Data structure [B03]

김종규, PhD

2017-03-22

Outline

- Abstract data type
- 연습문제 풀이

Administrivia

- 조교선생님
  - 김민수, 김우일
  - 실습과 채점에 대한 모든 것
  - 과제 구현시 궁금한 부분에 대한 질문
  - 기타 궁금한 것들

Queue 의 활용

그림: Super computer

Job queue

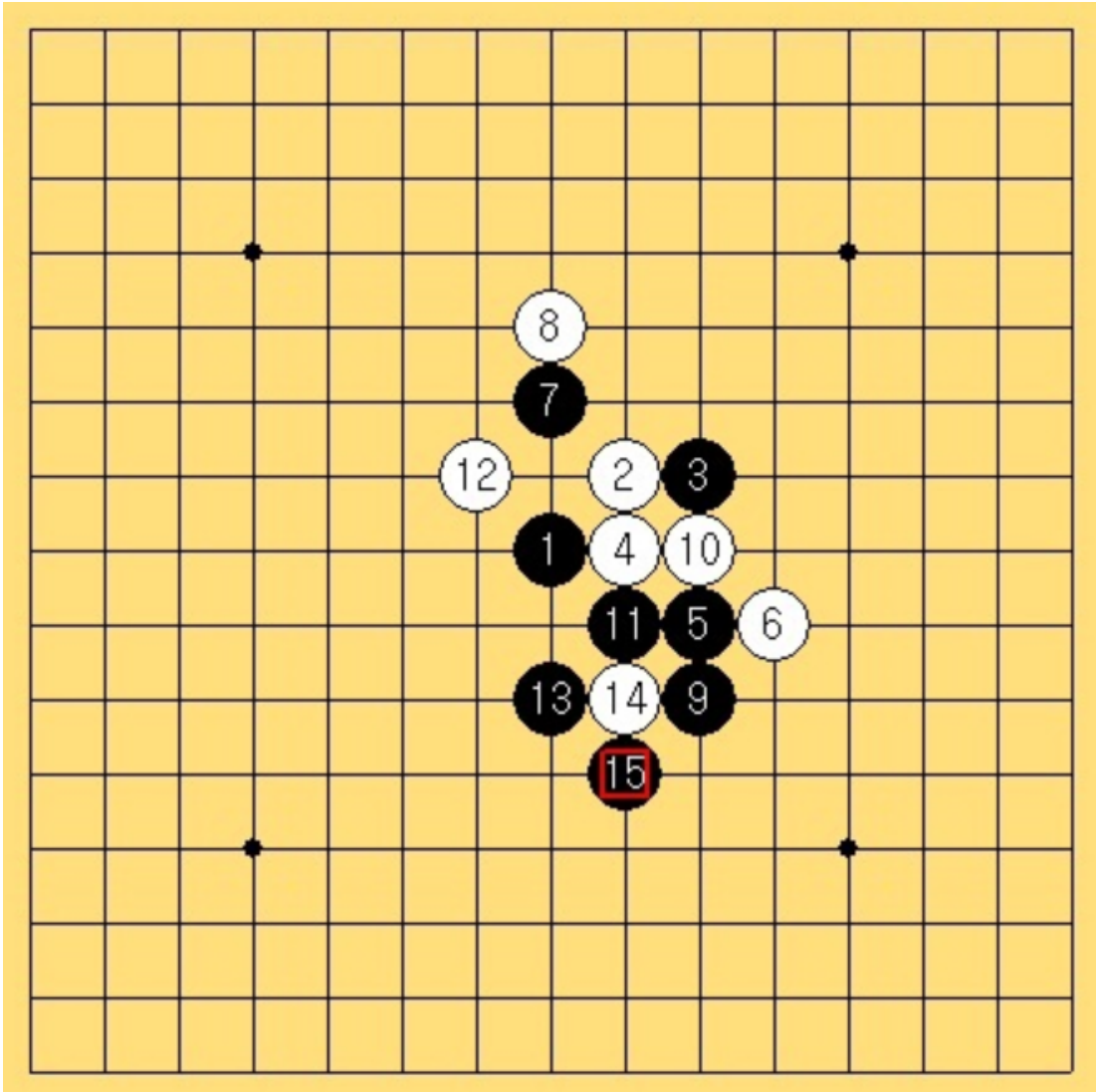

그림: 오목

Polish calculator

- 3 + 4 * 2 =
  - 4 2 multiply $\longrightarrow$ 8
  - 3 add $\longrightarrow$ 11

$\longrightarrow$ Reverse Polish notation

- 3 4 2 multiply add

Polish calculator

그림: Programmable calculator

Stack in C

```
#define STKSZ 100
int stk[STKSZ];
int top = 0;
```

Stack in C

```
void push(int val) {
  stk[top++] = val;
}
int pop() {
  return stk[--top];
}
int is_empty() {
  return top == 0;
}
```

Stack in C

```
int main() {
  push(1);
  push(2);
  push(3);
  printf("%d\n", pop());
  printf("%d\n", pop());
  printf("%d\n", pop());
  return 0;
}
```

Question

- 다음과 같이 push 를 정의하는 것이 가능할까?

```
void push(int val) {
  top = top + 1;
  stk[top] = val;
}
```

⟶ 가능하다, 그러나 pop, empty 의 정의도 바뀌어야 한다

Answer

- 다음과 같이 정의할 수 있다

```
#define STKSZ 100
int stk[STKSZ];
int top = -1;
int pop() {
  top = top - 1;
  return stk[top + 1];
}
int is_empty() {
  return top == -1;
}
```

$\longrightarrow$ 일관성이 중요

다음 함수는 무슨 일을 할까?

- 두 개의 stack (s1, s2)

```
def p(item):
  push(s1, item)
def q():
  if (empty(s2)):
    while not empty(s1):
      push(s2,pop(s1))
  return pop(s2)
```

$\longrightarrow$ queue

$\longrightarrow$ Stack 을 필요한 만큼 만들어서 사용할 수 있을 때 큰 장점

Stack as a value

$\longrightarrow$ C 에서 어떻게 구현할까?

Stack as an abstract data type


```
struct Stack {
  int stk[STKSZ];
  int top = 0;
};
void push(struct Stack* stk, int val) {
  stk->stk[stk->top++] = val;
}
```

⟶ Error: struct 의 element 는 초기화시킬 수 없다

Initialization of an abstract data type

```
void init(struct Stack* stk) {
  stk->top = 0;
}
struct Stack stk;
int test() {
  init(&stk);
  push(&stk, 1);
  push(&stk, 2);
  push(&stk, 3);
  printf("%d\n", pop(&stk));
  printf("%d\n", pop(&stk));
  printf("%d\n", pop(&stk));
}
```

$\longrightarrow$ Stack 을 더 이상 사용하지 않는다면?

Abstract data type

- 어떤 목적을 위하여 값이 정의되고 (예: stack)
- 그 값을 다룰 수 있는 연산이 정의되고 (예: push, pop, empty)
- 이 연산의 결과가 일관성있게 유지될 수 있어야 한다

Lifecycle of an abstract data type

- Init, destroy

```
struct Stack {
  int* stk;
  int top;
};
void init(struct Stk* stk, int initsz) {
  stk->stk = (int}malloc(initsz * sizeof(int));
  stk->top = 0;
}
void destroy(struct Stk* stk) {
  free(stk->stk);
}
```

OOP and abstract data type

- 공통으로 갖는 것
  - Initialization (init)
  - Operations
  - Termination (destroy)
- OOP 에 특화된 부분
  - Inheritance
  - Polymorphism

Abstract data type in python

```
class Stack:
    def __init__(self):
        self.items = []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def is_empty(self):
        return self.items == []
```

Abstract data type in python

```
stk = Stack()
stk.push(1)
stk.push(2)
stk.push(3)
print(stk.pop())
print(stk.pop())
print(stk.pop())
```

Wrap-up

- Abstract data types define values and the operations on them
- To maintain consistency, all the operations of an abstract data type should be considered as a whole