의 나이 변경");
    }
}

class Child extends Parent{
    public void set_name(String param_n){
        System.out.println("자식의 이름 변경");
    }
    public void set_height(int param_h){
        System.out.println("자식의 키 변경");
    }
    public static void main(String[] args){
        Child c = new Child();
        c.set_name("홍길동");  // 자식의 이름 변경
        c.set_age(40);  // 부모의 나이 변경
        c.set_height(170);  // 자식의 키 변경
    }
}
```

⑸ 메서드 오버로딩(Method Overloading)

- 같은 이름의 메서드를 인자만 다르게 하여 중복 정의 하는 것이다.
- 메서드의 이름이 같아야하고, 인자의 개수나 타입이 틀려야 한다.
- 예제

```
class Person {
    String name;
    int age;
    int height
    public void set_data(String p_name){
        name = p_name;
    }
    public void set_data(String p_name, int p_age){
        name = p_name;
        age = p_age;
    }
    public static void main(String[] args){
        Person p1 = new Person();
        p1.set_data("홍길동");

        Person p2 = new Person();
        p2.set_data("홍길동", 40);
    }
}
```

4. 추상클래스와 인터페이스

(1) 추상클래스(abstract class)

- 반드시 오버라이딩 해서 사용할 미완성의 메서드를 하나 이상 가진 미완성 클래스
- 추상클래스를 상속받은 자식 클래스에서 추상 메서드를 구현해서 사용한다.
- 추상 클래스는 객체를 생성할 수가 없다.
- 예제

```
abstract class Remote{
    public int volume = 10;
    public int channel = 1;
    public void volume_up(){
        this.volume++;
    }
    abstract void channel_change(int i);
}
class TV_Remote extends Remote{
    void channel_change(int i){
        channel = i;
    }
    public static void main(String[] args){
        TV_Remote tv = new TV_Remote();
        tv.volume_up();
        tv.volume_up();
        System.out.println( "볼륨 : " + tv.volume ); // 볼륨 : 12
        tv.channel_change(5);
        System.out.println( "채널 : " + tv.channel ); // 채널 : 5
    }
}
```

⑵ 인터페이스(interface)

- 추상클래스의 극단으로, 모든 메서드가 추상적인 형태이다.
- 예제

```
interface Remote{
    public void volume_up();
    public void channel_change(int i);
}
class TV_Remote implements Remote{
    public int volume = 10;
    public int channel = 1;
    public void channel_change(int i){
        this.channel = i;
    }
    public void volume_up(){
        this.volume++;
    }
    public static void main(String[] args){
        TV_Remote tv = new TV_Remote();
        tv.volume_up();
        tv.volume_up();
        System.out.println( "볼륨 : " + tv.volume ); // 볼륨 : 12

        tv.channel_change(5);
        System.out.println( "채널 : " + tv.channel ); // 채널 : 5
    }
}
```

Section 4. 파이썬

1. 파이썬(Python)

(1) 파이썬 소개

- 네덜란드 출신의 프로그래머인 귀도 반 로섬(Guido van Rossum)이 1989년에 개발한 프로그래밍 언어
- 구글에서 만들어진 소프트웨어의 50% 이상이 파이썬으로 만들어졌다고 함

(2) 파이썬의 특징

- 컴파일 과정이 필요없는 스크립트 언어이다.
- 동적 타이핑(Dynamic typing)을 지원한다.
- 플랫폼 독립적이다.
- 간결하고 쉬운 문법으로 되어 있다.
- 높은 확장성 및 이식성이 좋다.
- 수많은 표준 라이브러리가 존재한다.

2. 파이썬 문법

(1) 파이썬 기초 문법

① 사칙연산

```
print( 2 + 5 )  # 7
print( 3 / 2 ) # 1
print( 3 / 2.0 ) # 1.5
print( 3 * 4 ) # 12
```

② 변수 사용

```
a = 3
b = 2.0
print( a / b ) # 1.5
```

③ 조건문

```
a = 10
if a > 10 :
    print("a가 10보다 크다")
elif a > 5 :
    print("a가 5보다 크다")
else :
    print("a가 5보다 작다")

출력 : a가 5보다 크다
```

④ for 반복문

```
for i in [1, 2, 3]:
    print( i )

출력 :
1
2
3
```

⑤ while 반복문

```
i = 0
sum = 0
while i < 3:
     sum = sum+i;
     i=i+1
print(sum) # 3출력
```

⑥ range

```
print( range(3) ) # [0, 1, 2]
print( range(3, 6) ) # [3, 4, 5]
print( range(3, 15, 3) ) # [3, 6, 9, 12]

int_val = range( 2, 10, 2 )
for i = in int_val:
    print(i)   # 2   4   6   8
```

⑦ 함수

```
def sum( i, j ):
     return i + j

print( sum(10, 2) ) # 12 출력
```

(2) 문자열

① 문자열 사용

```
str = 'hung'
str1 = 'jik'
print( str + str1 ) # hungjik 출력
```

② 문자열 함수

```
# join : 여러 개의 문자열을 하나로 결합
str = '^'.join(['a','b','c'])
print(str) # a^b^c

str = ''.join(['a','b','c'])
print(str) # abc

# partition : 인자값을 기준으로 첫 번째 조건이 발견되면, 분리하여 반환한다.
str = "hungjik_wonju_kangwon"
arr = str.partition('_');
print(arr) # ('hungjik', '_', 'wonju_kangwon')

# split : 인자값을 기준으로 분리하여 리스트로 반환
str = "hungjik_wonju_kangwon"
arr = str.split('_');
print(arr) # ['hungjik', 'wonju', 'kangwon']

# format : %연산자와 포캣 스트링을 사용하는 방법
print("Int : %d, Str : %s, float : %f" % (43, "hungjik", 168.5) )
# Int : 43, Str : hungjik, float : 168.500000

# len : 문자열 길이를 구해준다.
str = "hungjik"
print( len(str) ) # 7

# find : 문자열의 첫 시작 위치를 구해준다.
str = "hungjikhungjik"
print( str.find('g') ) # 3

# upper : 모든 문자를 대문자로 변경한다.
str = "hungjik"
print( str.upper() ) # HUNGJIK

# lower : 모든 문자를 소문자로 변경한다.
str = "HUNGJIK"
print( str.lower() ) # hungjik

# lstrip, rstrip, strip : 좌/우 공백을 제거한다.
str = "   hungjik   "
print( str.strip() ) # hungjik
```

3. 자료구조

(1) 리스트 (List)

- 배열과 같이 여러 요소들을 갖는 집합으로 새로운 요소를 추가하거나, 갱신, 삭제 하는 일이 가능하다.
- 파이썬의 리스트는 동적배열로서 자유롭게 확장할 수 있는 구조를 갖는다.
- 리스트의 요소들은 Square bracket([ ])으로 둘러쌓여 컬렉션을 표현한다.
- 각 요소들은 서로 다른 타입이 될 수 있으며, 컴마(,)로 구분한다.
- 리스트 선언

```
list = []
list = [“hungjik”, 43, 168.5]
print(list)  # ['hungjik', 43, 168.5]

print( list[1] ) # 43

list[1] = “test”
print(list) # ['hungjik', 'test', 168.5]

print( list * 3 )
# ['hungjik', 'test', 168.5, 'hungjik', 'test', 168.5, 'hungjik', 'test', 168.5]
```

- 리스트 슬라이스

```
list = [ 1, 2, 3, 4, 5, 6, 7, 8 ]
print( list[1:3] ) # [2, 3]
print( list[:3] ) # [1, 2, 3]
print( list[3:] ) # [4, 5, 6, 7, 8]
print( list[1::2] ) # [2, 4, 6, 8]
```

- 리스트 추가, 수정, 삭제

```
list = [ 1, 2, 3, 4, 5, 6, 7, 8 ]
list.append(9) # 추가
list[2] = 10 # 수정
del list[0]
print(list) # [2, 10, 4, 5, 6, 7, 8, 9]
```

- 리스트 병합

```
list1 = [ 1, 2, 3 ]
list2 = [ ‘a’, ‘b’, ‘c’ ]
tot = list1 + list2
print( tot ) # [1, 2, 3, 'a', 'b', 'c']
```

- 리스트 검색

```
list = [ 1, 2, 3, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4 ]
print( list.index(2) ) # 1
print( list.count(2) ) # 3
```

⑵ 튜플 (Tuple)

- 리스트와 비슷하게 여러 요소들을 갖는 구조이다.
- 리스트와 다른 점은 새로운 요소의 추가, 수정, 삭제가 불가능하다.
- 튜플의 요소들은 괄호를 사용한다.
- 튜플 선언

```
tup = (“hungjik”, 43, 168.5)
print( tup ) # ('hungjik', 43, 168.5)
```

⑶ Set

- 중복이 없는 요소들로만 구성된 구조이다.
- Set 선언

```
set_data = { 1, 2, 3, 1, 2, 3, 4 }
print(set_data) #set([1,2,3,4])
```

⑷ 딕셔너리 (dict)

- 키(Key)-값(Value) 쌍으로 되어 있는 구조이다.
- 키 값으로 값을 찾을 수 있는 해시테이블 구조를 갖는다.
- 딕셔너리 선언

```
dict_data = {“흥직”:80, “경직”:75, “창훈”:90}
print( dict_data ) # {“흥직”:80, “경직”:75, “창훈”:90}
print( dict_data[“흥직”] ) # 80

dict_data[“흥직”] = 88
print( dict_data[“흥직”] ) # 88
```

- 딕셔너리 메서드

```
dict_data = {“흥직”:80, “경직”:75, “창훈”:90}

# 키만 추출하기
keys = dict_data.keys();
print( keys ) # [‘흥직’, ‘경직’, ‘창훈’]
for in in keys:
    print(in)

# 값만 추출하기
values = dict_data.values();
print( values ) # [75, 90, 80]
for in in values:
    print(in)

# 키가 있는지 체크하여 가져오기
print( dict_data.get(“흥직”) ) #80
```