פתרון חלק יבש תרגיל בית 3

5 חלק יבש

בחלק זה מספר שאלות הנוגעות לממשק התור ולשימושים בו. עליכם לענות על השאלות בקובץ הנקרא dry.pdf.

א. על מנת לאפשר איטרציה על תור קבוע (const), הממשק מאפשר לנו שימוש ב-ConstIterator בנוסף לאיטרטור הרגיל של התור. מדוע לא ניתן להסתפק בלהגדיר את פעולות האיטרטור הרגיל כ-const?

פתרון:

על מנת לענות על השאלה בצורה ברורה, נבין את המשמעות של ConstIterator לעומת Iterator. ב– ConstIterator , אנו לא מאפשרים לבצע שינויים לערכים שעליהם האיטרטור מבציע, לעומת Iterator רגיל, שהם כן מאפשרים אפשרות כזאת. אילו היינו מגדרים את פעולות האיטרטור הרגיל כ –const , איטרטורים רגילים לא היו יכולים לבצע שינויים לערכים שהם מבצעים עליהם. ולכן, במקרה שלנו, אילו היינו מגדירים את פעולות האיטרטור הרגיל כ –const , איטרטור שמבציע על תור שאינו const , לא יוכל לבצע עליו שינויים, וזה דבר שכן היינו רוצים לאפשר.
במילים אחרות, Iterator אמור לשמש אותנו לכתיבה וקריאה, ולכן לא נוכל להגדיר את פעולות האיטרטור הרגיל כ – const , לעומת ConstIterator , שאמור לשמש אותנו לקריאה בלבד, ללא ביצוע שינויים לערכים שעליהם ה ConstIterator מצביע.
נסתכל על הדוגמאות הבאות מהקוד שלנו, שמראה את השינוי באופרטור *, בין Iterator ל ConstIterator:

עבור Iterator:

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

במקרה זה, אנו רוצים שע"י שימוש באופרטור *, שמחזיר את הערך של התור ב –index מסויים , יוחזר הערך by reference , על מנת לאפשר שינוי שלו.

עבור ConstIterator:

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

במקרה זה, לאחר שימוש באופרטור *, שמחזיר את הערך של התור ב –index מסויים, יוחזר הערך by reference אבל ב const , על מנת **לא** לאפשר שינוי שלו.

בנוסף, אם ננסה להגדיר Const Iterator , לא נוכל לקדם את האיטרטור עצמו, ולכן לא נוכל לעבור על אברי התור.

ב. באילו מהפונקציות בממשק התור קיימות הנחות על הטיפוס הטמפלייטי? עבור כל אחת מהפונקציות הללו פרטו את הנחות.

פתרון:

בפונקציות הבאות בממשק התור קיימות הנחות על הטיפוס הטמפלייטי:

1. Constructor - ב Constructor שלנו אנו מבצעים: m_data(new T[INITIAL_SIZE]),
בעת שימוש ב new, עבור כל אחד מהמשתנים מסוג T שנוצרים, אנו קוראים ל default constructor, על מנת ליצור אותו, ולכן הטיפוס הטמפלייטי T יצטרך **default constructor**, כלומר בנאי חסר פרמטרים, או בנאי עם פרמטרים בעל ערכים דיפולטיים.
2. Copy Constructor – ב Copy Constructor אנו משתמשים ב new באופן זהה לשימוש בסעיף 1, ולכן גם כאן, הטיפוס הטמפלייטי T יצטרך **default constructor**, כלומר בנאי חסר פרמטרים, או בנאי עם פרמטרים בעל ערכים דיפולטיים.
בנוסף לכך, אנו משתמשים באופרטור השמה בין שני טיפוסים טימפלטיים T, ולכן כתוצאה מכך, T ייצטרך גם **אופרטור השמה.**
3. =Operator - באופן זהה ל Copy Constructor , אנו משתמשים ב - new ולכן אנו נטרך **default constructor**, כלומר בנאי חסר פרמטרים, או בנאי עם פרמטרים בעל ערכים דיפולטיים, בנוסף לכך, אנו משתמשים באופרטור השמה בין שני טיפוסים טימפלטיים T, ולכן כתוצאה מכך, T ייצטרך גם **אופרטור השמה.**
4. PushBack – בפונקציה pushBack, אנו משתמשים באופרטור השמה בין שני טיפוסים טימפלטיים T, ולכן כתוצאה מכך, T ייצטרך גם **אופרטור השמה.** בנוסף לכך, אנו משתמשים ב - new ולכן אנו נטרך **default constructor**, כלומר בנאי חסר פרמטרים, או בנאי עם פרמטרים בעל ערכים דיפולטיים.
5. PopFront - אנו משתמשים באופרטור השמה בין שני טיפוסים טימפלטיים T, ולכן כתוצאה מכך, T ייצטרך גם **אופרטור השמה.**
6. Filter – הפונקציה filter משתמשת ב pushBack, ולכן בפרט נצטרך **אופרטור השמה.** בפרט, משתמשים באיטרציה על איברי המערך , באמצעות איטרטור , ולכן נצטרך אופרטור השמה.
בנוסף, הפונקציה filter מבצעת בדיקה על איברי המערך באמצעות התנאי שהיא מקבלת, ולכן בפרט נצטרך **אופרטור ==** , שבודק האם איברי המערך מקיימים את התנאי או לא מקיימים אותו. (נשים לב, שהפונקציה של התנאי שמועברת היא באחריות המשתמש, ולכן, ההנחה היא שמשתמשים שם באופרטור ==, במידה ומשתמשים בדברים נוספים או אחרים, אין ביכולתנו לדעת זאת, כלומר אנו רק צריכים שה –function operator / פונקציה תעבוד עם הטיפוס הטמפלייטי).
בנוסף, אנו נצטרך שה Condition שלנו יהיה לו **אופרטור סוגריים,** כלומר למצביע לפונקציה או ל function object, על מנת שנוכל לבדוק אותו על ה data שלנו. (זוהי לא הנחה על הטיפוס הטימפלייטי של התור, אלא על ה function object או הפונקציה שמתקבלת).
7. Transform – הפונקציה transform מבצעת איטרציה על איברי המערך, באמצעות איטרטור , ולכן נצטרך **אופרטור השמה.** בפרט , משתמשים באופרציה שמשנה את איברי המערך , ולכן נצטרך גם שם אופרטור השמה.(נשים לב, שהפונקציה של השינוי שמועברת היא באחריות המשתמש, ולכן, ההנחה היא שמשתמשים שם באופרטור השמה, במידה ומשתמשים בדברים נוספים או אחרים, אין ביכולתנו לדעת זאת, כלומר אנו רק צריכים שה –function operator / פונקציה תעבוד עם הטיפוס הטמפלייטי).
בנוסף, אנו נצטרך שה Transform שלנו יהיה לו אופרטור סוגריים, כלומר למצביע לפונקציה או ל function object, על מנת שנוכל להפעיל אותו על האיברי התור data. (זוהי לא הנחה על הטיפוס הטימפלייטי של התור, אלא על ה function object או הפונקציה שמתקבלת).

ג. סטודנט בקורס מבוא לתכנות מערכות שכח מהאזהרות שקיבל בתרגול ומימש את המחלקה Queue בקובץ cpp במקום בקובץ h. מהי השגיאה שיקבל כאשר ינסה לקמפל את התרגיל ובאיזה משלבי הקומפילציה היא מתרחשת?

פתרון:

הקומפיילר משתמש בתבניות על מנת ליצור קוד לפי הצורך, תהליך יצירת מופע של הקוד המוגדר על ידי התבנית מתבצע בזמן קומפילציה.
לפיכך, המהדר צריך לקבל גישה ליישום המתודות, כדי להפעיל אותן עם ארגומנט התבנית. במידה וזה לא היה בקובץ header, הן לא היו ניגשות, ולכן לא תהיה אפשרות ליצור את התבנית.
עבור הקומפיילר , זו לא בהכרח שגיאה, מאחר והפונקציות עלולות להיות מוגדרות במקום אחר, שבה ה linker ימצא אותם. ולכן, אם ה linker לא יימצא את הקוד הזה, הוא מעלה שגיאה. (כלומר אנו מקבלים שגיאה בשלב ה linking , משגיאה שנוצרה בתהליך הקומפילציה).

ד. סטודנטית בקורס מבוא לתכנות מערכות סיימה לפתור את תרגיל בית 3, והחליטה להשתמש במימוש התור מהתרגיל לפרוייקט צד שהיא מפתחת בשעות הפנאי. במימוש פרוייקט הצד הסטודנטית נדרשה לסנן תור של מספרים שלמים, כך שישארו בתור רק מספרים המתחלקים במספר כלשהו **שאינו ידוע בזמן קומפילציה** אלא רק בזמן ריצה. הסבירו כיצד ניתן לממש את הפונקציונליות הדרושה בעזרת הפונקציה filter.

פתרון:
על מנת לממש את הפונקציונליות הדרושה בעזרת הפונקציה filter, על הסטודנטית להשתמש ב function object. לשם כך, היא תצטרך להגדיר class DivideBy, כך שבזמן ריצה, נוכל ליצור ממנו אובייקט עם המספר שאותו נרצה לבדוק אם המספרים בתור מתחלקים בו, שנסמנו ב – x, ואז נוכל באמצעות filter, שיכולה לקבל function obejct, לסנן את התור כך שרק מספרים המתחלקים ב x ישארו בו.
לשם הבהרת הפתרון נסביר זאת בנוסף ע"י קוד:

```
class DivideBy{

public:
    DivideBy(int divisor) : m_divisor(divisor) {}
    bool operator()(int number) const {
        return number % m_divisor == 0;
    }
private:
    int m_divisor;

};
```

המחלקה DivideBy , ממומשת באופן מקוצר רק בשביל המחשה.
הסטודנטית תצטרך ליצור אובייקט עם המספר x המתואר , כלומר DivideBy divisor(x);
לאחר מכן, filter תקבל את התור של המספרים השלמים שבידי הסטודנטית , ואת divisor , ובאמצעותם היא תוכל לסנן את התור ולקבל תור חדש כנדרש.