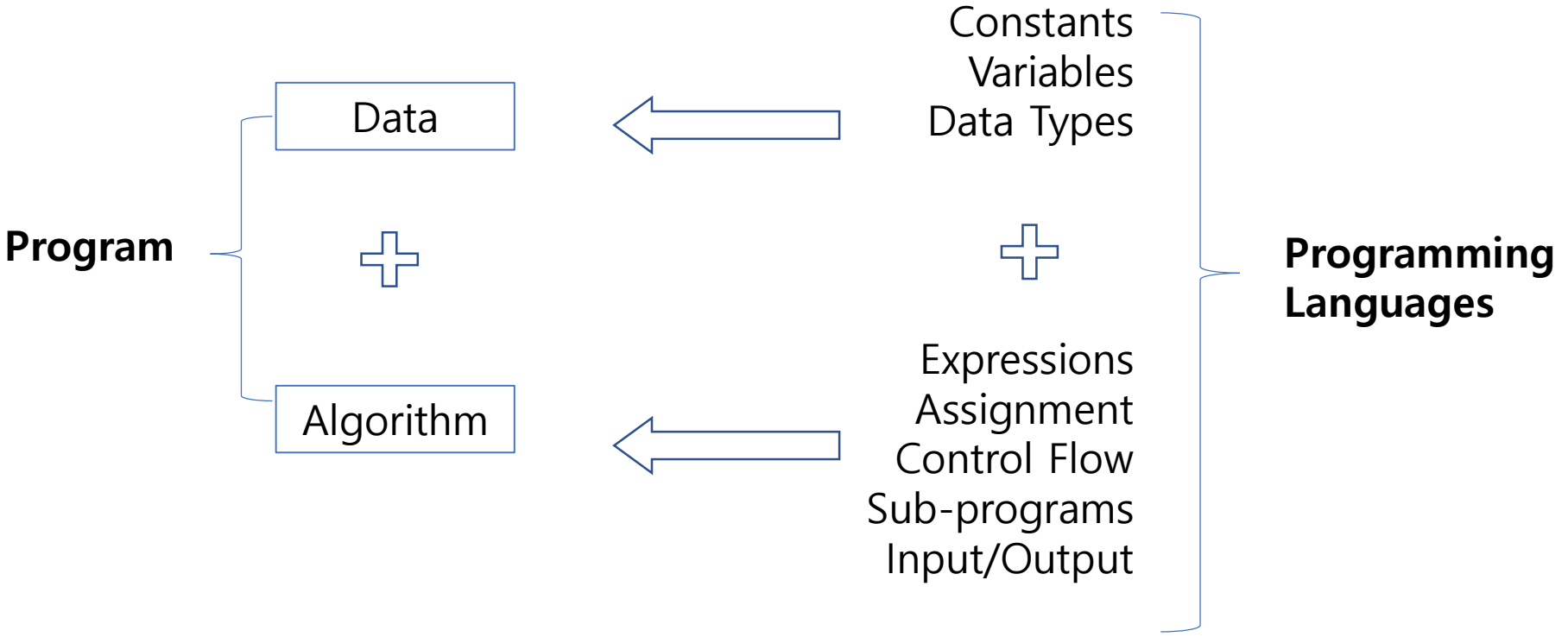
Structured Programming
- Review

Contents

- Programs and Programming Languages
- Evolution of Programming Languages
- Structured Programming
- Program Development Procedure
- Example: Card Game

Programs and Programming Languages

- Program은 주어진 problem을 논리적으로 해결하는 절차를 programming language로 기술한 것
  - Program이 해결하려고 하는 문제는 모두 data와 관련되어 있다.
  - 논리적인 해결절차를 alogrithm이라 부른다.



Evolution of Programming Languages

Years	Programming Styles	Major Languages
50′	High-level Languages Procedural Programming	FORTRAN, COBOL
60′	Establishment of PL Theories	ALGOL, BASIC
70′	**Structured Programming**	C, Pascal
80′	Object-Oriented Programming	C++, Objective-C, Python, Ada
90′	Distributed Programming, Web	Java, PHP, JavaScript, Ruby
00′	Component-based	C# (.NET)
10′	Mobile Programming	Swift

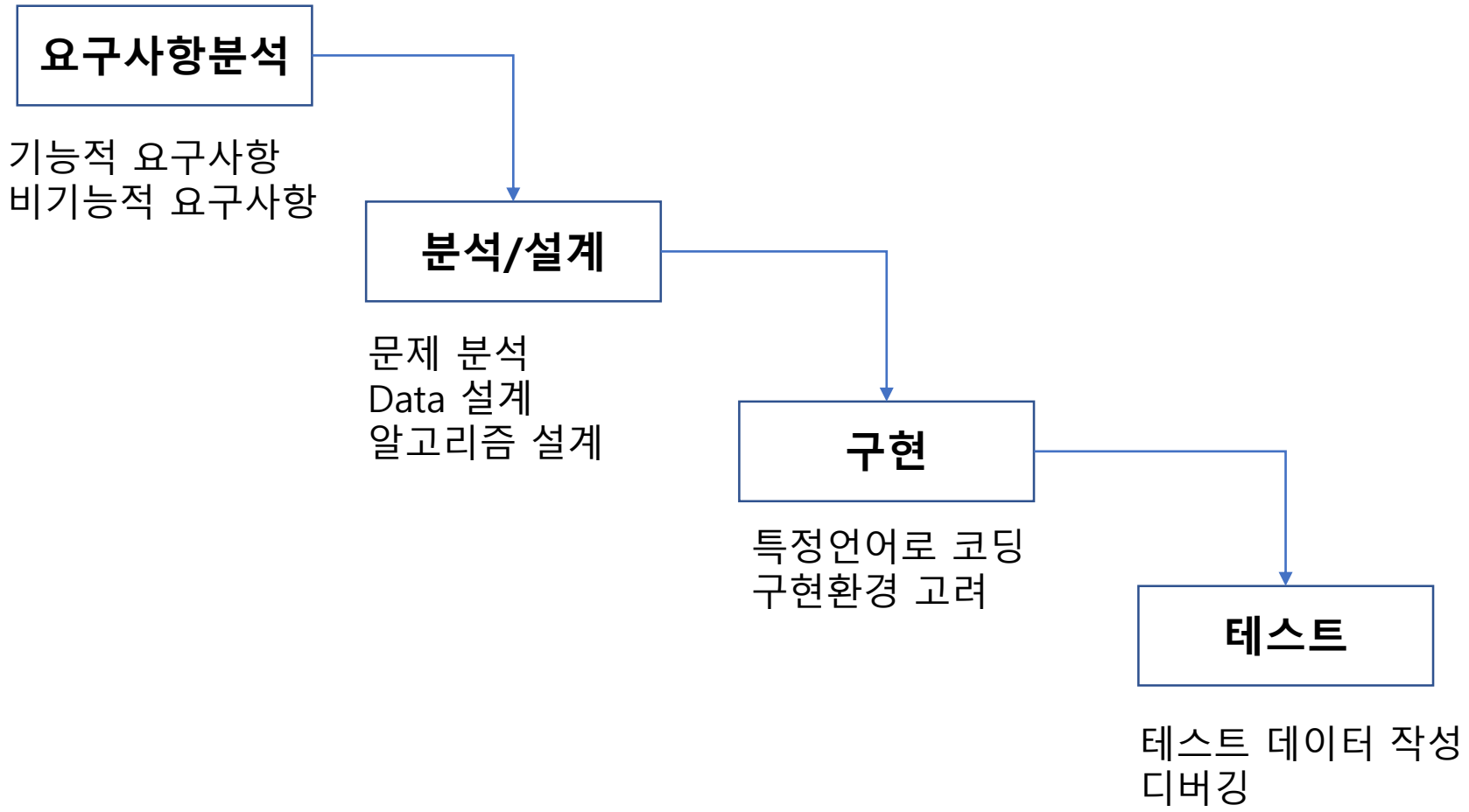
Structured Programming

- Structured Programming Paradigm
  - 프로그램을 top-down 방식으로 큰 모듈을 작은 모듈들로 분할해 나감
  - 모듈의 코드는 구조화된 control flow 문(assignment, if, while)으로 구성함 (goto-free)
- Structured Programming의 특징
  - 모듈들이 데이터를 주로 공유
  - 데이터보다 알고리즘을 더 중시하는 경향이 있음 (Program = Algorithm + Data)
- Structured Programming 의 문제점
  - 모듈의 재사용이 어려움 (특정 모듈은 특정 program을 위해 개발되는 경향)
  - 데이터를 공유함으로써 프로그램 유지보수가 어려움 (데이터의 수정시 전체 프로그램에 영향)
  - 70년대 software crisis 대두됨

⇨ ***Object-Oriented Programming Paradigm의 탄생 배경이 됨***

Program Development Procedure

- 다양한 개발 방법이 있으나 여기서는 가장 간단한 **waterfall** 개발방식을 소개한다.



Example: Card Game

- 문제 정의
  - Poker 카드 52장(Card Deck)을 가지고 두명의 플레이어가 카드 게임을 진행한다.
  - 각 플레이어는 한장씩의 카드를 번갈아 뽑아 비교하여 승자를 결정한다.
  - 게임에서 진 플레이어는 이긴 플레이어에게 일정 금액을 준다.
  - 게임은 카드가 더 이상 없을 때까지 계속해서 진행할 수 있다.
  - 더 이상 카드가 없으면 다시 카드를 모아 계속해서 진행할 수 있다.
  - 게임이 종료되면 두 플레이어의 정보를 출력한다.

기능적인 요구사항 분석

- 요구사항 분석
  - 개발하고자 하는 시스템이 무엇(What)을 해야하는지 관점에서 파악하는 것임 (How가 아님)
  - 고객이 요구하는 것을 정확하게 파악하는 것이 중요 (고객과의 소통)
  - 테이블 등으로 요구사항을 분류하여 명확하게 정리하는 것이 필요
  - 테스트 할 때 기준으로 사용함

- 카드 게임 요구사항 분석
  - 플레이어의 카드 비교는 어떻게 이루어지나?
    - 카드의 숫자가 높으면 이긴다 (단, Ace > King > Queen > Jack > 10의 순이다)
    - 같은 숫자인 경우 문양(suit)에 따라 결정한다. (가령, Spade > Diamond > Heart > Club)
  - 한게임 당 진 플레이어가 이긴 플레이어에게 얼마를 주어야 하나?
    - 값을 지정할 수 있어야 한다
  - 한 플레이어가 가진 잔고(balance)가 게임당 지정 금액보다 적으면 어떻게 하나?
    - 게임을 종료해야한다.

분석 및 설계 : Data 설계

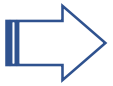
- 데이터 설계 방법
  - 문제 정의나 요구사항 분석 문서에서 핵심이 되는 데이터를 발굴한다.
  - 데이터를 문제에 알맞게 모델링한다 (추상화).
- 데이터의 종류
  - 입력 데이터
  - 결과 데이터
  - 중간결과 데이터
- 카드게임 데이터 분석

주요 데이터

Card

Card Deck

Player



데이터 설계

Card = { suit:Int, value:Int }

CardDeck = deck[52]:Card

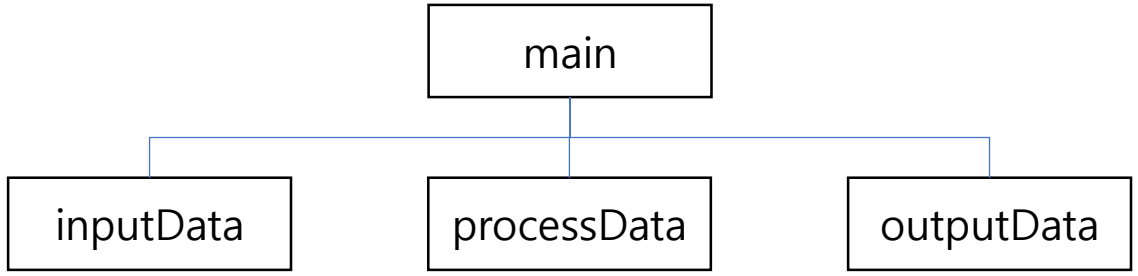
Player = { name:String, balance:Int, hand:Card }

Card와 Player는 구현시 구조체로 구현될 수 있으며
Card Deck은 배열로 구현될 수 있다.

분석 및 설계 : 알고리즘 설계

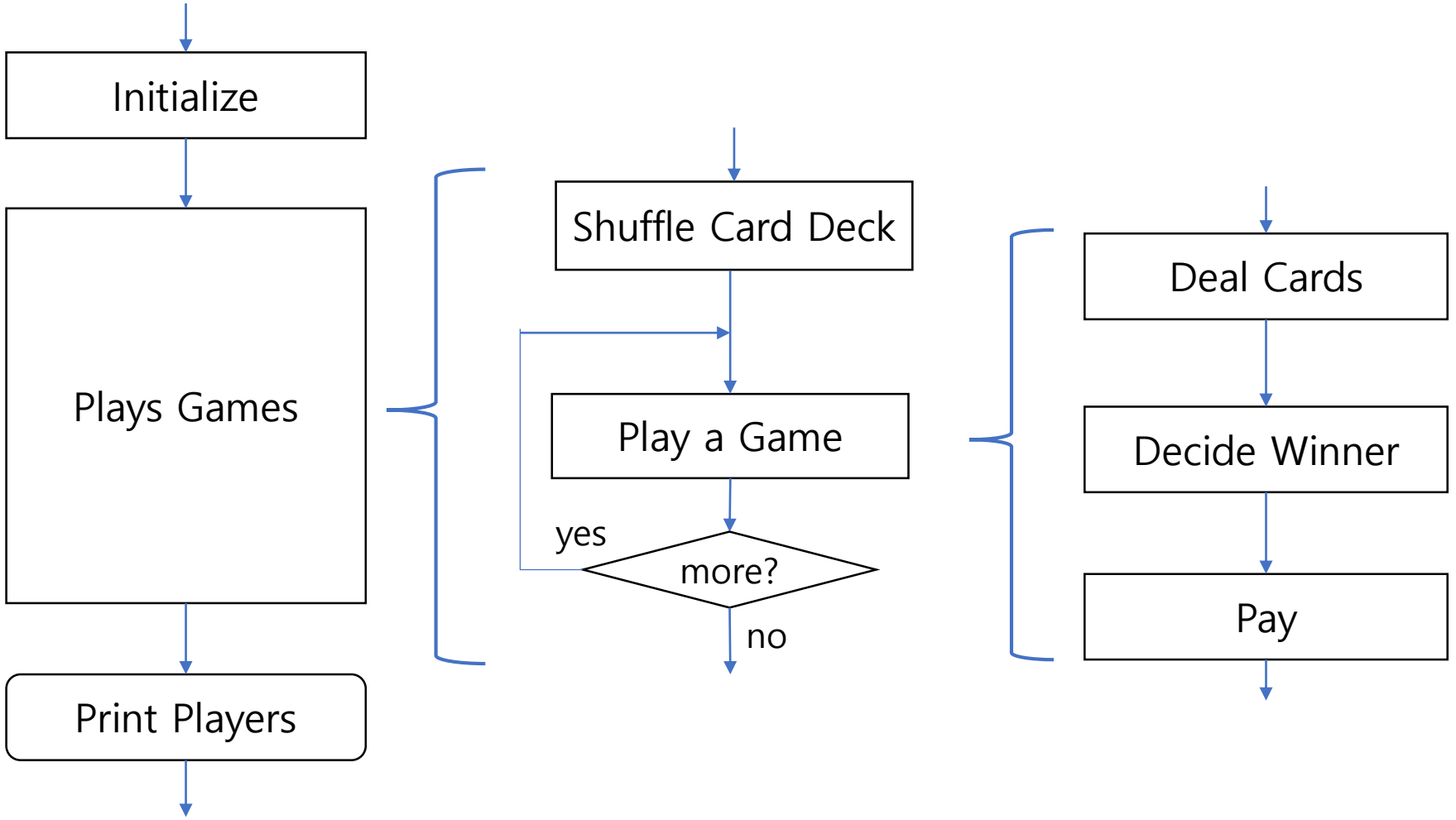
- 알고리즘 설계
  - Top-down 설계 (하향식 설계) – main 모듈(module)을 설계한 후 포함된 하위 모듈을 설계한다. 각 하위 모듈 또한 이와 비슷한 방법으로 설계해나간다.
  - 개략적인 설계 – 하나의 모듈을 주요 하위 태스크를 중심으로 큰 그림으로 설계한다.
  - 상세 설계 – 하나의 모듈을 문장 수준의 단위로 구체적인 설계를 한다.
- 알고리즘의 표현
  - Flow-chart
  - Pseudo-code

- 모듈 차트 (module chart or module hierarchy chart)



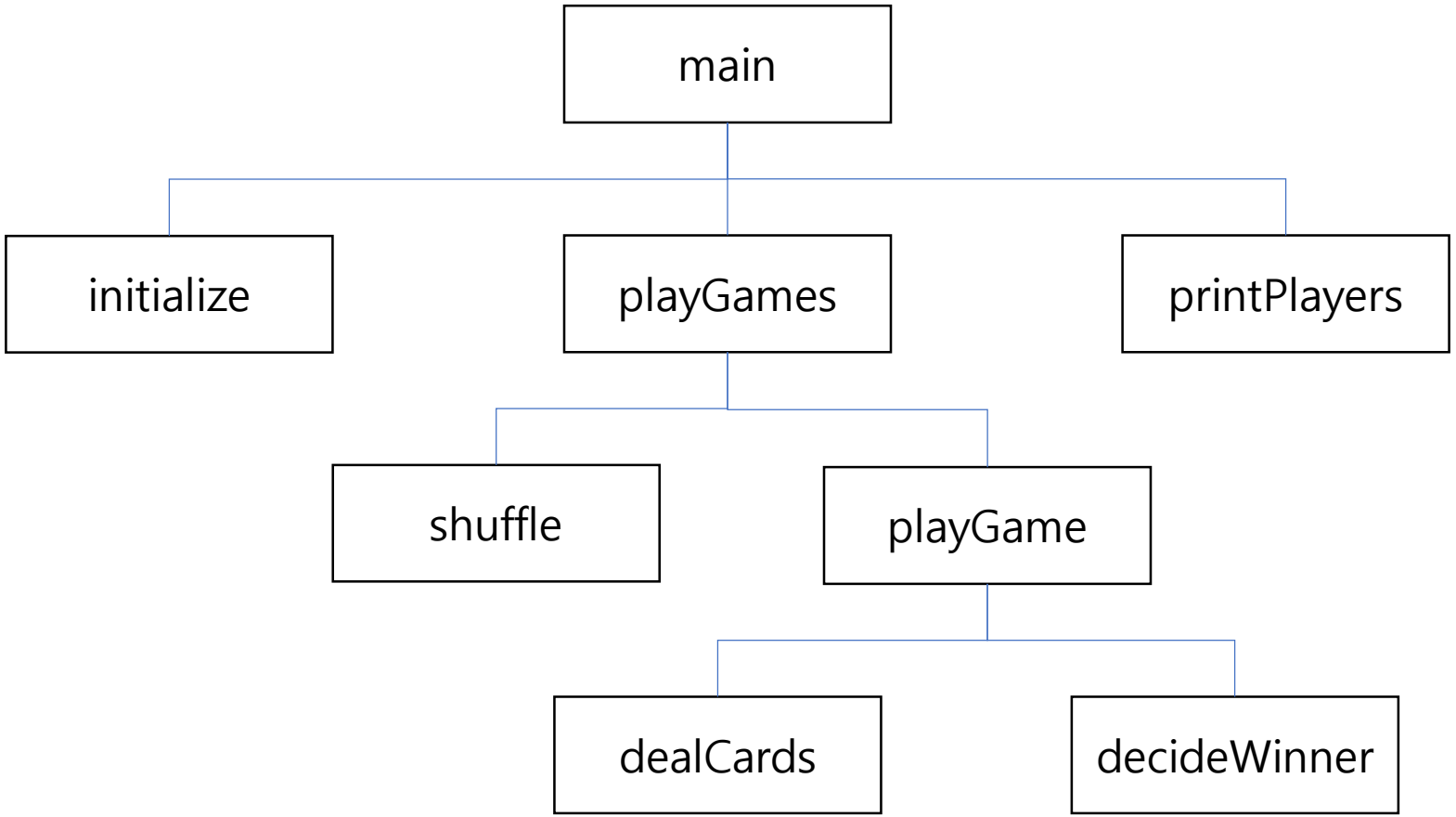
분석 및 설계 : 알고리즘 설계

- 카드게임 알고리즘 설계



분석 및 설계 : 모듈 차트

- 카드게임 모듈차트



분석 및 설계 : 세부 알고리즘 설계

- 주요 모듈의 세부 알고리즘 작성
  - Input 및 Ouput (return 값) 명시
  - Algorithm (pseudo code 또는 flow chart) 작성
  - 예외 조건 파악

- 카드게임의 주요 세부 알고리즘
  - shuffle
  - dealCards
  - decideWinner
  - etc.

Fucntion : shuffle
Input : deck[52]:Card (unshuffled)
Ouput: deck[52]:Card (shuffled)
Algorithm:

```
for(i=0; i<52; i++)
    select a random number k (0~51)
    swap deck[i] and deck[r]
end for
```

데이터와 알고리즘 연결

- 데이터를 알고리즘에 어떻게 전달할 것인가?

- Global Data vs Local Data
  - Global data – 모듈 간 공유
    - 모듈간 데이터 전달을 신경쓰지 않으므로 프로그래밍이 편리하다.
    - 모든 모듈이 데이터를 공유함으로써 유지보수가 어려워진다.
  - Local data – parameter passing
    - 명확한 통로를 통해 데이터를 주고받기 때문에 유지보수가 용이하다.
    - Parameter passing을 신경써야 하므로 프로그래밍이 복잡해지고 수행시간이 더 든다.
- 카드게임에서는?
  - 프로그램 사이즈가 작고 또한 많은 모듈들이 Card와 Player를 공통적으로 사용하기 때문에 global data가 유리할 것으로 판단 (절대적이지 않음)

구현 (코딩) 및 테스트

- 자료구조 결정
  - Card의 구현

```
typedef struct Card {
        int suit;
        int value;
    } CARD;
```

  - Player의 구현

```
typedef struct Player {
        char name[20];
        int balance;
        CARD hand;
    } PLAYER;
PLAYER palyers[2];
```

  - Card Deck의 구현

```
CARD deck[52];
```

구현 (코딩) 및 테스트

- 모듈의 구현
  - Top-down 구현
  - Bottom-up 구현

- Top-down 구현
  - 모듈 차트의 main 모듈에서 출발하여 하위 모듈로 진행하면서 구현
  - 하위 모듈이 구현되기 전이기 때문에 상위모듈을 테스팅하기 위해서는 dummy 모듈 이용
  - 상위 모듈의 테스트가 불완전할 수가 있음

- Bottom-up 구현
  - 가장 하위 모듈부터 시작해서 상위로 진행하면서 구현
  - 이미 구현된 모듈을 이용하여 구현하기 때문에 테스팅이 용이
  - 하위 모듈을 위한 테스팅 데이터를 직접 마련해야 함