프로그래밍 언어 : 컴퓨터 시스템을 동작시키기 위한 프로그램 작성 언어
명령어들의 조합. 비트 (0/1)의 값으로 작성되거나 변환되어 컴퓨터가 이해할 수 있도록 한다.

● 비트(bit)
컴퓨터를 이해하기 위한 가장 기본적인 용어로 Binary Digit의 약칭
이진법을 이용, 1바이트는 8비트

●명령어 형식
연산코드(4비트)+오퍼랜드1(4비트)+오퍼랜드2(8비트)
(1)연산코드: 수행될 연산을 지정(load, add 등)
(2)오퍼랜드: 연산을 수행하는 데 필요한 데이터 또는 데이터의 주소

프로그램의 구성요소: 자료구조와 알고리즘
●자료구조
컴퓨터에 데이터를 삽입, 삭제, 수정하게 해 주는 논리적인 공간구조 의미

자료의 형태에 따른 자료 구조 분류
단순구조: 프로그래밍 언어에서 제공하는 기본 데이터 타입(int, float, double, char 등)
선형구조:자료들 사이의 선후 관계가 일대일인 구조이다.(연결자료구조, 순차자료구조, 스택, 큐, 덱 등)
비선형구조: 자료들 사이의 선후 관계가 계층 또는 그물 형태를 가지는 구조(트리, 그래프 등)
파일구조:보조 기억 장치에 데이터 값이 실제로 기록되는 자료구조(순차파일, 색인파일 등)

●알고리즘
넓은 의미에서 자료 구조와 함께 프로그램을 구성하는 요소
좁은 의미에서는 어떤 문제에 대한 답을 찾는 해법

알고리즘의 5가지 특성
입력: 외부로부터 입력되는 자료가 0개 이상이어야 한다.
출력: 출력되는 결과가 1개 이상이어야 한다.
명확성: 각 명령어의 의미가 명확하여야 한다.
유한성: 정해진 단계를 지나면 종료되어야 한다.
유효성: 모든 명령은 실행이 가능한 연산들이어야 한다.

●변수
어떤 값을 주기억 장치에 기억하기 위해서 사용하는 공간

●식별자

프로그램의 구성 요소를 구별하기 위한 기준으로 변수명이 식별자에 속한다.

●바인딩

변수와 변수에 관련된 속성을 연결하는 과정으로,정적바인딩과 동적 바인딩으로 구분된다.

(1)정적 바인딩: 프로그램 실행 시간 전에 속성을 연결하는 방식

(2)동적 바인딩: 프로그램 실행시간에 속성을 연결하는 방식

●선언

변수에 이름, 데이터타입 등의 속성을 부여하는 작업으로, 명시적 선언과 묵시적 선언으로 구분

(1)명시적 선언: 선언문을 이용하여 변수 이름을 나열하고 속성을 부여하는 방식

(2)묵시적 선언: 별도의 선언문 없이 디폴트 규칙에 의해 속성이 부여되는 방식

● 할당: 변수에 메모리 공간을 바인딩 하는 작업

●데이터 타입의 유형

불린타입: 조건이 참인지 거짓인지 판단하고자 할 때 사용

문자타입: 문자 하나를 저장하고자 할 때 사용

문자열 타입: 나열된 여러개의 문자를 저장하고자 할때 사용

정수 타입: 정숫값을 저장하고자 할 때 사용

부동 소수점 타입: 소수점을 포함하는 실숫값을 저장하고자 할 때 사용

배열 타입: 여러 데이터를 하나로 묶어서 저장하고자 할 때 사용

●Java와 C/C++의 정수타입

Java

byte : 1byte : -128~127

short:2byte : -32,768~32,767

int:4byte : -2,147,483,648~2,149,483,647

long:8byte:

C/C++

short: 2byte:-32,768~32,767

unsigned short : 2byte

int : 2byte(or 4byte)

unsigned int: 2byte(or 4byte)

long: 4byte

unsigned long(4byte)

1-2 . 연산자와 명령문 활용

● 산술연산자

+ : 양쪽의 값을 더한다

- 왼쪽에서 오른쪽 값을 뺀다

* 두개의 값을 곱한다

/: 왼쪽값을 오른쪽 값으로 나눈다

%: 왼쪽값을 오른쪽 값으로 나눈 나머지를 계산한다.

● 시프트 연산자

비트를 이동시키는 연산자.

<<: 왼쪽값을 오른쪽 값만큼 비트를 왼쪽으로 이동시킨다.

>>:왼쪽값에 오른쪽 값만큼의 부호 비트를 채우면서 왼쪽으로 이동시킨다.

●관계연산자

> :왼쪽에 있는 값이 오른쪽에 있는 값보다 크면 참을 반환, 그렇지 않으면 거짓을 반환

<: 왼쪽에 있는 값이 오른쪽에 있는 값보다 작으면 참, 그렇지 않으면 거짓을 반환

>=:왼쪽에 있는 값이 오른쪽에 있는 값보다 크거나 작으면 참, 아니면 거짓

<=:왼쪽에 있는 값이 오른쪽에 있는 값보다 작으면 참, 아니면 거짓

==: 양쪽 값이 같으면 참

!=: 양쪽값이 다르면 참

● 논리연산자

두 연산자 사이의 논리적인 관계를 정의하는 연산자.

&& : 두개의 논릿값이 모두 참이면 참, 그렇지 않으면 거짓

|| : 두개의 논릿값 중 하나가 참이면 참, 아니면 거짓

●비트 연산자

비트 연산자는 0과 1의 각 자리에 대한 연산을 수행하며, 0 또는 1의 결괏값을 가진다.

&: 두 값을 비트로 연산하여 모두 참이면 참, 그렇지 않으면 거짓

| : 두 값을 비트로 연산하여 하나가 참이면 참, 그렇지 않으면 거짓

^: 두 값을 비트로 연산하여 서로 다르면 참, 그렇지 않으면 거짓

●조건문

if 문 : 조건이 참인지 거짓인지에 따라 경로 선택
case/switch문: 조건에 따라 여러 개의 선택 경로 중 하나를 취하고자 할 때 사용하며, 프로그래밍 언어에 따라 case또는 switch를 명령문으로 사용

●반복문
while문: 수식이 거짓이 될 때 까지 해당 문장을 반복해서 실행한다.
for문: 시작과 종료 조건을 지정하여 참인 동안에는 해당 문장을 반복해서 실행한다.
do문: 초깃값, 최종값, 증감값을 지정하여 반복을 실행한다.

● 사용자 정의 자료형의 개요
C/C++, Java와 같은 프로그래밍 언어에서는 사용자가 직접 자료형을 만드는 것이 가능.
C++에서는 열거체, 구조체, 공용체로 구분하여 작성 가능

● 열거체: 괄호 안에 연속적인 값이 들어가는 자료형, 특정 값을 넣어주지 않으면 1씩 늘어나기 때문에 상수 배열이라고 부르기도 한다.
● 구조체: 괄호 안에 멤버 변수를 사용하는 자료형. 내부에 멤버 변수 자료형을 마음대로 선언할 수 있어며 멤버 함수의 작성도 가능.
● 공용체: 구조체와 거의 유사하나 조금 더 범주가 크다. 열거체나 구조체와 달리 공용체명을 작성하여야 하고, 구조체를 멤버로 사용할수 있다.

●추상화(=일반화)
복잡한 문제의 본질을 이해하기 위해 세부 사항은 배제하고 중요한 부분을 중심으로 간략화 하는 기법
(기능 추상화 / 자료 추상화/ 제어추상화)

●기능 추상화: 입력 자료를 출력 자료로 변환하는 과정을 추상화 하는 방법.
●자료 추상화: 자료와 자료에 적용할 수 있는 오퍼레이션을 함께 정의하는 방법.
●제어 추상화: 외부 이벤트에 대한 반응을 추상화 하는 방법.

●상속: 상위 수준 그룹의 모든 특성을 하위 수준 그룹이 이어받아 재사용 또는 확장
●구체화: 하위 수준 그룹이 상위 수준 그룹의 추상적인 부분을 구체화

●프로그래밍 언어의 유형 분류
프로그래밍언어: 개발 편의성에 따라 저급언어와 고급언어
실행하는 방식에 따라 명령형/함수형/논리형/객체 지향형 언어
구현 기법에 따라 컴파일 방식의 언어, 인터프리터 방식의 언어, 혼합형 언어로 분류

1. 개발 편의성에 따른 분류

(1) 저급언어: 기계가 이해할 수 있도록 만들어진 기계어, 어셈블리어

(2) 고급언어: 개발자가 소스코드를 작성할 때 쉽게 이해할 수 있도록 작성된 언어로 C, C++, Java등이 속함.

2. 실행하는 방식에 따른 분류

(1)명령형 언어(=절차형언어): 컴퓨터에 저장된 명령어들이 순차적으로 실행되는 프로그래밍 방식으로 절차형 언어라고도 불린다.

FORTRAN, COBOL, PASCAL, C등

(2) 함수형 언어: 수학적 수식과 같은 함수들로 프로그램을 구성하여 호출하는 방식으로 LISP등의 프로그래밍 언어가 함수형 언어에 속한다.

(3) 논리형 언어: 규칙에 대한 활성화 조건이 만족되면 연관된 규칙이 실행되는 구조.

추론과 관계 규칙에 의해 원하는 결과를 얻어내는 방식. PROLOG등이 논리형 언어에 속한다.

(4) 객체 지향 언어: 객체 간의 메시지 통신을 이용하여 프로그래밍 하는 방식으로, JAVA와 C++등이 객체 지향 언어에 속한다.

3. 구현 기법에 따른 분류

(1) 컴파일 방식의 언어: 고급 언어를 기계어로 번역하는 방식의 언어.

FORTRAN, PASCAL, C, C++

컴파일 방식은 실행에 필요한 정보가 컴파일 시간에 계산되어 실행속도가 빠르다.

(2) 인터프리터 방식의 언어: 고급언어 명령문을 하나씩 번역하고 실행하는 방식

BASIC, PROLOG, LISP, SNOBOL등이 인터프리터 방식의 언어에 속한다. 프로그램 실행시에 계산된다.

(3) 혼합형 방식의 언어: 고급 언어를 컴파일 하여 중간 언어로 변환한후, 인터프리터에 의해 번역을 실행하는 방식의 언어를 의미, JAVA

● 컴파일러: FORTRAN, C등과 같은 고급 언어를 기계어로 번역하는 도구. 컴파일 방식의 언어는 모두 컴파일러를 필요로 한다. 컴파일러는 프로그램의 한종류.

● 인터프리터: 프로그램 문장을 하나씩 번역하고 실행할 수 있도록 하는 프로그램. 컴파일 과정이 없기 때문에 개발하는 과정에서 사용하면 유용. 인터프리터 방식의 언어는 모두 인터프리터를 필요로 한다. 실행속도가 느리고 메모리 사용이 비효율적.

●절차지향 프로그래밍

객체라는 개념이 등장하기 이전에 **모듈, 변수, 함수**를 사용하여 개발하는 방법.

프로그램을 순차적으로 수행시키는 방법으로 자료구조와 명령 중심으로 프로그램을 구성한다.

●객체 지향 프로그래밍

객체와 객체간의 통신(=메시지)를 통해 프로그램 구현

●**객체지향프로그래밍** 구성요소

-**객체(Object)** 개체, 속성, 메소드로 구성된 클래스의 인스턴스 의미

-**클래스(Class)** 객체의 타입을 정의하고 객체를 생성하는 틀

-**메시지(Message)**객체 간의 통신

●**객체**의 구성요소

개체(Entity) / 속성(Attribute) / 메소드(Method)

개체: 현실 세계에 보이는 본질

속성: 자료 저장소 역할, 절차지향에서는 변수와 대응

메소드: 호출단위 의미, 절차지향프로그래밍 함수와 대응

● 프로그래밍 언의 특성을 활용하여 최적화 수행
1. 프로그래밍 언어가 가지는 패턴을 확인하여 적용
(1) 프로그래밍 언어가 가지는 패턴 파악
(2) 프로그래밍 언어의 특성을 활용하여 패턴을 적용
2. 보안약점 확인 후 최적화
3. 프로그래밍 언어의 특성에 따라 발생할 수 있는 성능, 가용성 문제 확인 후 최적화
4. 불필요하거나 중복된 코드의 제거 등을 통해 개선할 부분은 없는지 파악하여 최적화를 수행한다.

● 프로그래밍 언어에서 제공하는 패턴을 이용하면 효율적이고 재사용성이 높은 프로그램 개발이 가능
●최적화 후 테스트를 수행하는 과정에서는 테스트 자동화 도구를 이용하면 효율적

● 라이브러리 : 필요할 때 찾아서 쓸 수 있도록 모듈화 되어 제공되는 프로그램
구성: 도움말/ / 설치파일 // 샘플코드

● 표준 라이브러리: 프로그래밍 언어가 기본적으로 가지고 있는 라이브러리
● 외부 라이브러리: 별도의 파일을 설치하여야 한다. 누군자 개발하여 설치 가능, 공유 가능

● 라이브러리는 모듈과 패키지를 총칭
모듈: 개별 파일
패키지: 여러 개의 모듈을 한개의 폴더에 묶어서 기능 제공