**CSE 465/565**

**Spring 2016**

**Homework #1**

*Instructions: Submit an electronic copy of all questions and programs. The electronic copies should be placed into the course’s Canvas site.*

*Submit to Canvas a single zip file that contains the following directory structure:*

*uniqueidHW1 top-level directory containing all of your stuff*

*uniqueidHW1.pdf ; which contains the answers to #2, #3, and report for #4*

*uniqueIDZPM directory containing your source code*

*main.cpp, helper.cpp, etc ; your source code in either C++ or Java*

*Main.java, Helper.java, etc ; please, no Java packages*

1. Survey completed/submitted

2. (5/5) Answer the following questions regarding class policies. Explain/show your work.  
A) How many points will be deducted if your first homework assignment is submitted exactly 1 hour late?

* 1.2pts will be deducted

B) How many points will be deducted if your first homework assignment is submitted exactly 24 hours late?

* 28.8 (29 according to syllabus) pts

C) Suppose you obtain a grade of 93 on every graded item during the entire semester and do not show up for 3 classes, what will be your final grade?

* 91% assuming you don’t drop us from the class

D) Does the class have a grader? If so, what is their name and email address?

* TA: Doug Blase / [blasedd@miamioh.edu](mailto:blasedd@miamioh.edu)

E) When is the first quiz?

* Tuesday

3. (35/35) The Vandelay Language Company is designing a new language based on Java and their language designers have been debating whether their language should allow variable declarations to occur at arbitrary positions in the source code. For example, allowing arbitrary declarations would allow:

Pros:

- ↑ Readability: Requiring variable declarations at the beginning of a function will increase readability since all variables used in a function will be immediately apparent instead of scattered throughout. In addition, it will also prevent confusion when variables with the same name of different types are declared throughout the function (such as if … var = string else var = int).

- ↓ Writability: Writability will be decreased since programmers won’t be able to create variables ad-hoc (such as for a temporary or in-progress calculation). In addition, writing programs will become more difficult since there is a global scope inside the function and a for/while loop would need its increment variables declared possibly far away from its use.

- ↑ Reliability: This approach is simpler since there is a single scope inside a function. In addition, memory management will likely be easier since memory can be allocated at some time before the function begins (this largely depends on the compiler/interpreter, though). This could make the compiler simpler since memory allocation/offsets can be handled in a single, one-time step.

- Cost: This approach sacrifices writability and benefits of multiple scopes to increase readability and possibly increase reliability (largely dependent on the skill of the programming team).

Cons:

- ↓ Readability: This approach could decrease readability. Even though variables are declared close to use, it’s possible to re-use a variable later on and increase confusion since it’s never being re-declared. In addition, the increase in scopes could allow variables of the same name with different values depending on the scope they’re found in (function-wide i increment in addition to for i increment).

- ↑ Writability: This can greatly increase writability since programmers won’t need to pre-plan functions or continually move up to the top of a function to add additional variables. In addition, this facilitates using temporary variables for complex calculation since they’re easy to create.

- ↓ Reliability: As mentioned, this largely depends on the competence of the programming team creating the compiler/interpreter. The increased use of scopes may make it harder to track which instance of a like-named variable should be used in each location.

- Cost: This approach sacrifices readability and mildly increases complexity. On the other hand, it improves functionality and flexibility due to the addition of scopes, ability to temporarily utilize variables in a very local sense adjacent to relevant code, and allow programmers to decide where to place variables.

Conclusion: 7/10

The inclusion of this feature increases functionality and has many benefits mentioned above. In addition, this gives the programmer the flexibility to write the code as they like. Coding styles/guidelines can dictate the proper placement rather than the technology (compiler/interpreter). This decision ultimately offers more functionality at the cost of not enforcing a particular style.

4. (55/55 points) Consider a very simple programming language named Z+-. The Z+- programming language has the following features:

All the test programs work correctly. I had to remove a couple error checking features such as matching quote detection. For this, the program doesn’t crash but will truncate the last character from the string (because it thinks this should be a “ sign). In addition the interpreter aggressively crashes when there is a type error (try..catch..exit) due to the way errors are passed around with returns and the way variable storage is designed. In addition, the interpreter is pretty generous on variable names and will allow everything except a equals including using “reserved” words as variables (I’ll use reserved loosely here…). This can lead to some pretty unpredictable behavior unless you understand this “feature”.

For instance:

PRINT = 3 ;

55 = 7 ;

ra nd omt hing ^&# $^ @%#$% 34321 = 7 ;

PRINT += 55 ;

PRINT PRINT ;

Yields: (Checks to see if it’s a variable before checking other types)

PRINT=10

Output:

for f in ../scripts/\*.zpm; do echo Executing $f:; ./z+-.exe $f; echo ; done

Executing ../scripts/bonus.zpm:

A=a b cw x y z

Executing ../scripts/prog1.zpm:

a=90

b=360

Executing ../scripts/prog10.zpm:

A=1

a=3

numItems=0

Executing ../scripts/prog2.zpm:

a=94

b=182

a=276

b=182

Executing ../scripts/prog3.zpm:

A=1032

B=31

Executing ../scripts/prog4.zpm:

A=XXXXXXXXXXXXXXXX

Executing ../scripts/prog5.zpm:

A=2034

B=-9117

Executing ../scripts/prog6.zpm:

A=117

B=10

C=3

D=4

E=214

Executing ../scripts/prog7.zpm:

A=96

B=95

Executing ../scripts/prog8.zpm:

A=50001

Executing ../scripts/prog9.zpm:

A=0

a=5

ERROR: Type Error on line 5

->A += "hello" ;

\*\* Didn’t get around to doing the comparison to other compiled/interpretted languages but I’ll take a look at this later and see how it compares to say… C++, Java, Python, and PHP equivalent programs later this weekend