# <컴퓨팅사고와SW 과제_내용>

구조적 언어 구성요소 설명 및 필요 이유를 서술하라  // 데이터를 받아 정보를 만들기 위한 과정이다. !!!!!!!!!

## 비구조적 언어는 통짜로 만들었다(보통 점프점프 하면서 만들었다-특정 부분으로 건너뛰는 GOTO 문을 사용>>분기(if, then, else)와 반복(do, while, until)과 같이 더 익숙한 구조로 대체했다.) why > 흐름제어문이 없으니까 / 함수라는 단위가 없었다.

## >>>  조그만한 변화가 있어도 전체를 다 바꿔야했었다.

## 이 때문에 구조적 언어( 흐름제어 , 함수)를 만들었다  // 기능별로 모듈화해서 사용>함수

흐름제어 : 프로그램 명령어가 여러줄 있으면 실행 순서가 위에서 아래로 > 단순 했음 > 조건에 따라 선택과 반복 실행

구조적 프로그래밍은 제어흐름의 직접적인 전환에 대해 규칙을 부과한다.

구조적 프로그램은 모듈을 기준으로 나뉜다

---

## ▼ 주석

소스코드가 많아져 길어지면 사람이 이해하기 어려움 때문에 설명글을 적는다

프로그래밍을 하다보면 상황을 설명하거나 작업 내용을 기억하기 위해 소스 파일에 메모를 남겨할때 주석문을 사용한다.

컴파일러가 번역을 하지 않아

- WHY? : (복잡한 소스코드들을 주석으로 구분하여 코드의 가독성을 좋게 하기 위해 )

# ▼ 변수와 상수(메모리 공간)

데이터를 입력받아 메모리에 저장(관리)해야함  이를 변수(프로그램이 실행 하면서 계속 변경 가능) 와 상수(한 번 넣으면 끝) 에 저장을 한다

변수 : 변하는 값, 프로그램이 실행 하면서 계속 바귀는 정보 / 사용자로부터 데이터를 받거나 처리한후 저장하는 저장공간의 의미도 있다  (메모리 기능 [기억공간])

상수 : 한번 넣으면 변하지 않는 값 / 한정된 입력 대상 (숫자 상수 , 문자상수, 문자열 상수로 나뉜다)

- WHY : 저장 하기 위한 방법 /  데이터를 저장하기 위해

# ▼ 연산자

들어온 데이터로 정보를 만들어 내야 한다.  >> 연산(명령)을 하여 HOW??  >>  연산( + , —, * ,  / )

**단순히 값을 계산하는 역할 뿐만 아니라 값과 값을 비교해 참,거짓을 판단하고 참과참 또는 참과 거짓으로 구성된 문장을 일정한 기준으로 연결하기도 한다.**

- WHY : 들어온 데이터로 정보를 만들어내기 위해!  프로그래밍 코드가 논리적으로 움직일 수 있도록 지시할 수 있습니다

# ▼ 흐름 제어 (플로우 컨트롤)

연산자로부터 받은 명령어들이 한 줄로 다 표현 할 수 없음
> 여러 줄로 복잡하게 작성된다.

명렁어들이 여러줄로 복잡하게 쭉 나열 된다 복자방게 . 경우에따라 선택 OR 반복을 하는 흐름제어문이 필요하다

- 프로그램 명령어가 여러줄 있으면 실행 순서가 위에서 아래로
  WHY : 여러 줄로 너무 길어져진걸 선택과 반복으로 제어하기 위해  ,  비구조적 언

어에서 조그만한 변화가 있어도 전체를 다 바꿔야 했다. 이를 보완하기 위해 흐름제어를 사용한다.

프로그램에 명령문이 실행되는 순서를 제어 하기 위해 (플로우 컨트롤 하기 위해)

프로그램 : 컴퓨터 명령어들의 집합 // 명령어들을 실행하는 순서 = 플로우(흐름)

3개로 구성되어있다 ( 1. 위에서 아래로 [순서대로] 한줄씩 2.선택(특정문장 선택할 수있고 안할수 있고) 과 반복 (계속해서 명령문을 재실행 하는 것) -선택과 반복에는 반드시 조건식이 따라온다 (괄호열고 괄호 닫고) 선택과 반복은 조건식을 이용. 조건식의 결과 값은 항상 [참(반복) or 거짓(반복을 멈춤]

---

구조적 언어 : 기능별로 특정 단위를 만들어서 작성을하자 >> 그단위는 먼데? >> 함수 / 각각의 기능 모듈을 함수라는 단위로 작성을 하자 >함수들이 모여 하나의 프로그램을 구성 = 구조적 언어의 특징

구조적 언어에서 프로그램에 작성 단위는 함수 >> 구조적언어는 항상 함수를 맞이한다. 함수한에서 알고리즘을 작성한다.

명령문이 실행 되는 순서를 흐름으로 표현

# ▼ 함수 // 기능별로 특정 단위(함수) 를 만들어 작성하자 -각각의 모듈을 함수라는 단위로 작성 >> 함수들이 모여 하나의 프로그램을 작성

프로그램의 명령문을 하나로 묶어서 사용하는 것이 함수

작업별 그룹으로 나누어 작성한 소스 코드는 관리하기도 편할뿐더러 다른 프로그래머가 보거나 나중에라도 코드의 내용을 쉽게 이해할 수 있다.

(프로그래머가 소스 코드를 작성할 때 어떤 작업 단위 없이 단순 명령문만 주욱 나열한다면 작업하는 그 순간에는 해당 부분만 고민하기 때문에 상관이 없다 .

하지만 일정 시간이 지나 문제가 발생하는 부분을 찾거나 기능을 개선하기 위해 소스 코드를 다시 보게 되면 내용을 파악하기 어려워서 작업의 효율성이 떨어진다)

- WHY : 소스코드가 너무 길어져 모듈화 하여 함수라는 단위로 나눠 쓰기 위해

| 비구조적 언어 | → | 구조적 언어 | → | 객체지향언어 |
|---|---|---|---|---|
| | | | | (구조적 언어를 기반으로 OOP를 만들어짐) |
| All in one | | 흐름제어문(Flow Control)<br>함수(Function) | | 객체(Object) |
| (저급언어) | | (고급언어) | | (고급언어) |
| | | Flow control : 명령문이 실행되는 순서를 제어할 수 있다. | | |
| | | **흐름제어**<br>1. 기본(위에서 아래로)<br>2. 선택(조건에 따라 쓰거나 안쓰거나)<br>3. 반복(재실행) | →(2,3은 조건식 이용, 조건식의 결과값은 True or False) | |
| | | | | |
| 프로그램의 기능에 따라 영역을 나누지 않음 | | 업데이트, 수정, 유지 및 보수가 용이 | | |
| | | ↑ | | |
| → 업데이트, 수정이 어려움 | → | 기능별로 특정 기준을 나누는 것으로 발전 → 함수 = 작성단위 | | |