CSCI 2100C Data Structures

Quiz 1
22 February 2022
12:30 – 13:15

Note:

- You must work on the paper sent to your own email address.
- The quiz lasts for 45 minutes, until 13:15.
- You should not communicate with other people (including other candidates) after the quiz starts, until 13:35 today.

IMPORTANT—How to submit your answers:

- You shall submit your answer to the Blackboard system by **13:35** today (that is, you will have 20 minutes after the quiz to prepare the submission).
- You should write down your student ID on the first page of the answers.
- You should include all your answers in one single file. The file should be in PDF or JPG format.

Answer ALL Questions.

1. Teresa defines the symbol table ADT in the following header file symtab.h. The keys are character strings (char*) and the values are pointed to by void pointers (void*).

```
/*
 * File: symtab.h
 */

typedef struct symtabCDT *symtabADT;

symtabADT EmptySymbolTable(void);
void Enter(symtabADT, char*, void*);
void *Lookup(symtabADT, char*);
int SymTabIsEmpty(symtabADT);
```

Teresa correctly implements the symbol table ADT using Hash tables in file symtab.c. She uses open addressing hashing as the collision resolution scheme. Part of the implementation file symtab.c is shown below.

```
/*
 * File: symtab.c
 * By: Teresa
 */

#include <stdlib.h>
#include <string.h>

typedef struct cellT { char *key; void *value; struct cellT *next; } cellT;

struct symtabCDT { cellT *bucket[17]; }; /* Note the number of buckets */

int Hash(char *s) { … } // Not shown.

symtabADT EmptySymbolTable(void) {
  symtabADT table = (symtabADT)malloc(sizeof(*table));
  for (int i=0; i<17; i++) table->bucket[i] = NULL;
  return table;
}

void Enter(symtabADT table, char *key, void *value) { … } // Not shown.

void *Lookup(symtabADT table, char *key) { … } // Not shown.
```

a) **(20 marks)** The function NrOfEntriesEntered accepts one argument of the type symtabADT. It returns the number of entries already entered in the symbol table as an int value. Write the function NrOfEntriesEntered for Teresa's implementation.

int NrOfEntriesEntered(symtabADT T1) { … … }

Teresa uses the following function as the hash function in her implementation file symtab.c.

```
/* Author: Teresa */

int Hash(char *k1) {
  unsigned long hashcode = 0;
  for (int i=0; k1[i]!='\0'; i++) hashcode = (hashcode<<7) + missing part 1 ;
  return missing part 2 ;
}
```

b) **(5 marks)** What should Teresa write in the place marked ***missing part 1*** in the above hash function?

c) **(5 marks)** What should Teresa write in the place marked ***missing part 2*** in the above hash function?

After some time, Katherine (who is Teresa's friend) decides to re-implement Teresa's symbol table ADT using <u>double hashing</u> as the collision resolution scheme. She uses 17 buckets in her implementation.

Katherine writes a testing program and enters a number of entries to an empty symbol table. The keys of the entries and their respective primary and secondary hash codes are shown in the following table.

Entries	**Key**	**Primary Hash code**	**Secondary Hash code**
First entry entered	"Shui Chuen O"	11	8
Second entry entered	"Shek Mun"	11	1
Third entry entered	"Fung Wo"	10	2
Fourth entry entered	"Kwong Yuen"	15	4
Fifth entry entered	"Pok Hong"	13	10
Sixth entry entered	"Sha Kok"	13	10

d) **(2 marks)** In which bucket will the entry with key "Shui Chuen O" be stored?
e) **(2 marks)** In which bucket will the entry with key "Shek Mun" be stored?
f) **(2 marks)** In which bucket will the entry with key "Fung Wo" be stored?
g) **(2 marks)** In which bucket will the entry with key "Kwong Yuen" be stored?
h) **(2 marks)** In which bucket will the entry with key "Pok Hong" be stored?
i) **(2 marks)** In which bucket will the entry with key "Sha Kok" be stored?

In order to better understand Hash tables, Katherine decides to implement symbol table ADT again using separate chaining as the collision resolution scheme. She still uses 17 buckets in her implementation.

Assume that Katherine writes a testing program and enters the same number of entries to an empty symbol table in the same order as before. She does not change the Hash functions used in her double hashing implementation, therefore, the keys of the entries and their respective primary and secondary hash codes are same as before, as shown in the following table (hint: not all information in the following table is useful).

Entries	**Key**	**Primary Hash code**	**Secondary Hash code**
First entry entered	"Shui Chuen O"	11	8
Second entry entered	"Shek Mun"	11	1
Third entry entered	"Fung Wo"	10	2
Fourth entry entered	"Kwong Yuen"	15	4
Fifth entry entered	"Pok Hong"	13	10
Sixth entry entered	"Sha Kok"	13	10

j) **(8 marks)** Draw a diagram of Katherine's new implementation after all these entries are entered, to show where the entries with different keys are stored. Remember to show the bucket numbers (bucket 0, bucket 1, … etc.) next to each bucket.

2. Michael has written the following header files stack.h and queue.h for the ADTs of stack of integers and queue of integers. He has correctly implemented the ADTs in stack.c and queue.c.

```
/* File: stack.h */
/* Author: Michael */

typedef struct stackCDT *stackADT;
typedef int stackElementT;

stackADT EmptyStack(void);
void Push(stackADT, stackElementT);
stackElementT Pop(stackADT);
int StackIsEmpty(stackADT);
```

```
/* File: queue.h */
/* Author: Michael */

typedef struct queueCDT *queueADT;
typedef int queueElementT;

queueADT EmptyQueue(void);
void Enqueue(queueADT, queueElementT);
queueElementT Dequeue(queueADT);
int QueueIsEmpty(queueADT);
```

a) **(20 marks)** What will be printed if the following program is run?

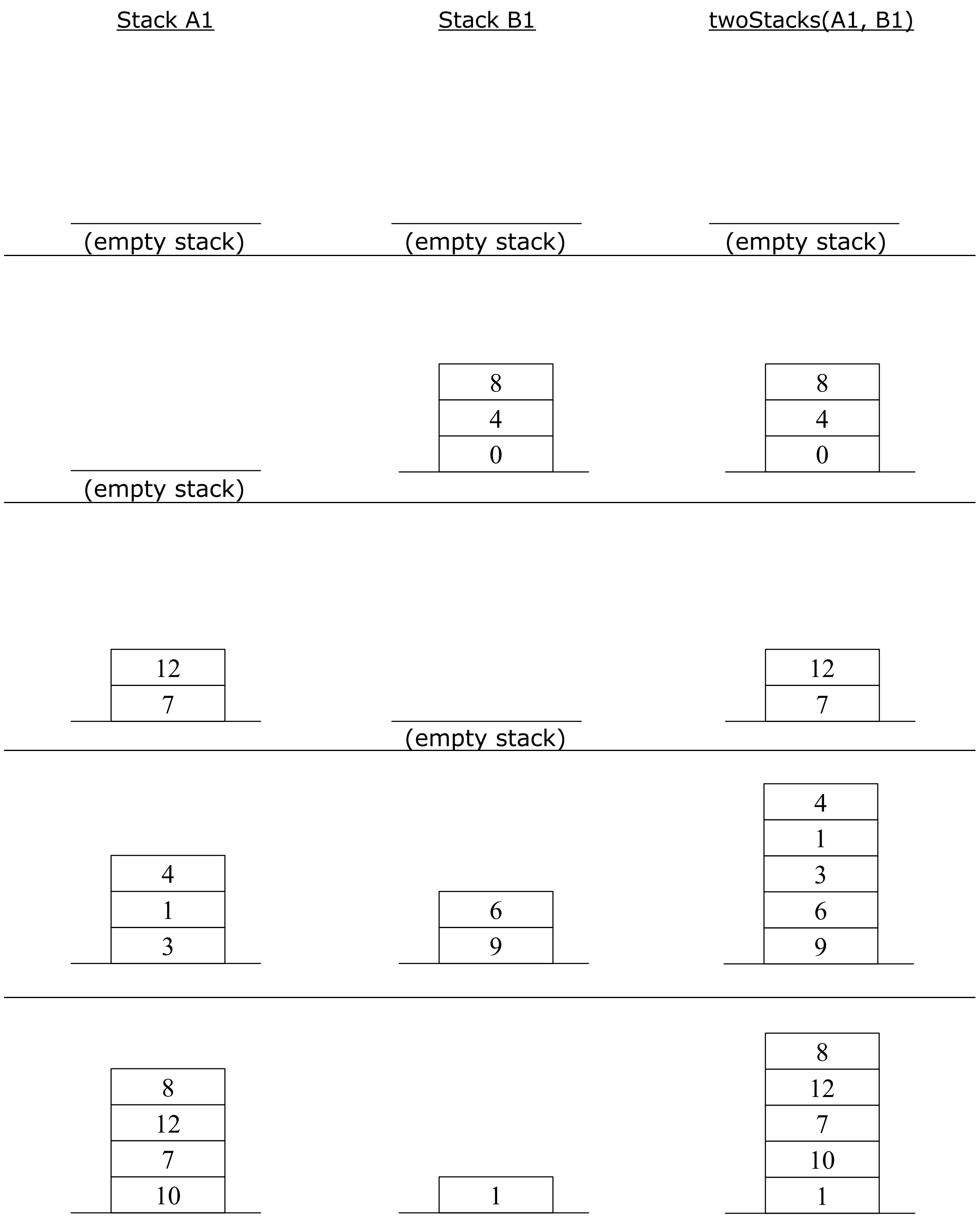
```
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"
#include "queue.h"

main() {
  stackADT S1 = EmptyStack();
  queueADT Q1 = EmptyQueue();
  Push(S1, 4); Push(S1, 5);
  Enqueue(Q1, Pop(S1)); Push(S1, 6); Enqueue(Q1, 7);
  Push(S1, 1 + Dequeue(Q1));
  while (!StackIsEmpty(S1)) printf("%d  ", Pop(S1)); printf("\n");
  while (!QueueIsEmpty(Q1)) printf("%d  ", Dequeue(Q1)); printf("\n");
}
```

Tom is going to use the stack ADT and queue ADT that Michael defines to write a function twoStacks. The function takes two stackADT arguments and returns a stackADT value:

stackADT twoStacks(stackADT A1, B1) { … … }

The function returns a new stack that is constructed with all elements of stacks A1 and B1. In the stack returned, all elements of stack B1 are put at the lower part, and all elements of stack A1 are put at the upper part, while the order of the elements are unchanged. Meanwhile, it is important that when the function returns, stacks A1 and B1 are not altered.

The following diagram shows some examples of the function call.



Tom has correctly written the function twoStacks, as shown in the following.

```
#include "stack.h"

stackADT twoStacks(stackADT A1, stackADT B1) {

  stackADT R = EmptyStack();
  stackADT AuxS = EmptyStack();
  stackElementT x;

  while (!StackIsEmpty(B1)) {x = Pop(B1); [missing part A]; Push(AuxS,x);}
  while ([missing part B]) Push(B1, Pop(AuxS));

  while (!StackIsEmpty(A1)) {x = Pop(A1); Push(R,x); [missing part C];}
  while ([missing part D]) Push(A1, Pop(AuxS));

  return [missing part E];
}
```

b) **(15 marks)** What should be written in the places marked ***missing part A***, ***missing part B***, ***missing part C***, ***missing part D***, and ***missing part E***?

Tom finds that the function stackDepth that returns the depth of a stack is not available in the header files Michael provides. Therefore, Tom decides to write this function in his application program.

```
#include "stack.h"

int stackDepth(stackADT T1) {

...

}
```

c) **(15 marks)** Complete the stackDepth function. Note that the argument should not be altered. (Hint: count the number of elements you can pop from the stack.)

— End of Quiz—