

תכנות מתקדם 1

מטרות התרגיל

היכרות עמוקה יותר עם שפת, שיפור יכולות עיצוב קוד. לימוד עצמי.

תרגיל ראשון

ירושה, פולימורפיזם, העמסת אופרטורים, לימוד עצמי של אלגוריתם, STL

# הנחיות כלליות

* עבודה ביחידים בלבד.
* אסור לתכנית שכתבתם לקרוס משום סיבה.
* אין לבצע הדפסות מיותרות למסך.
* אם המשתמש הכניס קלט שאינו תקין יש להודיע לו על כך עם Exception. (try catch block and throw)
* יש להימנע ככל הניתן משכפול קוד.
* יש להפריד את התכנית באופן חכם לקבצי הצהרה וקבצי מימוש. אין לממש פונקציות שאינן בסיסיות בקובץ הצהרה.
* במידה ואתם כותבים קוד שאינו פשוט, יש להוסיף שורת תיעוד. אין לתעד קוד מובן מאליו.
* כל שגיאה שזורקים צריכה להיות מסוג const char\* (throw “bad input”) בmain.cpp יש דוגמאות.
* יש לעקוב אחרי מסמך הקונבנציות שפורסם.
* ניתן להניח הנחות מפשטות במידה ואין פגיעה בדרישות התרגיל.
* אין לבזבז משאבי מערכת – ובפרט, לשחרר זיכרון מוקצה.
* הפעילו שיקול דעת.

# הוראות הגשה

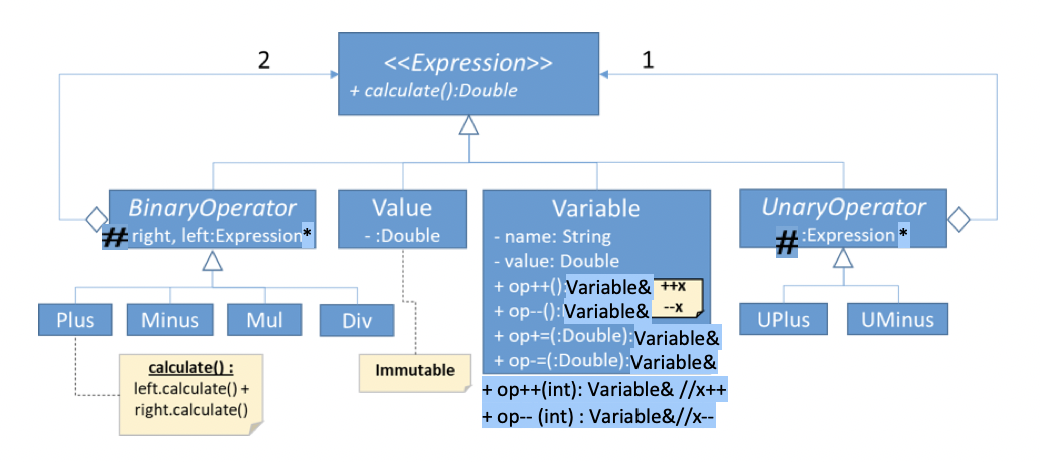
1. יש להגיש קובץ ZIP (אך ורק ZIP ולא שום פורמט מקובץ אחר) בשם ex1.zip.
2. קובץ ה-ZIP יכיל קובץ בשם **details.txt**, ואת כל שאר הקבצים .
3. הקובץ details.txt יכיל שתי שורות בדיוק – בשורה הראשונה תעודת הזהות של הסטודנט המגיש ובשורה השנייה השם המלא שלו באנגלית.
4. **נא לא להגיש את הקובץ Expression.h** שאותו קיבלתם בתרגיל.
5. אתם יכולים לממש .cpp ו.h לכל מחלקה ולהגיש את כולם, או לשים הכל בex1.h וex1.cpp
6. כל הקבצי הגשה: **ex1.h, ex1.cpp, details.txt**

וכל קובץ אחר שאתם צריכים כדי שיעבוד לכם, כלומר עם הmain.cpp.

1. **אין להגיש קובץ main!**
2. מצורף קובץ main.cpp לדוגמא.
3. על התרגיל להתקמפל ולעבוד ביחד עם קובץ הmain שסופק והקבצים הנוספים שסופקו לכם. התרגיל יקומפל בסביבת Linux עם Compiler שתומך **בסטנדרט C++11.**
4. צריך שהתכנית שלכם תתקמפל בלי הערות. יש להתייחס להערות כמו אל שגיאות.

חלק 1 – ירושה, פולימורפיזם ואופרטורים

נתון ה class diagram הבא:



**Notice the new changes**

Overloaded operators need to return Variable&, and will operate on value. += and -= will only accept a double (as the diagram shows). Notice that ++ & -- postfix will behave the same as the prefix ones.

Binary and Unary Operator have **protected Expression \***

**UPlus returns the value inside the input. Meaning calling calculate() is enough.**

The UMinus returns the (-) of the expression value that is passed in.

All derived classes from BinaryOperator and UnaryOperator (Plus, Minus, UPlus, etc) Will only need to have constructors that accept: Expression\*. This means that each of them will have one constructor.

All destructors must be virtual destructors.

See <https://stackoverflow.com/questions/461203/when-to-use-virtual-destructors>

Expression.h כבר נתון לכם ואסור לשנותו ובפרט אינו להגשה. עליכם לממש את התרשים לעיל במדויק בקבצים ex1.h ו ex1.cpp בהתאמה.

זאת אומרת, כל המחלקות והמימוש שלהן שאינם Expression, נכנס לex1.

בקובץ main.cpp שסופק לכם תוכלו למצוא דוגמאות ריצה. עליכם לגרום לדוגמאות לרוץ ללא שגיאות קומפילציה או ריצה, ולהגיע אל התוצאה הנכונה. למשל:

// x=3

Variable \*x = new Variable(“x”, 3);

++(\*x); // x=4

// -5\*(3.5+x)

Expression\* e=new Mul( new UMinus(new Value(5.0)) , new Plus( new Value(3.5) , x) );

cout<< e->calculate() <<endl; //-37.5

delete e; // משחרר את כל האובייקטים בביטוי זה

חלק 2 – לימוד עצמי של אלגוריתם, שימוש בתור ובמחסנית של STL

בסעיף זה עליכם ללמוד בכוחות עצמכם אלגוריתם פשוט כגון Shunting-yard שהופך ביטוי infix לעץ חישוב ולממש אותו במחלקה Interpreter. הפעולה interpret() תקבל מחרוזת של ביטוי ותחזיר מצביע למופע של Expression. לדוגמה:

Interpreter i;

Expression \*e = i.interpret(“5\*(3+(-4))”);

cout<< e->calculate() << endl; // -5

המחלקה interpreter מיועדת לעבוד גם עם משתנים. העבודה איתם תהיה בצורה הבאה:

Interpreter i;

I.setVariables(“x=4;y=5”); **//setVariables will always be called before interpret.**

Expression \*e = i.interpret(“x\*(y+1)”);

cout<< e->calculate() << endl; // 24

משתנים שהגדרתם בתוך מחלקה אובייקט של interpreter לא רלוונטיים למשתנים שהגדרתם באובייקטים אחרים מסוג Expression (ראו חלק 1 של התרגיל), ולא יהיו רלוונטיים לאובייקטים אחרים מסוג interpreter. הכוונה היא שהמשתנים שאתם מגדירים לא יהיו גלובאליים.

(x above will be a different x in another interpreter)

המחלקה interpreter מיועדת לעבוד גם עם אופרטורים אונאריים. העבודה איתם תהיה בצורה הבאה:

Interpreter i;

Expression \*e = i.interpret(“-(+(1+2))”);

cout<< e->calculate() << endl; // -3

**הכוונה היא שנדרש להשתמש בסוגריים בכל פעם שמשתמשים באופטור אונארי מול משתנה.** כלומר, הדוגמא הבא **לא תקינה** לדרישות התרגיל וצריכה לזרוק exception:

Interpreter i;

Expression \*e = i.interpret(“1-+-+-2”);

**קלט לinterpreter: שימו לב שלא יהיו רווחים בקלטים שנעביר לinterpreter.**

ממשו את המחלקה Interpreter בקובץ ex1.h ו ex1.cpp בהתאמה או בinterpreter.h and interpreter.cpp. בקובץ main.cpp שסופק לכם תמצאו דוגמאות ריצה נוספות. גרמו להם לרוץ ללא שגיאות קומפילציה או ריצה והחזירו את התוצאות הנכונות.

**בדיקה:**

הבדיקה עצמה אוטומטית וחסויה. הבדיקה מכילה בדיקות דומות אך לא זהות לאלו שנמצאות ב main.cpp. אם אין לכם שגיאות ריצה או קומפילציה ב main אז לא צפויות לכם שגיאות כאלו בבדיקה החסויה.

**Validity of input**

**Approach – the approach should be checking what *is* valid, and not what isn’t. Meaning if you know what is the valid form of an input, you don’t need to check all possibilities of what it can’t be, only that if it doesn’t equal what it should, throw an error.**

**Set Variables:**

Input – a valid input will be of the form: <variable\_name>”=”<number>

If there are multiple variables, a valid input has to be of the form:

<variable\_name>”=”<number>”;”<variable\_name>”=”<number>”;”<variable\_name>”=”<number>

Notice at the end of the declaration, it is valid to sometimes include “;”, and sometimes not.

**Variable\_name** – Cannot start with a number, and all other characters must be either: alphabet letters, numbers, or “\_”. Anything else in the variable name must throw an error.

4hello // error

X4 // fine

X4\_5 // fine

X4.0 // error

y$ // error

**Number –** either an int or double.

Examples for input to setVariables

“x4==5” //error

“g.0=67”; //error

“yz=34gh”;//error

“x=1.6;y=4.0”//fine

Anything not of the valid form explained above will throw an error.

**Interpret:**

A valid input must of be of infix form

Each ‘(‘ must have a closing ‘)’ and each ‘)’ must have an opening ‘(‘.

**The following things will always happen:**

( ) will always come after a minus and before a variable name.

-(x), -(x+y)

There will be no ‘-‘ in front of variables.

Examples:

-(x) + 5 //okay

-(+(-(x))) //okay

-(2\*(3+4.2)) //okay

1.0-(-(-4.0)) //okay

56--x7.0 //error

-3 \*x //okay, numbers can have the unary minus in front

-(x+y) // notice y can have a negative value inside. For example, setVariables(“y=-5”).

-3 \*\*x // error

**\*\***

Set variables will always be called before interpret, however, interpret can still contain variables that have not been set, and therefore must throw an exception.

**Please refer to the above examples, as well as the ones in the main. We are not looking to fail you on difficult examples.**