**חלק יבש**

שרה גריפית + ליאור בר יוסף

341312304 + 207022443

סעיף א':

לא ניתן להסתפק בלהגדיר את פעולות האירטור הרגיל כ-const משום שיש מתודות ופונקציות אשר משנות את התור.

ה-ConstIterator הוא איטרטור אשר מצביע על אובייקט const. כלומר, הוא מצביע על אובייקטים אשר לא ניתן לשנות את ערכם או האיברים בהם.

האיטרטור הרגיל מאפשר שינויים על האיברים שהוא מצביע עליהם. זהו מתאים עבור מקרים שבהם ניתן ואף רצוי לשנות את התור, ולכן התור לא יהיה const.

אך אם התור כן const, אז אסור לשנות אותו, ולכן הConstIterator שומר על כך שלא יבצעו בתור שינויים. ולכן צריך גם איטרטור רגיל וגם ConstIterator.

סעיף ב':

קיימות לא מעט מתודות ופונקציות בממשק התור שעליהם יש הנחות על הטיפוס הטמפלייטי:

**הConstructor של הטיפוס הטמפלייטי:**

התור מבוסס על רשימה מקושרת של אובייקטים ממחלקת Node. כל node מכיל משתנה מהטיפוס הטמפלייטי.

ולכן ביצירת ה-nodes, הconstructor או הCopy Constructor של הnodes נקרא, ומשם נקראה הconstructor של הטיפוס הטמפלייטי.

יש כאן הנחה שאם יש צורך בכך, יש לטיפוס הטמפלייטי constructor לא דיפולטי מתאים, וכן אם יש צורך **בהקצאת זיכרון** עבורו, אז זהו נעשה בconstructor שלו.

הכרזה על משתני מחלקת Node:

class Node{  
public:  
 T m\_data;  
 Node\* m\_next;

רשימה של פונקציות ומתודות שיוצרות NODES ולכן מסתמכות על ההנחה הזו:

* ה-constructor של Queue בשורות:

m\_node = new Node;

Node \*nodeNew = new Node;

* הcopy-constructor – מתודת Queue(const Queue& original) בשורות:

m\_node = new Node;

Node \*nodeNew = new Node;

**הDestructor של הטיפוס הטמפלייטי:**

כל אחד מהnodes מכיל משתנה מהטיפוס הטמפלייטי.

כאשר מוחקים את הnodes, הdestructor של הטיפוס הטמפלייטי נקרא גם הוא. יש כאן יש הנחה, שאם הוקצה זיכרון עבור יצירת הטיפוס הטמפלייטי, אזי **הזיכרון משוחרר** כמו שצריך כאשר נקרא הdestructor שלו.

רשימה של פונקציות ומתודות שמשחררות את הזיכרון של Node ולכן מסתמכות על ההנחה הזו:

* בConstructor של Queue בשורות:

delete m\_node;

delete nodeNew;

* הCopy Constructor של Queue בשורות:

delete m\_node;

delete nodeNew;

* המתודה destroyNode() בשורה:

delete node;

**הAssignment Operator של הטיפוס הטמפלייטי:**

יש מתודה אחת אשר משתמשות בפעולת השמה עבור הטיפוס הטמפלייטי.

ההנחה כאן היא **שקיימת** מתודה כזו עבור הטיפוס הטמפלייטי, ושהיא מבצעת **העתקה** של האובייקט (ולא למשל העברת פוינטרים בלבד) **ומקצה זיכרון** עבורו במקרה הצורך.

* מתודת pushback() בשורה:

endIt.m\_currentNode->m\_data = data;

**הטיפוס הטמפלייטי מאותחל לערך תקין ולא NULL:**

יש כמה פונקציות ומתודות בהן מוחזר reference לאובייקט מהטיפוס הטמפלייטי.

ההנחה כאן היא שהאובייקט הזה מאותחל ו**איננו NULL**, משום שreference לא יכול להיות NULL. כלומר, בשום מקום בתכנית, האובייקט הזה לא הופך לNULL: ביצירה שלו, בשינוי שלו בפונקצית transform וכו'...

רשימה של פונקציות ומתודות שמניחות שהטיפוס הטמפלייטי תקין:

* מתודת front() בשורות:

T& Queue<T>::front()

return m\_node->m\_next->m\_data;

* מתודת front() constבשורות:

const T& Queue<T>::front() const

return m\_node->m\_next->m\_data;

* פונקציית filter() בשורות:

for (const T& elem : givenQueue) {

* פונקציית transform() עבור השורות:

for (T& elem : givenQueue) {

* המתודה operator\* בבשורות הבאות:

T& Queue<T>::Iterator::operator\*() const

return m\_currentNode->m\_data;

* המתודה operator\*

const T& Queue<T>::ConstIterator::operator\*() const

return m\_currentNode->m\_data;

**הנחות על הטיפוס הטמפלייטי עבור המצביע לפונקציה/function object:**

* פונקציית filter():

if (c(elem)) {

ניתן לראות שאובייקט const מהטיפוס הטמפלייטי נשלח לפונקציית c מטיפוס טמפלייטי שני אשר מתקבלת כקלט של הפונקציה filter. ההנחה כאן היא שהפונקציה c **מקבלת** **אובייקט מהטיפוס הטמפלייטי** הראשון, **איננה משנה** את האובייקט וכן **מחזירה ערך בולייאני**.

* פונקציית transform()

c(elem);

ניתן לראות שאובייקט מהטיפוס הטמפלייטי נשלח לפונקציית c מטיפוס טמפלייטי שני אשר מתקבלת כקלט של הפונקציה. ההנחה כאן היא שהפונקציה c **מקבלת** **אובייקט מהטיפוס הטמפלייטי** הראשון.

סעיף ג':

Text

Description automatically generatedכאשר הסטודנט ינסה לקמפל את התרגיל כשמחלקת הQueue שלו ממומש בקבוץ “.cpp”, אז נזקרת שגיאה של **"Undefined Reference"**:

השגיאת קומפילציה הזו מתרחשבת **בשלב הקישור** (linking process), והיא נזרקת כאשר יש פנייה לאבוייקט כלשהו (מחלקה, פונקציה, משתנה, וכו'...) והלינקר בשלב הקומפילציה לא מוצא את ההגדרה שלו.

במקרה שלנו:

עבור מחלקת Queue, השתמשנו ב**template**.

ההחלפה של הטיפוס הטמפלייטי קורה בזמן הקומפילציה. כלומר, הקומפיילר "יוצר עותק" של המחלקה לפי הטיפוס הטמפלייטי הנדרש בתכנית. לכן הקומפיילר צריך גישה ישירה לכל המחלה (כולל מימושה) כדי להתאים אותה לטיפוס הנדרש.

יש לקומפיילר גישה לקבוץ ה".h", ולכן בתוך קובץ זה צריך להיות כל המידע על המחלקה הטמפלייטי. **אם מימוש המחלקה ממוקם** **מחוץ לקובץ ה".h",** **אין לקומפיילר גישה אליו**. הקומפיילר לא יצליח **לקשר** בין המחלקה לבין המימוש שלה, וכתוצאה מכך, הקומפליציה נכשלת ונזרקת השגיאה המתאימה"Undefined Reference".

לכן עבור templates, צריך למקם את מימוש המחלקה בתוך קובץ ה".h" ולא בתוך קובץ ה".cpp".

סעיף ד':

הפונצקיית filter() במחלקת Queue מקבלת שני פרמטרים: const reference לאובייקט מטיפוס queue, וגם object function או מצביע לפונקציה. היא עוברת על כל האיברים בתור, ובודקת אם האיברים עונים על התנאי של הobject function/מצביע לפונקציה הניתן כפרמטר. אם כן, אז האיברים נכנסים לתור חדש, אשר הוא הערך המוחזר מהפונקציה filter().

הסטודנטית רוצה להשתמש בפונקציה הזו כדי לסנן תור של מספרים שלמים כך שישארו רק מספרים המתחלקים במספר כלשהו שאינו ידוע בזמן הקומפילציה (אלא רק בזמן ריצה).

היא יכולה לעשות זאת באמצעות השלבים הבאים:

* בניית function object:

הסטודנטית צריכה לבנות מחלקה חדשה, אשר מכילה member אחד אשר הוא המספר שבו היא רוצה לחלק את האיברים בתור.

המחלקה תכיל מתודה אחת אשר overloads את operator(), שבודקת אם הקלטים המתקבלים מתחלקים בmember של האובייקט עליו נקראת המתודה.

class DividesBy {

public:

//Constructor

DividesBy(int divisor) : m\_divisor(divisor) {}

//Default copy constructor, assignment operator and destructor

DividesBy(const DividesBy&) = default;

DividesBy & operator=(const DividesBy& other) = default;

~ DividesBy() = default;

//operator() overloading

bool operator()(int dividend) const

{

return (dividend % m\_divisor == 0);

}

private:

int m\_divisor;

}

נשים לב, שבבניית אובייקט מהטיפוס של המחלקה הזו נקראת הconstructor (לדוגמא *DividesBy firstDivision(5);*), והקלט (בדוגמא שלנו, המספר 5) יהיה המספר שבו רוצים לחלק את האיברים בתור (*m\_divisor = 5*). בקריאת אותו האובייקט שנבנה עם הסוגריים "()"

(*bool result = firstDivision(25);*), זהו בעצם קריאה למתודה operator() על הקלט (כלומר יבדק אם 25 מתחלק ב5, ולכן בresult יהיה true).

* בניית אובייקט תור מטיפוס int (כי היא דורשת שהתור יכיל רק מספרים שלמים), והכנסה לתור מספרים שלמים לפי צרכיה, כמו בדוגמא למטה:

Queue<int> queue1;

queue1.pushBack(564);

* השמה לתוך התור המקורי את תוצאת הפונקציה filter(), כאשר לפונקציה ניתן אותו התור כפרמטר ואת ה-function object שנבנה בשלב קודם.

הfunction object שישלח כפרמטר יקבל כערך התחלתי את המספר שצריך לחלק בו את כל האיברים בתור. הערך ההתחלתי יכול להיות משתנה שנקבע בתכנית של הסטודנטית בשלב קודם לכן, ובכך עונה על הדרישה כי המספר הזה נקבע רק בזמן ריצה.

ההשמה של התוצאה של filter() היא לתוך אותו התור המקורי, משום שהסטודנטית רוצה לסנן את התור המקורי ובעצם לשנות אותו. ההשמה תעתיק את התור החדש שנוצר בfilter() לתוך התור המקורי, ובכך הסטודנטית תישאר עם התור המקורי המעודכן.

queue1 = filter(queue1, DividesBy(divisor))

להלן התכנית לדוגמא:

**מחלקת DividesBy בקובץ "DividesBy.h":**

#ifndef QUEUE\_H\_DIVIDESBY\_H  
#define QUEUE\_H\_DIVIDESBY\_H  
  
/\*  
 \* DividesBy:  
 \* This class builds function objects that check if the input integer divides by a given number  
\*/  
class DividesBy {  
public:  
 //Constructor  
 DividesBy(int divisor) : m\_divisor(divisor) {}  
  
 //Default copy constructor, assignment operator and destructor  
 DividesBy(const DividesBy&) = default;  
 DividesBy & operator=(const DividesBy& other) = default;  
 ~ DividesBy() = default;  
  
 //operator() overloading  
 bool operator()(int dividend) const  
 {  
 return (dividend % m\_divisor == 0);  
 }  
private:  
 int m\_divisor;  
};  
  
#endif //QUEUE\_H\_DIVIDESBY\_H

**התכנית הראשית בקובץ "mainQuestionD.cpp":**

#include "Queue.h"  
#include "DividesBy.h"  
#include <iostream>

int main () {  
 Queue<int> queue1;  
 for (int i = 1; i <= 100; i++) {  
 queue1.pushBack(i);  
 }  
 int queueSize = queue1.size();  
 std::cout << "Before changes, the queue's size is: " << queueSize << std::endl;  
 std::cout << "The numbers in queue: " << std::endl;  
 for (int& elem : queue1) {  
 std::cout << elem << std::endl;  
 }  
  
 //Can be decided upon during the student's program  
 int divisor = 5;  
  
 queue1 = filter(queue1, DividesBy(divisor));  
  
 queueSize = queue1.size();  
 std::cout << "After changes, the queue's size is: " << queueSize << std::endl;  
 std::cout << "The numbers in queue: " << std::endl;  
 for (int& elem : queue1) {  
 std::cout << elem << std::endl;  
 }  
   
 //Continuation of student's program  
  
 return 0;  
}

**התוצאה של התוכנית:**

אכן התור סונן בהצלחה!

*Graphical user interface, text

Description automatically generated-(all the numbers between 1 and 100)-*

Text

Description automatically generated