Tutorial 10

ZHANG Xinyun

- Assignment 1 exercises

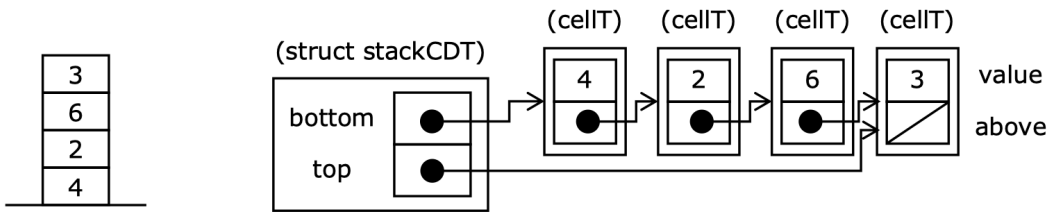
Q1

In the lecture we have completed a concrete implementation using linked lists for stack ADT. In this exercise we are going to try two new implementations of the stack ADT.

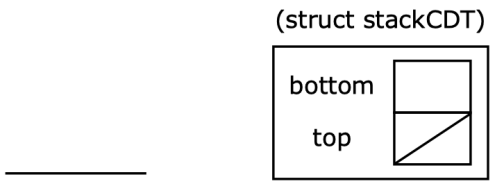
The first new implementation will be called version 5.0. The main idea is to implement a stack using a linked list. Specifically, assuming that all stack elements are integers, we write in "stack.c" the following concrete implementation

```
#include "stack.h"
/* version 5.0 */

typedef struct cellT {stackElementT value; struct cellT *above;} cellT;

struct stackCDT {
  cellT *bottom;
  cellT *top;
};
```

The following diagrams show the implementation of a nonempty stack:



The following diagrams show the implementation of an empty stack:



Complete the implementation version 5.0.

Q1

1. Header file - Stack.h

```
typedef struct stackCDT *stackADT;
typedef int stackElementT;
stackADT EmptyStack(void);
void Push(stackADT stack, stackElementT element);
stackElementT Pop(stackADT stack);
int StackDepth(stackADT stack);
int StackIsEmpty(stackADT stack);
```

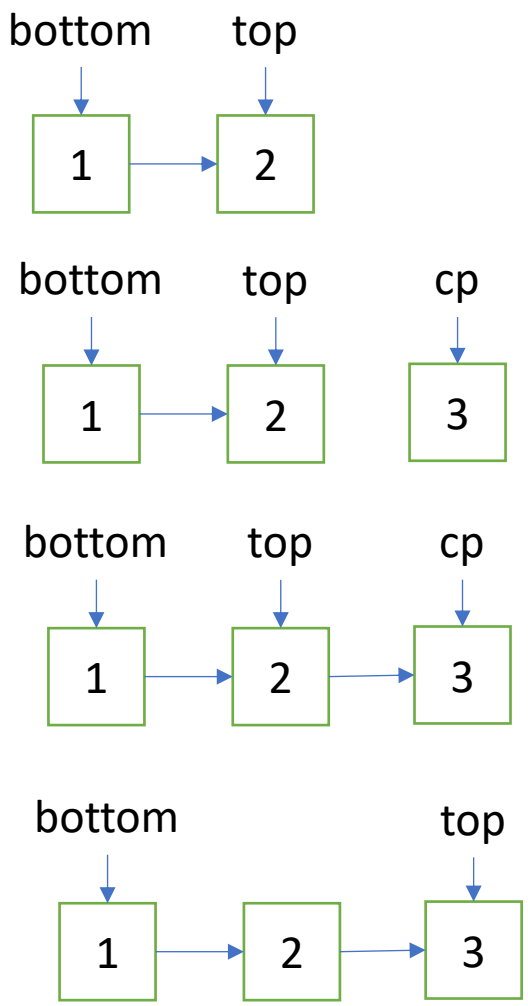
Q1

2. Implementation file - Stack.c

```c
typedef struct cellT{
    stackElementT value;
    struct cellT* above;
} cellT;

struct stackCDT {
    cellT *bottom;
    cellT *top;
};

stackADT EmptyStack(void){
    stackADT stack;
    stack = (stackADT)malloc(sizeof(*stack));
    stack->bottom=NULL;
    stack->top=NULL;
    return(stack);
}
```

Q1

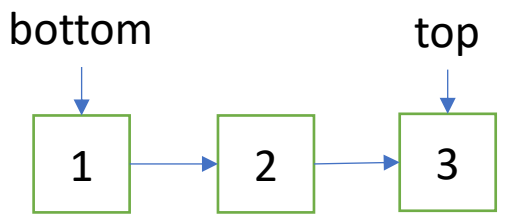
2. Implementation file - Stack.c

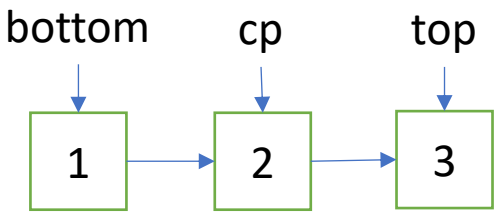


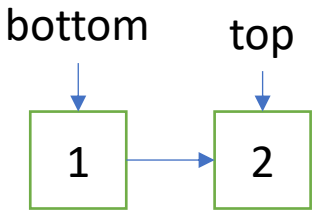

```
void Push(stackADT stack, stackElementT element){
    cellT *cp;
    cp = (cellT*) malloc(sizeof(cellT));
    cp->value = element;
    cp->above = NULL;
    if(stack->top==NULL){
        stack->top = cp;
        stack->bottom = cp;
    }
    else{
        stack->top->above = cp;
        stack->top = cp;
    }
}
```

Q1

2. Implementation file - Stack.c








```
stackElementT Pop(stackADT stack){
    cellT *cp, *element;
    if(stack->top == stack->bottom){
        element = stack->top;
        stack->top = NULL;
        stack->bottom = NULL;
    }
    else{
        for(cp=stack->bottom; cp->above!=NULL; cp=cp->above){
            if(cp->above == stack->top){
                element = cp->above;
                cp->above = NULL;
                stack->top = cp;
                break;
            }
        }
    }
    return element->value;
}
```

Q1

2. Implementation file - Stack.c

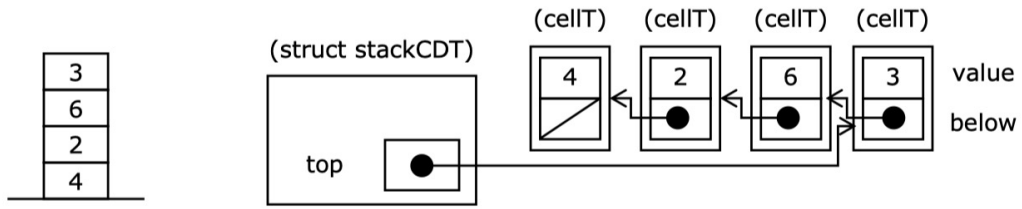
```
int StackDepth(stackADT stack){
    int n=0;
    cellT* cp;
    for(cp=stack->bottom; cp!=NULL; cp=cp->above){
        n++;
    }
    return(n);
}
int StackIsEmpty(stackADT stack){
    return(stack->top == NULL);
}
```

Q2

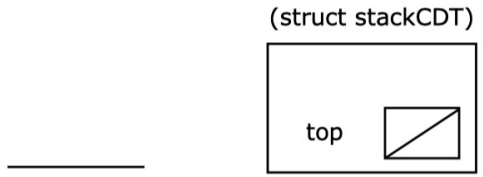
If we carefully examine the implementation version 5.0 of stackADT, it can be concluded that the design is bad. First, the field bottom in struct stackCDT is not necessary. In addition, the direction of the pointers in the linked list should be reversed. One can write the improved implementation, which will be called version 5.1.

```
#include "stack.h"
/* version 5.1 */

typedef struct cellT {stackElementT value; struct cellT *below;} cellT;

struct stackCDT {
  cellT *top;
};
```

The following diagrams show the implementation of a nonempty stack:



The following diagrams show the implementation of an empty stack:



Q2

1. Header file - Stack.h

```
typedef struct stackCDT *stackADT;
typedef int stackElementT;
stackADT EmptyStack(void);
void Push(stackADT stack, stackElementT element);
stackElementT Pop(stackADT stack);
int StackDepth(stackADT stack);
int StackIsEmpty(stackADT stack);
```

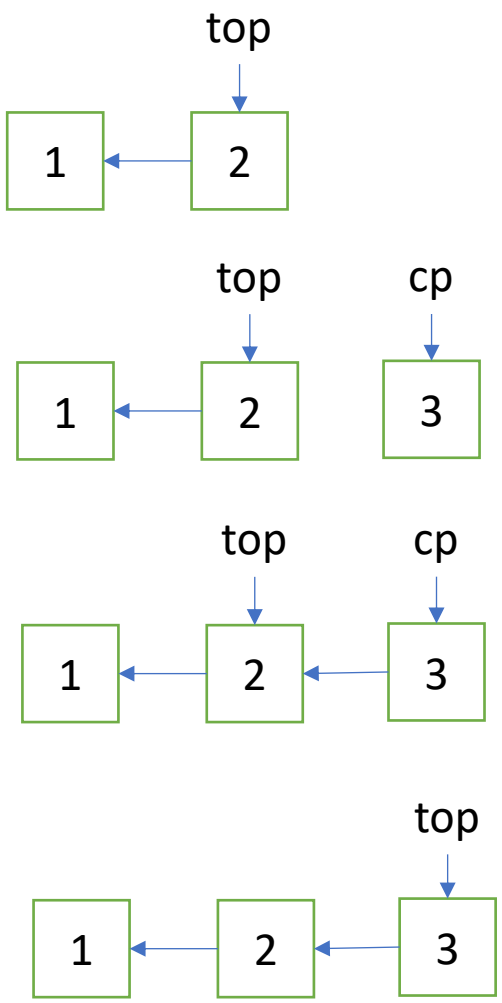
Q1

2. Implementation file - Stack.c

```c
typedef struct cellT{
    stackElementT value;
    struct cellT* below;
} cellT;

struct stackCDT {
    cellT *top;
};
stackADT EmptyStack(void){
    stackADT stack;
    stack = (stackADT)malloc(sizeof(*stack));
    stack->top=NULL;
    return(stack);
}
```

Q1

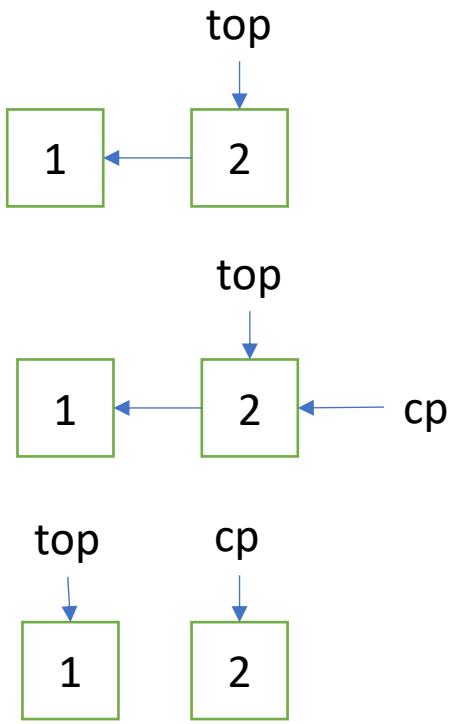
2. Implementation file - Stack.c




```c
void Push(stackADT stack, stackElementT element){
    cellT *cp;
    cp = (cellT*) malloc(sizeof(cellT));
    cp->value = element;
    cp->below=NULL;
    if(stack->top==NULL){
        stack->top = cp;
    }
    else{
        cp->below = stack->top;
        stack->top = cp;
    }
}
```

Q1

2. Implementation file - Stack.c




```
stackElementT Pop(stackADT stack){
    cellT *cp;
    cp = stack->top;
    stack->top = cp->below;
    cp->below = NULL;
    return cp->value;
}
```

Q1

2. Implementation file - Stack.c

```c
int StackDepth(stackADT stack){
    int n=0;
    cellT* cp;
    for(cp=stack->top; cp!=NULL; cp=cp->below){
        n++;
    }
    return(n);
}
int StackIsEmpty(stackADT stack){
    return(stack->top == NULL);
}
```

Q3

Write a program to simulate the following scenario. You should use the "queue.h" header file defined in the lecture notes. Your program should work well with the implementation shown in the lecture notes, as well as any correct implementation. (Note also that you need to appropriately modify the queueElementT.)

We consider a small bus terminus, from which three bus routes, namely, route A, route B and route C, start. At each of the bus stops of routes A, B and C, there is an LCD display board, on which the time the next bus leaves as well as the bus fare will be displayed. If the last bus has departed, then the display is turned off. Note that the bus fare of a bus route can be different at different time of a day. Passengers waiting for the buses form a queue at each of the bus stops.

We define a display board as a structure:

```
typedef struct displayBoard {
   int departureTime = -1;
   int busFare = 0;
} displayBoard;


typedef struct busStop {
   stackADT departureTime;
   stackADT busFare;
   queueADT passengerQ;
} busStop;
```

Q3

The member `departureTime` of `busStop` is a stack of integers that stores the departure times of a bus route, expressed in terms of the number of minutes since 0:00 of a day. The member `busFare` is a stack of integers that stores the different bus fares corresponding to the departure times stored in `departureTime`. Therefore, the heights of both stacks are always the same.

Finally, `passengerQ` is a queue of `PassengerADT` type defined as follows.

```
typedef struct Passenger {
   int PassengerClass; /* 0: child; 1: adult; 2: senior citizen */
   int routeAOK;
   int routeBOK;
   int routeCOK;
   int timeEnqueued;
} Passenger;

typedef Passenger *PassengerADT;
```

Note that a passenger can be a child, an adult, or a senior citizen. A child's bus fare is always half of the full fare, rounded down to the nearest dollar. A senior citizen's fare is either half of the full fare rounded down to the nearest dollar, or $2, whichever is the lower. If a passenger can take a route A bus, then `routeAOK` should be set to 1, otherwise it should be zero. The same is true for `routeBOK` and `routeCOK`. Finally, `timeEnqueued` records the time the passenger joins the queue.

Now write a C program to simulate the operations of the bus terminus in a day from 0:00 to 23:59. The main program consists in a loop, in which a control variable (say `time`) takes the integer values from 0 (representing 0:00) to 1439 (representing 23:59).

Q3

(1) Update the display boards using the information in the two stacks in a bus stop if necessary.

(2) If necessary, read information from 'PassengerArrival.txt'.

a. When a passenger arrives, he joins the corresponding queue immediately. However, if he can take more than one route, he will join the shortest queue. For example, if a passenger can take either route A or route B, and the current length of the queue at bus stop A is shorter than that at bus stop B, then he joins the queue at bus stop A. If the lengths of the two queues are exactly the same, then the passenger joins the queue for route with a lower bus fare. Finally, if the lengths of the queues and bus fares are all the same, then a passenger prefers route A to route B, and route B to route C.

b. Remember to initialise/ update the variables and members of structures.

(3) If a bus should depart at the current time, then all passengers in the corresponding queue get on the bus, and the queue becomes an empty queue. We assume that the buses are always big enough for all passengers in the queue to get on.

Q3

1. Load bus information

BusFare.txt, DepartTime.txt

10
8
6
4
9

```
void LoadSingleValue(char* path, stackADT stack){
    stackADT helperStack = EmptyStack();
    FILE *fp = fopen(path, "r");
    if(!fp){
        printf("Fail to open the file!!!");
        exit(0);
    }
    char line[100];
    int n = 0;
    while(fgets(line, sizeof(line), fp)!=NULL){
        line[strcspn(line, "\r\n")]='\0';
        if(strlen(line) == 0)
            continue;
        int element = atoi(line);
        Push(helperStack, element);
        n++;
    }
    fclose(fp);
    int len = StackDepth(helperStack);
    for(int i=0; i < len; i++){
        stackElementT element = Pop(helperStack);
        Push(stack, element);
    }
}
```

Q3

1. Load bus information

```
void LoadBusInformation(char* departPath, char* farePath, busStop* stop){
    stackADT departureTime = stop->departureTime;
    stackADT busFare = stop->busFare;
    LoadSingleValue(farePath, busFare);
    LoadSingleValue(departPath, departureTime);
}
```

Q3

2. Load passenger information

460 1 A
460 2 B A
461 0 A C
479 1 C
481 1 C
483 1 C
490 0 A C
490 0 A C
500 1 A C
500 2 B
550 0 C B
580 1 A B C
600 2 A B
610 0 B C

```c
void LoadPassengerArrival(char* passengerArrivalPath, queueADT queue){
    FILE *fp = fopen(passengerArrivalPath, "r");
    if(!fp){
        printf("Fail to open the file!!!");
        exit(0);
    }
    char line[100];
    int n = 0;
    while(fgets(line, sizeof(line), fp)!=NULL){
            line[strcspn(line, "\r\n")]='\0';
            if(strlen(line) == 0)
                continue;
            PassengerADT passenger = (PassengerADT) malloc(sizeof (Passenger));
            passenger->PassengerClass=-1;
            passenger->routeAOK=0;
            passenger->routeBOK=0;
            passenger->routeCOK=0;
            passenger->timeEnqueued=0;
            char *token = strtok(line, " ");
            int cnt = 0;
            while(token != NULL){
                // printf("the token is %s\n", token);
                if(cnt == 0){
                    int time = atoi(token);
                    passenger->timeEnqueued = time;
                }
                if(cnt == 1){
                    int passengerType = atoi(token);
                    passenger->PassengerClass = passengerType;
                }
                if(cnt > 1){
                    if(strcmp("A", token) == 0)
                        passenger->routeAOK = 1;
                    if(strcmp("B", token) == 0)
                        passenger->routeBOK = 1;
                    if(strcmp("C", token) == 0)
                        passenger->routeCOK = 1;
                }
                token = strtok(NULL, " ");
                cnt++;
            }
            n++;
            Enqueue(queue, passenger);
    }
    fclose(fp);
}
```

Q3

3. Simulation – main framework

```
for(int time=0; time<1439; time++){
    // 1. update the DisplayBoard
    // 2. passenger selects a queue
    // 3. update the busStop
}
```

Q3

3. Simulation – update the DisplayBoard

```
void UpdataDisplayBoard(displayBoard* board, busStop* stop, int currrentTime){
    // check whether the last bus has left or not
    if(currrentTime > board->departureTime){
        int len = StackDepth(stop->departureTime);
        if(len > 0){
            board->departureTime = Pop(stop->departureTime);
            board->busFare = Pop(stop->busFare);
        }
        else{
            // no future bus
            board->departureTime = -1;
            board->busFare = 0;
        }
    }
}
```

Q3

3. Simulation – passenger to choose the bus

(1) Compare B and C
(2) Compare A with the winner of B and C

	A	B	C
select	0	1	1
fare	8	6	4
queueLen	10	5	8

```
int CompareTwoRoutes(int const *queueLen, int const *fare, int const *select, int i, int j){
    // both i and j are not selected
    if(select[i] == 0 && select[j] == 0)
        return -1;
    // only j is selected
    if(select[i] == 0)
        return j;
    // only i is selected
    if(select[j] == 0)
        return i;
    // both i and j can be selected
    // 1. compare queue length
    if(queueLen[i] > queueLen[j])
        return j;
    if(queueLen[i] < queueLen[j])
        return i;
    // 2. compare fares
    if(fare[i] < fare[j])
        return i;
    if(fare[i] > fare[j])
        return j;
    // 3. alphabetic order
    if(i < j)
        return i;
    return j;

}
```

Q3

3. Simulation – enqueue the passenger and update the statistics

```
switch (index) {
    case 0:
        selectedQueue = busStopA.passengerQ;
        cnt = &cntA;
        totalFare = &fareA;
        totalTime = &timeA;
        timeToDepart = displayBoardA.departureTime;
        break;
    case 1:
        selectedQueue = busStopB.passengerQ;
        cnt = &cntB;
        totalFare = &fareB;
        totalTime = &timeB;
        timeToDepart = displayBoardB.departureTime;
        break;
    case 2:
        selectedQueue = busStopC.passengerQ;
        cnt = &cntC;
        totalFare = &fareC;
        totalTime = &timeC;
        timeToDepart = displayBoardC.departureTime;
        break;
    default:
        printf("Bus route selection error!");
        exit(0);
}
Enqueue(selectedQueue, passenger);
// check whether there are buses
if(timeToDepart > 0) {
    (*totalTime) += timeToDepart - passenger->timeEnqueued;
    (*cnt)++;
    (*totalFare) += fare[index];
}
```

Q3

3. Simulation – update the busStop

```
if(displayBoardA.departureTime == time)
    busStopA.passengerQ = EmptyQueue();
if(displayBoardB.departureTime == time)
    busStopB.passengerQ = EmptyQueue();
if(displayBoardC.departureTime == time)
    busStopC.passengerQ = EmptyQueue();
```