מבוא לתכנות מערכות
תרגיל בית מספר 4 (C++)

סמסטר אביב 2007

תאריך פרסום: 15/07/07

תאריך הגשה: 12/08/07 עד השעה 15:00 .

משקל התרגיל: 10% מהציון הסופי (תקף)

מתרגל אחראי לתרגיל: מאשה ניקולסקי nikol@cs.technion.ac.il

שעות קבלה לתרגיל:

18.07	יום ד'	10:30 – 11:30
19.07	יום ה'	15:30 – 16:30
25.07	יום ד'	10:30 – 11:30
26.07	יום ה'	15:30 – 16:30
01.08	יום ד'	10:30 – 11:30
		16:30 – 17:30
10.08	יום ו'	11:30 – 13:30

הנחיות חשובות:

- בכל השאלות בנוגע לתרגיל, יש לפנות למאשה (ולא למתרגל אחר). בכל פנייה למאשה בקשר לתרגיל, יש לכתוב בשורת הנושא (Subject): mtm4
- **מומלץ לקרוא את כל התרגיל ולעבור על הדוגמאות לפני תחילת הפתרון.**
- שאלות אשר התשובה עליהן נמצאת במסמך זה, בדוגמאות או בדף העדכונים של התרגיל- לא תענינה.

שימו לב לבדיקה האוטומטית

תרגיל זה יעבור בנוסף לבדיקה הידנית, גם בדיקה אוטומטית. לכן על התוכנית שתכתבו ליצר בדיוק את הפלט הנדרש בתרגיל. תוכנית הבדיקה האוטומטית מפנה את הפלט אשר התוכנית שלכם מייצרת לתוך קובץ, ולאחר מכן מפעילה את הפקודה diff של UNIX כדי לוודא שתוכן הקובץ זהה בדיוק לקובץ שהיא מצפה לקבל. לפיכך הבדל של תו אחד (אפילו סוף שורה או רווח מיותר) בין הקובץ המצופה לבין הקובץ שיווצר ע"י התוכנית שלכם יגרום לכך שתוכניתכם תיכשל בבדיקה. שימו לב, כי אינכם מתבקשים לכתוב ()main בתרגיל זה, אלא רק מחלקות. הבדיקה האוטומטית תיעשה עם ()main שאנחנו נכתוב. על מנת שתוכלו לבדוק את עצמכם ישנן מספר דוגמאות הנמצאות במדריך ~mtm/public/0607b/ex4/examples. יש לקרוא בעיון את הדגשים שבסוף התרגיל, הם מכילים מידע חשוב על שמות הקבצים אותם עליכם להגיש.

נושא התרגיל: מערכת לניהול מסעדה

מטרת התרגיל

כתיבת תוכנית ב - C++, כתיבת מודולים, כתיבה ושימוש ב - Templates למימוש מבנה נתונים גנרי.

תיאור התרגיל

חלק ראשון ושני של התרגיל יעסקו בכתיבת מחלקות המממשות מבנה נתונים גנרי של תור. בחלק השלישי תשתמשו במבנה הנתונים הגנרי לבניית מערכת לניהול יעיל של מסעדה.

חלק ראשון: הגדרת מבנה נתונים גנרי בשפת C++

בחלק זה הנכם נדרשים לממש מחלקה בשם Queue. מחלקה זו מממשת מבנה נתונים מסוג FIFO (First In First Out). המימוש מאפשר להוסיף אובייקטים לסוף התור ולהוריד אובייקטים מתחילת התור. כמו כן ניתן לבחון אובייקט הנמצא בראש התור ואובייקט הנמצא בסוף התור.

בסוף המסמך מופיע תיאור הפעולות המוגדרות על המחלקה הגנרית Queue.
בחלק זה משימתכם להגדיר (function declaration) ולממש (function definition) את המתודות מתוך התיאור המילולי הניתן.

הערות:

1. בנוסף לפעולות המוגדרות יתכן ותצטרכו לממש פונקציות נוספות על מנת שהמחלקה Queue תפעל באופן תקין. חשבו מהן פונקציות אלו והוסיפו אותן למימוש.

2. ניתן להניח כי לעצם המוכנס ל - Queue יש:

```
Default constructor and destructor
Copy constructor
operator=
operator<<
```

אין להניח מעבר לכך כל הנחה על האיברים.

3. אין להשתמש באף מחלקה מה - STL לצורך מימוש Queue.
4. בתרגיל זה אינכם נדרשים להשתמש במנגנון ה – exceptions, לכן בעת הקצאת זכרון יש להניח שההקצאה מצליחה תמיד.

בתיקייה ~mtm/public/0607b/ex4/examples ישנה דוגמא q_test.cpp המדגימה את השימוש במחלקה. עליכם לבנות את המחלקה Queue כך שהדוגמא תרוץ בצורה תקינה ותייצר את הפלט q_out ו – q_err.

חלק שני: הגדרת מבנה נתונים גנרי בשפת C++

בחלק זה נרצה לממש מחלקה PQueue המייצגת מבנה נתונים של תור עדיפויות (Priority Queue). בתור עדיפויות לכל אובייקט עדיפות (מספר שלם) והאובייקטים נשמרים בתור ממוינים לפי עדיפות זו, כשבראש התור נמצא האובייקט בעל העדיפות הגבוהה ביותר.
אם לשני אובייקטים אותה עדיפות אזי האובייקט שהוכנס מאוחר יותר יופיע אחרי האובייקט שהוכנס מוקדם יותר (כלומר הוא יופיע קרוב יותר לסוף התור).

בחלק זה משימתכם להגדיר (function declaration) ולממש (function definition) את המתודות מתוך התיאור המילולי הניתן.

הערות:

1. בנוסף למתודות המוגדרות יתכן ותצטרכו לממש פונקציות נוספות על מנת שהמחלקה PQueue תפעל באופן תקין. חשבו מהן פונקציות אלו והוסיפו אותן למימוש.

2. ניתן להניח כי לעצם המוכנס ל - PQueue יש:

Default constructor and destructor
Copy constructor
operator=
operator<
operator<<

אין להניח מעבר לכך כל הנחה על האיברים.

3. אין להשתמש באף מחלקה מה - STL לצורך מימוש PQueue.
4. בתרגיל זה אינכם נדרשים להשתמש במנגנון ה – exceptions, לכן במקרה של הקצאת זכרון יש להניח שההקצאה מצליחה תמיד.

בתיקייה ~mtm/public/0607b/ex4/examples ישנה דוגמא p_test.cpp המדגימה את השימוש במחלקה. עליכם לבנות את המחלקה PQueue כך שהדוגמא תרוץ בצורה תקינה ותייצר את הפלט p_out ו – p_err.

חלק שלישי: הגדרת מחלקה Restaurant

בחלק זה הינכם נדרשים לממש מחלקה Restaurant. תאור מדוייק של השירותים המסופקים על ידי המחלקה מופיע בסוף המסמך. בחלק זה עליכם להגדיר את המטודות בדיוק כפי שמתואר במסמך.

הערות:

1. בנוסף למתודות המוגדרות יתכן ותצטרכו לממש פונקציות נוספות על מנת שהמחלקה Restaurant תפעל באופן תקין. חשבו מהן פונקציות אלו והוסיפו אותן למימוש.
2. אין להשתמש באף מחלקה מה - STL לצורך מימוש Restaurant.
3. ניתן להשתמש במחלקה string.
4. בתרגיל זה אינכם נדרשים להשתמש במנגנון ה – exceptions, לכן במקרה של הקצאת זכרון יש להניח שההקצאה מצליחה תמיד.
5. בחלק מפונקציות המחלקה מופיעים ערכי חזרה אפשריים. ערכים אלה מופיעים בסדר עדיפותם, מהגבוה לנמוך. כלומר במקרה שקרו שתי שגיאות תוחזר זו עם העדיפות הגבוהה יותר.

בתיקייה ~mtm/public/0607b/ex4/examples ישנה דוגמא r_test.cpp המדגימה את השימוש במחלקה. עליכם לבנות את המחלקה Restaurant כך שהדוגמא תרוץ בצורה תקינה ותייצר את הפלט r_out ו – r_err.

בדיקה עצמית, הידור וקישור

לצורך בדיקת התרגיל סיפקנו 3 דוגמאות הרצה הנמצאות במדריך ~mtm/public/0607b/ex4/examples/. שימו לב שאלו רק דוגמאות (אלו לא הקבצים באמצעותם נבדוק את הפתרון שלכם), והן לא אמורות לבדוק את כל המקרים שהמחלקות שלכם אמורות להיות מסוגלות לטפל בהם (כלומר: אין מניעה שהמחלקות שלכם תעבודנה נכונה עם 3 הדוגמאות הנתונות, אך תכשלנה באחד המבחנים בבדיקה האוטומטית).

יש להעתיק את הקבצים אליכם:

```
cp -rf ~mtm/public/0607b/ex4/examples/* .
```

יש להריץ:

עבור המחלקה Queue

```
g++ –ansi –Wall –pedantic-errors q_test.cpp –o q_test
```

עבור המחלקה PQueue

```
g++ –ansi –Wall –pedantic-errors p_test.cpp –o p_test
```

עבור המחלקה Restaurant

```
g++ –ansi –Wall –pedantic-errors r_test.cpp Test.cpp
Restaurant.cpp Report.o RestaurantOrder.o OrderItem.o –o
r_test
```

שימו לב שאם כתבתם עוד מחלקות יש להוסיף את שמות קבצי ה- cpp שלהם לאחר Restaurant.cpp.

```
p_test >! out1 >&! err1
q_test >! out2 >&! err2
r_test >! out3 >&! err3
diff p_out out1
diff q_out out2
diff r_out out3
diff p_err err1
diff q_err err2
diff r_err err3
```

אם פקודת ה - diff לא מדפיסה כלום למסך, התוכנית שלכם רצה כמו שצריך. אחרת, אם diff הדפיס אפילו רווח, התוכנית שלכם לא עבדה כמו שצריך. כאמור, עליכם לבדוק את תוכניתכם עם קלטים נוספים!

הגשת התרגיל

מה מסופק לכם

תיקייה ~mtm/public/0607b/ex4 כוללת את מבנה התיקיות הבא:

src:

inlude: קבצי header של המחלקות הניתנות לכם.

bin: קבצי object של המחלקות הניתנות לכם, מחולקים לפי פלתפורמות:

unix: קבצי o. של t2.

linux: קבצי o. linux.

vs_2003: קבצי obj. של Visual Studio 2003.

vs_2005: קבצי obj. של Visual Studio 2005.

examples: קבצי דוגמא עם הפלט המצופה.

קבצים Test.h, Test.cpp – קבצים אלו משמשים לבדיקת המחלקה Restaurant.

מה עליכם להגיש

1. מסמך design – עמוד אחד מודפס (**דף מצד אחד בלבד!**) המתאר את מבנה המערכת בחלק השלישי של התרגיל. יש לפרט **במשפט אחד או שניים** (שימו לב, אין לכתוב סיפורים) מה מטרתה של כל מחלקה במערכת ומה היחס בין המחלקות (למשל יחס הכלה...). את היחס בין המחלקות ניתן ורצוי לתאר בעזרת תרשים (ציור).
שימו לב, חלק גדול מניקוד הבדיקה הידנית ינתן על ה - design של המערכת. על כן השקיעו זמן ומחשבה בתכנון המערכת טרם כתיבת הקוד.

2. קבצים:
   - Queue.h
   - PQueue.h
   - Restaurant.h, Restaurant.cpp
   - קבצים אחרים שכתבתם לצורך המימוש.

   אין להגיש את הקבצים שקבלתם.

עליכם להגיש קובץ מכווץ(**zip, ולא rar. !!!**) שבו הקבצים המכילים את תוכניתכם, דרך אתר האינטרנט של הקורס, או ישירות מחשבון "Grades" שלכם. ניתן ליצור את קובץ ה – zip בשתי הדרכים הבאות:

1. ב - windows: בעזרת WinZip.
2. ב – unix : בעזרת הפקודה zip. לדוגמא:

```
zip zip_file_name file_1 file_2 … file_n
```

שימו לב שאם הינכם מגישים את הקובץ דרך Windows, אך את ה – zip הכנתם ב-Unix, עליכם להקפיד להעבירו ל - Windows ב - binary mode. במידה ותשכחו לעשות זאת תקבלו הודעת שגיאה מסוג [11] במערכת ה – Grades .

דרישות , הגבלות הערות, רמזים ותוספות

- שימו לב ששמות הקבצים יהיו בדיוק כפי שצויין.

- יש להקפיד על שימוש נכון ומדוייק ב - const. זהו חלק מהתרגיל. אי הקפדה על שימוש ב-const תעלה בהרבה נקודות.

- יש להקפיד על שימוש נכון ומדוייק ב - reference. זהו חלק מהתרגיל. אי הקפדה על שימוש ב – reference תעלה בהרבה נקודות.

- יש להקפיד על תכנון נכון של התוכנית וכתיבה נכונה בשפת C++. בשלב זה של לימודיכם הנכם מסוגלים ונדרשים להיות מסוגלים לתכנן ולכתוב תוכנית בסדר הגודל הנתון בעצמכם. **תכנון התוכנית מהווה חלק מהותי מהתרגיל.**

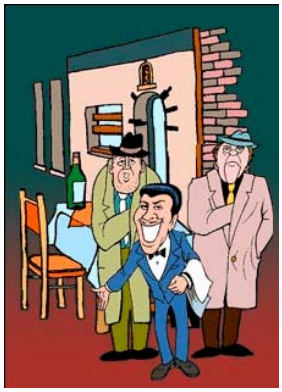
- את צורת ואופן התיעוד אנו מניחים לכם לקבוע לעצמכם. ההנחיה היחידה שתינתן, הינה כי מתכנת הרוצה להיעזר בפונקציות שאתם כותבים, צריך להיות מסוגל לבצע זאת ע"י הסתכלות בהצהרת הפונקציה וקריאת התיעוד, ואינו צריך לקרוא את גוף הפונקציה.

- יש להקפיד על ניהול זיכרון נכון: לשחרר זיכרון שאין בו צורך, ולא להקצות יותר זיכרון מהנדרש.

- על התרגיל להיות מוגש בזוגות. הנכם רשאים להגיש לבד, אך הדבר אינו מומלץ. עומס התרגיל תוכנן עבור שני סטודנטים. הגשה לבד לא תעניק הקלות. באתר הקורס מופיעה רשימת שידוכים לחסרי בני זוג.

- עליכם להגיש את התוכנית בשני אופנים: הן באמצעות האינטרנט והן באמצעות הגשת עותק מודפס של התוכנית לתא הקורס. להגשה המודפסת עליכם להוסיף כ"שער" את דף ההגשה שניתן בעמוד האחרון ולמלא בו את כל הפרטים באופן ברור.

- הגשה באיחור תגרור הורדת נקודות. האיחור המוצדק היחיד הנו שירות מילואים בתקופת התרגיל. עבור כל יום מילואים (של אחד מבני הזוג) תינתן הארכה בת יום אחד. כל יום איחור לא מאושר יגרור הורדה של 5 נקודות מציון התרגיל. לא יתקבלו תרגילים שיוגשו באיחור של יותר משבוע (פרט למילואים). הגשה באיחור מבוצעת באותו נוהל כמו הגשה רגילה (ידנית ואלקטרונית). יש לצרף הסבר לסיבת האיחור. המגישים באיחור עקב מילואים צריכים לצרף צילום של אישור מילואים לתרגיל המודפס. מאחרים אחרים יכולים לספק נימוק בכתב (ולצרף אישורים שונים אם ברצונם בכך), אך לא מובטח כי הנימוק יתקבל.



שינויים, עדכונים והודעות בנוגע לתרגיל

כל ההודעות הנוגעות בתרגיל ימצאו באתר של הקורס http://webcourse.cs.technion.ac.il/234122 בדף התרגילים (Assignments / Homework updates / ex4).
דף זה יכיל שאלות ותשובות נפוצות. רק הודעות דחופות תשלחנה בדואר. עליכם לעקוב אחר האתר והעדכונים שיפורסמו בו.

בהצלחה בתרגיל האחרון במת"ם!

class Queue Specification

A template container class that provides an access to the front and back elements. Elements can be added at the back and removed from the front, and elements can be inspected at either end of the queue.

The class defines the following operations:

push
Description:
Adds an element to the back of the queue.
Parameters:
Element to insert to the queue.
Return Value:
void

pop
Description:
Removes an element from the front of the queue.
Parameters:
none
Return Value:
void
Remarks:
The queue must be nonempty to apply this operation. If the queue is empty when pop operation is applied, the behavior of the function is undefined.

back
Description:
Returns a reference to the last and most recently added element at the back of the queue.
Parameters:
none
Return Value:
The last and most recently added element at the back of the queue. If the queue is empty, the return value is undefined. (See detailed remark in the pop function.)
Remarks:
If the return value of back is assigned to a const reference, the queue object cannot be modified. If the return value of back is assigned to a reference, the queue object can be modified. (See an example in q_test.cpp that is supplied to you.)

front
Description:
Returns a reference to the first element at the front of the queue.
Parameters:
none
Return Value:
The first element at the front of the queue. If the queue is empty, the return value is undefined. (See detailed remark in the pop function.)
Remarks:
If the return value of front is assigned to a const reference, the queue object cannot be modified. If the return value of front is assigned to a reference, the queue object can be modified. (See an example in q_test.cpp that is supplied to you.)

empty
Description:
Tests if a queue is empty.
Parameters:
none
Return Value:
true if the queue is empty; false if the queue is nonempty.

size
Description:
Returns the number of elements in the queue.
Parameters:
none
Return Value:
The current length of the queue.

operator<<
Description:
Prints the elements of the queue separated by spaces.
Parameters:
A queue to print and a stream to print to.
Return Value:
ostream& - reference to the modified stream.
Remarks:
If the queue is empty nothing is printed.

class PQueue Specification

Priority queue is a queue where elements are arranged by their priorities from highest (in the front of the queue) to lowest (in the back of the queue). The class provides an access to the front element, which is of the highest priority. New elements can be added to PQueue and the front element of PQueue can be inspected and removed.

The class defines the following operations:

push
Description
Adds an element to the queue based on the priority of the element from operator<.
Parameters:
Element to insert to the queue.
Return Value:
void
Remarks:
The front of priority queue is occupied by the element with highest priority in the container.

pop
Description:
Removes an element from the front of the queue.
Parameters:
none
Return Value:
void
Remarks:
The queue must be nonempty to apply the operation. If the queue is empty when pop operation is applied, the behavior of the function is undefined.

front
Description:
Returns a const reference to the element with the highest priority.
Parameters:
none
Return Value:
A const reference to the element with highest priority.
Remarks:
The queue must be nonempty to apply this operation. If the queue is empty when front operation is applied, the behavior of the function is undefined.

empty
Description:
Tests if the queue is empty.
Parameters:
none
Return Value:
true if the queue is empty; false if the queue is nonempty.

size
Description:
Returns the number of elements in the queue.
Parameters:
none
Return Value:
The current length of the queue.

operator<<
Description:
Prints the elements of the queue separated by spaces.
Parameters:
A queue to print and a stream to print to.
Return Value:
ostream& - reference to the modified stream.
Remarks:
If the queue is empty nothing is printed.

class Restaurant Specification

This class represents an entity of type restaurant; includes a menu served to the customers and maintains three queues: a queue of people outside the restaurant waiting to be seated, a queue of customers waiting at the tables for a waiter to take their order and a queue of customers waiting for a check.

The class defines the following members:

Constructor	
Restaurant(const char* tfile, const char* mfile)	
Description:	
Creates a restaurant object.	
Parameters:	
tfile	Name of a file specifying tables' layout at the restaurant. File format: The first line specifies an integer - total number of tables. Every consecutive line specifies a string and an integer - table name and maximum number of people that can sit at the table. For example: 3 B 8 H 6 A 2 Meaning, a restaurant created with this file will have 3 tables with names B, H and A and max capacities 8, 6, and 2. Assumptions: 1. The file always exists and there is no problem opening it. 2. There are no empty lines in the file. 3. The format is always correct: 3.1. The first line is a positive integer n, then there are exactly n lines of the specified format "string integer". 3.2. The string(table name) is not empty("") and contains exactly one word. 3.3. Table name is a unique, case sensitive identifier of a table, meaning - it can appear in the file only once (there are no two tables with the same name). 3.4. The integer(table max capacity) is positive. 4. The words in the file are separated by spaces.

mfile	Name of a file specifying a menu. File format: A string - dish category and an integer - number of dishes in that category. List of dishes in the category, each dish on a separate line. Each line is a string - dish name and a double - dish price. There are total of 6 categories, which appear in the specified order: Appetizer, Main, Dessert, SoftDrink, HotDrink, Alchohol. For example: Appetizer 1 k 24.5 Main 1 f 64.0 Dessert 0 SoftDrink 1 o 10 HotDrink 0 Alchohol 2 s 45 t 65 Assumptions: 1. The file always exists and there is no problem opening it. 2. There are no empty lines in the file. 3. The format is always correct: 3.1. There are exactly 6 dish categories and they appear in the specified order in the format described above. For example, “Main” appears as “Main” and not as “MAIN”, “MaIN” or any other combination. 3.2. Number of dishes in each category is non negative and matches the number of lines that appear afterwards. 3.3. The string(dish name) is not empty(“”) and contains exactly one word. 3.4. Dish name is a unique, case sensitive identifier of a dish, meaning - it can appear in the file only once (there are no two dishes with the same name). 3.5. The price of a dish is positive. 4. The words in the file are separated by spaces.

Remarks
Since you are not required to use exceptions in this exercise, and the right way to report an error in memory allocation is to throw an exception, you can assume that any allocations you perform in the constructor succeed.

Member function	
RSResult AddToWaitList(int nGroupSize)	
Description:	
Adds a group of people to the queue outside the restaurant. The groups join the queue in the order of their arrival.	
Parameters:	
nGroupSize	Number of people in the group.
Return value:	
RSResult	Defined in Report.h. Function result depending on the outcome of the operation. Possible values: (in order of precedence) ILLEGAL_GROUP_SIZE - nGroupSize is non positive or exceeds the maximum capacity of the restaurant tables. For example, you cannot let a group of size 12 wait to enter a restaurant with table sizes of 4 and 8. SUCCESS

Member function	
RSResult SitAtTable()	
Description:	
Assigns a table to a group of people at the head of the waiting queue outside the restaurant. The function fails if the outside queue is empty or if there are no free tables in the restaurant. The assigned table is a free table of minimum capacity that can accommodate the group. If there is more than one such table, a table with the lowest name will be assigned. For example: Assume a restaurant has the following tables: A 2; B 2; C 3; D 6; E 8; and there is a line of groups outside of the following sizes: 2 4 6. Then after 3 consecutive calls to SitAtTable, the first group of size 2 will be assigned table A, second group of size 4 will be assigned table D, and the third group will be assigned table E. Once a group was assigned a table, that table/group moves to a queue waiting for service. Tables are served in order determined by group size and its time of getting the table. The larger the group, the higher its precedence in the waiting queue. Groups of the same size are served according to their times of getting the table, the earlier the group arrived the earlier it is served.	
Parameters:	
none	
Return value:	
RSResult	Defined in Report.h. Function result depending on the outcome of the operation. Possible values: (in order of precedence) NO_WAITING_GROUP - There is no one in the queue outside the restaurant. NO_FREE_TABLE - There are no free tables in the restaurant. SUCCESS

Member function	
RSResult GetOrder(int nSize, const string* items, const int* itemsNum)	
Description:	
Takes an order from a table/group at the head of the queue waiting for service. The function fails if an illegal order is requested or if there is no one waiting for service. Once a table was served, it moves to a queue waiting for a check. Order of tables waiting for a check is determined by the group's size and its time of entering the queue. The larger the group, the higher its precedence in the check queue. Groups of the same size will be served according to their times of joining the check queue.	
Parameters:	
nSize	Number of DIFFERENT dishes in the order; must be positive.
items	An array of size nSize containing names of the ordered dishes.
itemsNum	An array of size nSize containing quantities of the ordered dishes in the same order as they appear in items.
Return value:	
RSResult	Defined in Report.h. Function result depending on the outcome of the operation. Possible values: (in order of precedence) BAD_PARAMETER - NULL pointer passed. ILLEGAL_ORDER - Order size is smaller than one, OR any dish quantity is smaller than one, OR there is no such dish on the menu. SERVED_LIST_IS_EMPTY - There is no one waiting for service. SUCCESS
Example:	
const string items[] = {"Coffee", "IceCream"}; const int itemsNum[] = {1, 10}; Restaurant r("tables.in", "menu.in"); r.GetOrder(2, items, itemsNum); Performs an order of size two; ordering one coffee and 10 ice creams.	

Member function	
RSResult GetCheck()	
Description:	
Brings a check to a table at the head of the queue waiting for checks. The function fails if there is no one waiting for a check.	
Parameters:	
none	
Return value:	
RSResult	Defined in Report.h. Function result depending on the outcome of the operation. Possible values: (in order of precedence) CHECK_LIST_IS_EMPTY - There is no one waiting for a check. SUCCESS

Member function
`void PrintWaitList() const`
Description:
Prints a queue of numbers corresponding to group sizes waiting to enter the restaurant. The order of the printed queue must reflect the correct order of admittance to the restaurant. For example: 2 3 3 4 5 2. The numbers correspond to group sizes (from left to right). To print the queue use the function `void ReportWaitingList(ostream& s, const Queue<int>& line)` which is supplied to you (see Report.h for function definition).
Parameters:
none
Return value:
none
Remarks:
If the queue is empty, pass an empty `Queue<int>` parameter to the function ReportWaitingList. For example: `ReportWaitingList(cout, Queue<int>());`

Member function
void PrintServedList() const
Description:
Prints a queue of strings corresponding to table names waiting to be serviced. The order of the printed queue must reflect the correct order of service in the restaurant. For example: H A S D G T. The strings correspond to table names (from left to right). To print the queue use the function void ReportServedList(ostream& s, const Queue<string>& line) which is supplied to you (see Report.h for function definition).
Parameters:
none
Return value:
none
Remarks:
If the queue is empty, pass an empty Queue<string> parameter to the function ReportServedList. For example: ReportServedList(cout, Queue<string>());

Member function
void PrintCheckList() const
Description:
Prints a queue of strings corresponding to table names waiting for a check. The order of the printed queue must reflect the correct order of service in the restaurant. For example: H A S D G T. The strings correspond to table names (from left to right). To print the queue use the function void ReportCheckList(ostream& s, const Queue<string>& line) which is supplied to you (see Report.h for function definition).
Parameters:
none
Return value:
none
Remarks:
If the queue is empty, pass an empty Queue<string> parameter to the function ReportCheckList. For example: ReportCheckList(cout, Queue<string>());

Member function
`void PrintOpenOrders() const`
Description:
Prints all the opened orders. An open order is an order that was taken from a table, and this table didn't get a check yet. The orders must be printed in the same order they will be paid for (get a check for...) Each order must include the following information: a list of ordered dishes, their price and quantities. To print this information use the class RestaurantOrder that is supplied to you. You may use this class just for printing purposes or/and in your implementation of the exercise. To print the orders use the function `void ReportOrderList(ostream& s, const Queue<RestaurantOrder>& orders)` which is supplied to you (see Report.h for function definition and RestaurantOrder.h for class RestaurantOrder definition).
Parameters:
none
Return value:
none
Remarks:
If there are no opened orders, pass an empty Queue<RestaurantOrder> parameter to the function ReportOrderList. For example: `ReportOrderList(cout, Queue<RestaurantOrder>());`

Member function
void PrintProfits() const
Description:
Prints restaurant profits. Profits are obtained from orders; the ones that were paid for (tables that already got a check) and the ones that are still opened. Profits must include the following information: a list of ordered dishes, their price and quantities. To print this information use the class RestaurantOrder that is supplied to you. You may use this class just for printing purposes or/and in your implementation of the exercise. To print the profits use the function void ReportProfit(ostream& s, const RestaurantOrder& order) which is supplied to you (see Report.h for function definition and RestaurantOrder.h for class RestaurantOrder definition).
Parameters:
none
Return value:
none
Remarks:
If there are no profits, pass a RestaurantOrder parameter with 0 dishes to the function ReportProfit. For example: ReportProfit(cout, RestaurantOrder(“”, NULL, 0));

Member function
`void PrintTables() const`
Description:

Prints information about restaurant tables, sorted first by table capacity in ascending order and then by table name in ascending order.
 	
For each table the following information must be printed: table name, table capacity, table status (one of the TABLE_FREE, TABLE_WAIT_SERVICE, TABLE_WAIT_CHECK; defined in Report.h), number of people siting by it.
 	
To print information about one table use the function
`void ReportTable(ostream& s,`
`const string& name,`
`int cap,`
`RSTableStatus st,`
`int numPeople = 0)`
which is supplied to you (see Report.h for function definition).

Parameters:
none
Return value:
none

מבוא לתכנות מערכות

סמסטר אביב 2007

תרגיל בית מספר 4 - C++

שם פרטי: _____

שם משפחה: _____

ת.ז.: | | | | | | | | |

t2/stud1 login : _____

פקולטה: _____

שם פרטי: _____

שם משפחה: _____

ת.ז.: | | | | | | | | |

t2/stud1 login : _____

פקולטה: _____

מספר הנקודות שהורדו בבדיקה הידנית
