# LAB 4

STRUCTURE AND CLASS

## Outline

- Structure in C
- Class
- Exercise
- Appendix

# Structure in C

- A user-defined data type
  - (vs. built-in data types: int, double, …)
- An aggregate (grouping) data type
  - Array - collection of elements of the same type
  - Structure - collection of elements of different types
- Usage
  - First define your own structure
  - Then declare objects using new structure type just like declaring objects of built-in types

## Structure Example (1/2)

- Definition

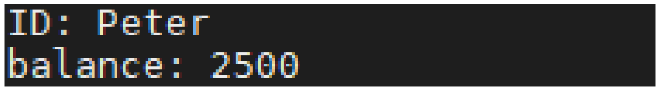
```c
struct EasyCard
{
    char id[10];
    int money;
};
```

- Usage

```c
int main () {
    struct EasyCard card = {"Peter", 1000};
    // struct EasyCard card = {.money = 1000, .id = "Peter"};
    card.money += 2000;
    struct EasyCard *card_ptr = &card;
    card_ptr->money -= 500;
    // (*card_ptr).money -= 500;
    printf("ID: %s\n", card.id);
    printf("balance: %d\n", card.money);
    return 0;
}
```

## Structure Example (2/2)

- Output



```
ID: Peter
balance: 2500
```

## Limitations of Structure

- Prohibit functions inside structures
- No static members
- No constructors and destructor

- All above are the features of class in C++

## Class

- The foundation for OOP in C++
- Class in C++ is an enhanced version of structure in C
  - Access modifiers
  - Functions inside classes
  - Static data members
  - Constructors and destructor
  - Operator overloading ➔ **Can be used as built-in data types**
- Usage
  - Same as structures

## Class Member Function

- Must define(i.e., implement) class member functions
- If defined outside class definition, MUST specify the class it belongs to
    - <ret_type> <cls_name>::<func_name> (…) {…}
    - “::” is called scope resolution operator
        - Different classes can have member functions with the same name
    - Items before “::” are called type qualifier
        - Class name serves as type qualifier here

## Private vs. Public

- Both data members and member functions can be either private or public
- Data members are usually private
  - You don’t know exact representation ➔ encapsulation
  - Manipulated through member functions
- Member functions are usually public
  - You can use public interface for manipulations
  - you needn’t know how these functions get implemented ➔ abstraction

# Class Example (1/2)

- Definition

```
class EasyCard
{
    public:
    EasyCard (const char *id, int money) : money(money) { strcpy(this->id, id); }
    const char * get_ID () { return id; }
    void add_value (int num) { money += num; }
    void pay (int num) { money -= num; }
    int get_balance () { return money; }

    private:
    char id[10];
    int money;
};
```

## Class Example (2/2)

- Usage

```
int main () {
    EasyCard card("Peter", 1000);
    cout << "ID: " << card.get_ID() << endl;
    card.add_value(2000);
    card.pay(500);
    cout << "balance: " << card.get_balance() << endl;
    return 0;
}
```

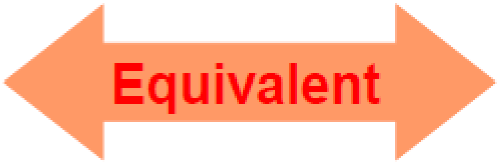
- Output

```
ID: Peter
balance: 2500
```

## Structure in C++

- In fact, structure in C++ is a class basically
- Difference
  - a structure assumes all members public by default
  - a class assumes all members private by default

```
struct C1 {
private:
        // …
};
```



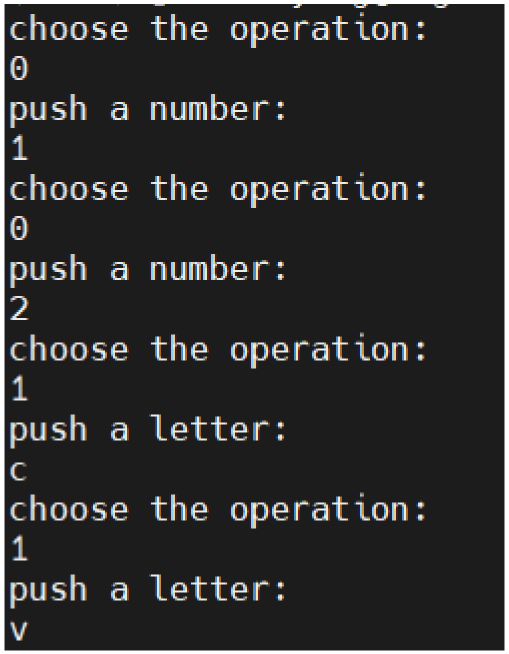
```
class C1 {

        // …
};
```

## Exercise Objective

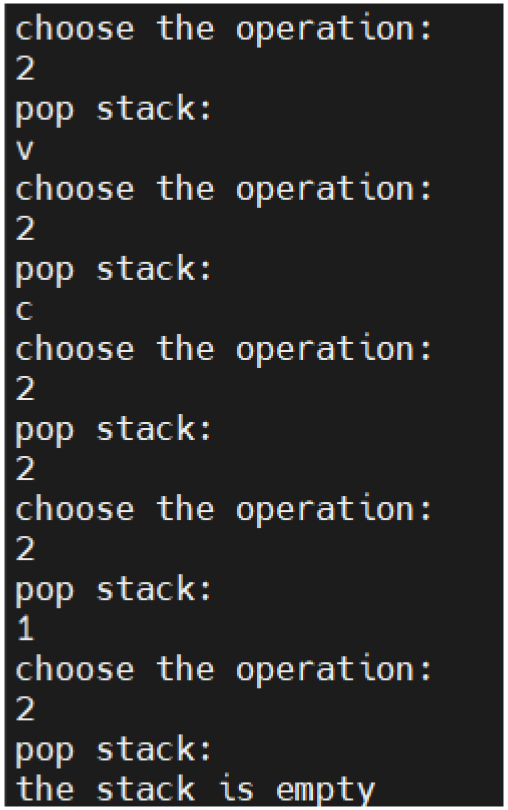
- Practice to write a class
- Learn how to solve a problem with the class

## Lab Exercise (1/2)

- You are asked to store input data in a stack
- Example



```
choose the operation:
0
push a number:
1
choose the operation:
0
push a number:
2
choose the operation:
1
push a letter:
c
choose the operation:
1
push a letter:
v
```



```
choose the operation:
2
pop stack:
v
choose the operation:
2
pop stack:
c
choose the operation:
2
pop stack:
2
choose the operation:
2
pop stack:
1
choose the operation:
2
pop stack:
the stack is empty
```

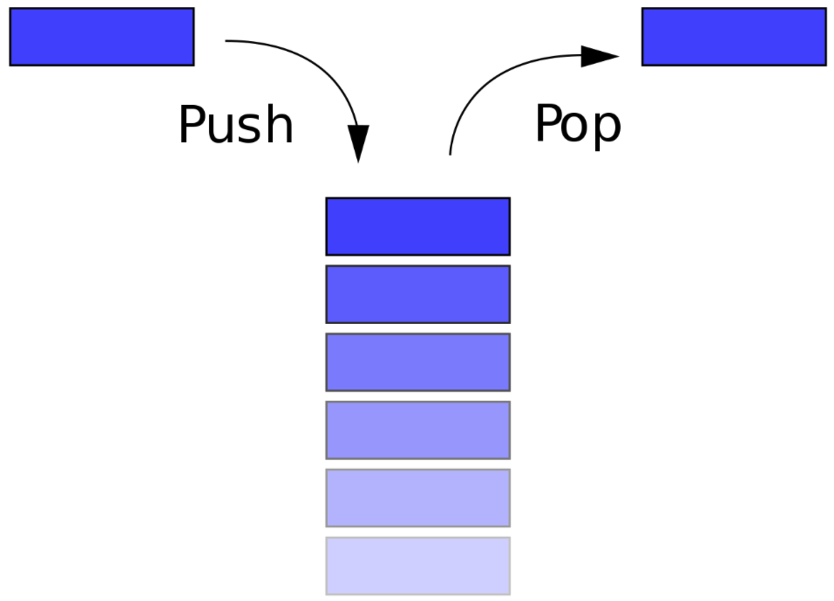
## Lab Exercise (2/2)

- You CANNOT use STL container (e.g., list, stack, …)
- You are asked to
  - Use class to construct a stack
  - Create a structure as data type in the stack
- No checking of syntax error needed

## Appendix

## Introduction to Stack

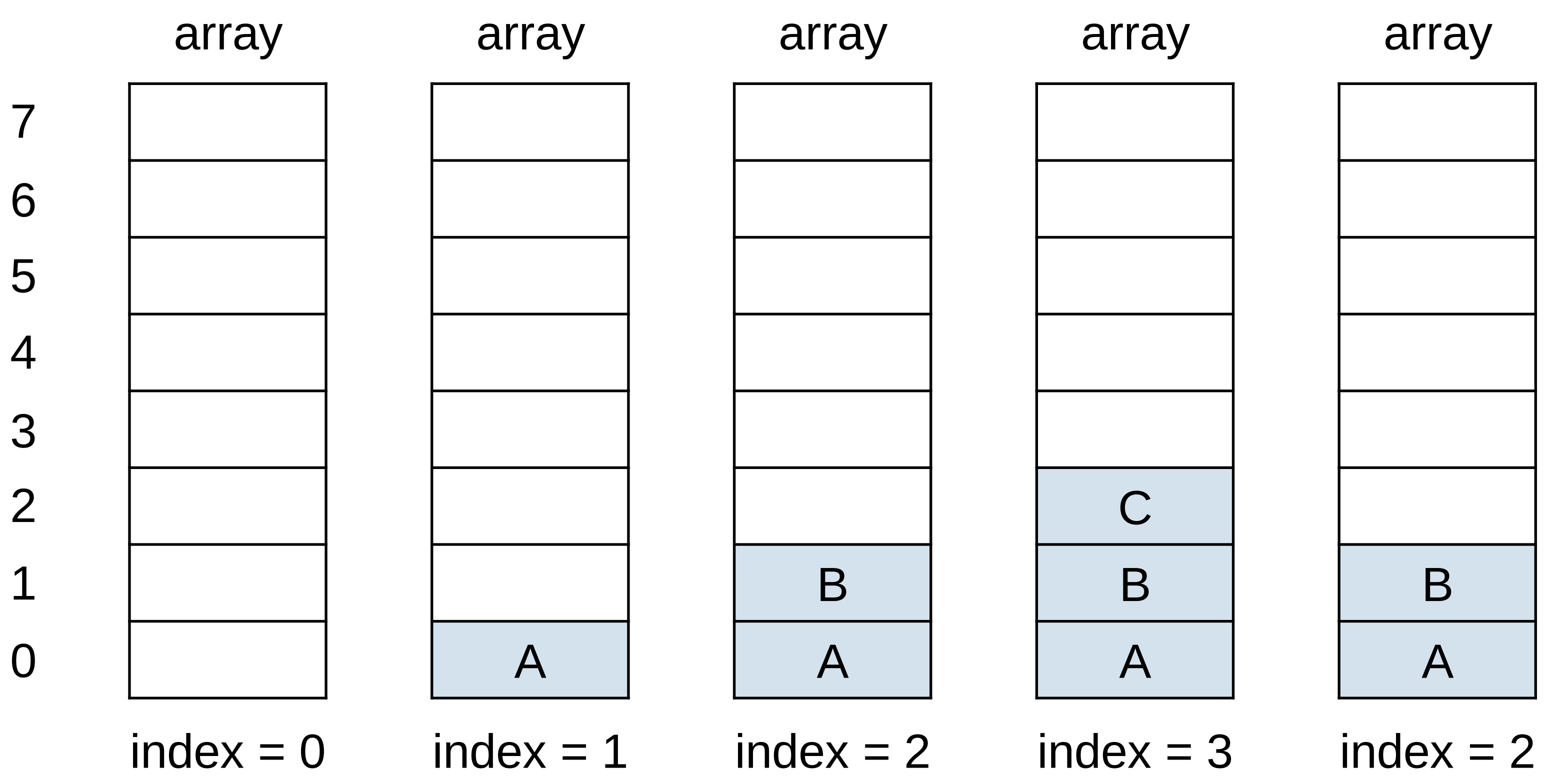
- A container with last-in-first-out (LIFO) property
- Two operation
  - push – put a data on the top of the stack
  - pop – fetch a data from the top of the stack



## Prescribed Functions for Stack

- Stack ();
  - Constructor for initializing data members
- void push (Data);
  - Perform push operation
- Data pop ();
  - Perform pop operation
- bool empty ();
  - Return true if the stack is empty, otherwise return false

# Data Representation of Stack


array
array
array
array
array
7
6
5
4
3
2
1
0
A
B
A
C
B
A
B
A
index = 0
index = 1
index = 2
index = 3
index = 2