Assignment 2

Graded

Student

Rin Pereira

Total Points

71.5 / 94 pts

Question 1

(no title) **17.5** / 30 pts

1.1 (no title) **2.5** / 3 pts

✔ **– 0.5 pts** Missing/ something wrong
b -> [global char] --> [char local to q]

1.2 (no title) **3** / 3 pts

✔ **– 0 pts** Correct

1.3 (no title) **3** / 3 pts

✔ **– 0 pts** Correct

1.4 (no title) **3** / 3 pts

✔ **– 0 pts** Correct

1.5 (no title) **3** / 3 pts

✔ **– 0 pts** Correct

1.6 (no title) **3** / 3 pts

✔ **– 0 pts** Correct

1.7 (no title) **0** / 6 pts

✔ **– 6 pts** Incorrect

1.8 (no title) **0** / 6 pts

✔ **– 6 pts** Incorrect

Question 2

(no title) **8** / 8 pts

✔ **– 0 pts** Correct

Question 3

(no title) **8** / 8 pts

✔ **– 0 pts** Correct

Question 4

(no title) **6** / 8 pts

– 2 pts Answer needs to provide some explanation for the context in which the compiler will be unable to differentiate between functions, by calling print() with only a single argument

Question 5

(no title) **0** / 8 pts

– 8 pts Incorrect

The compiler / translator can't unambiguously know which function is being referenced by the programmer

Answer does not address the ambiguity issues that can arise from the combination of default arguments and overloading

Question 6

(no title) **8** / 8 pts

– 0 pts Correct

Question 7

(no title) **8** / 8 pts

– 0 pts Correct

Question 8

(no title) **8** / 8 pts

– 0 pts Correct

Question 9

(no title) **8** / 8 pts

– 0 pts Correct

Q1
30 Points

Determine the symbol table and output for the following code. Please note the use of format specifiers and use an ASCII lookup table if necessary.

The expected symbol table format is as follows:
`(identifier) --> [bound attributes] --> [bound attributes]`

Note: the rightmost set of attributes are the visible attributes. The rightmost set shadows all other attribute sets.

```
#include <stdio.h>     // line 1
int a = 101;
char b = 'c';

int q(char b) {
   int a = 81;
   printf("%c\n",a);
   printf("%d\n",b);   // line 7
   return a;
}

void p() {
   double b = 6.2;
   printf("%c\n",a);
   printf("%f\n",b);   // line 13
   a = q(a);
}

int main() {
   char a = 'W';
   printf("%c\n",b);   // line 17
   p();
   return 0;
}
```

Q1.1
3 Points

Using static scoping determine the symbol table at line 7

(a) --> [int, global] --> [int, local to q]
(b) --> [char, global]
(q) --> [int, function]

Q1.2
3 Points

Using static scoping determine the symbol table at line 13

(a) --> [int, global]
(b) --> [char, global] --> [double, local to p]
(q) --> [int, function]
(p) --> [void, function]

Q1.3
3 Points

Using static scoping determine the symbol table at line 17

(a) --> [int, global] --> [char, local to main]
(b) --> [char, global]
(q) --> [int, function]
(p) --> [void, function]
(main) --> [int, function]

Q1.4
3 Points

Using dynamic scoping determine the symbol table at line 17

(a) --> [int, global] --> [char, local to main]
(b) --> [char, global]
(q) --> [int, function]
(p) --> [void, function]
(main) --> [int, function]

Q1.5
3 Points

Using dynamic scoping determine the symbol table at line 13

(a) --> [int, global] --> [char, local to main]
(b) --> [char, global] --> [double, local to p]
(q) --> [int, function]
(p) --> [void, function]
(main) --> [int, function]

Q1.6
3 Points

Using dynamic scoping determine the symbol table at line 7

(a) --> [int, global] --> [char, local to main] --> [int, local to q]
(b) --> [char, global] --> [double, local to p]
(q) --> [int, function]
(p) --> [void, function]
(main) --> [int, function]

Q1.7
6 Points

Use static scoping determine the output (assume execution begins at main())

c

e

6.200000

Q

101

Q1.8
6 Points

Use dynamic scoping determine the output (assume execution begins at main())

c

W

6.200000

Q

87

Q2
8 Points

Define the concept of binding and explain the difference between static and dynamic binding.

Binding is when you connect an identifier with at least one attribute. This is alike assigning a variable that is an integer a value of '2'. The difference between static and dynamic binding is that static binding occurs before execution while dynamic binding occurs during execution. An example of static binding would be setting a variable to be an integer. An example of dynamic binding would be changing a variable 'x' from the value '0' to '1'.

Q3
8 Points

Many programming languages (including C and Java) prohibit the redeclaration of variable names in the same scope. Discuss the reasons for this rule. Could variables be overloaded the same way functions are in a language like Java?

The reason why many programming languages prohibit the redeclaration of variable names in the same scope is to prevent programmers from overloading variables, which would cause them to work in unexpected ways. If I wanted a variable to be 'x' at the beginning of my code and wanted to reassign the value, it would be redundant to re-declare the variable, as it already exists. Variables cannot be overloaded in the same way functions are in a language like Java since they don't take inputs in like functions do and they don't typically return values like functions do. The only reason functions do this is in order to be extra flexible, such as the primitive functions of addition, subtraction, division, multiplication, and modulo. It would be unnecessary to include that type of functionality in terms of variables since it would cause unnecessary shadowing of variables and their values and that type of functionality is not needed since you can simply just re-assign the value of a variable. The only reason I could see someone wanting this type of functionality is if they were trying to re-assign a pre-existing variable to a different data type, but even so, it would just be more logical to just create a new variable.

Q4
8 Points

Default arguments in function definitions, such as:

```
void print( int x, int base = 10);
```

present a problem for overload resolution. Describe the problem.

The reason why this would present a problem for overload resolution is because since the function/method defines int base = 10, it means that if there is also a function/method with the parameters of using int x, the programming language would have a difficult time figuring out the distinction between this function/method and just the one that has int x as its parameter.

Q5
8 Points

Are default arguments reasonable in the presence of overloading? Justify your answer.

Default arguments are reasonable in the presence of overloading because it gives the user of the function/method call more flexibility in defining default values of other values are provided. I can see this being used in practice such as defining default arguments for move functions/methods in a game since sometimes you might not want to define the distance of how far a character moves whereas sometimes you might want to define a bunch of parameters, such as distance or speed. Using overloaded functions in this case would provide more flexibility since I could set the default value to be move 5 steps and the user can still change that if they want to, but if they just call the function/method without defining how many steps, it won't immediately throw an error, but just use the default value.

Q6
8 Points

What is the difference between a statically typed and a dynamically typed language? What are some advantages and disadvantages for each? (At least four total examples)

The difference between a statically typed and dynamically typed language is the place where data types are checked. In a statically typed language, the data types are checked before compile time whereas in a dynamically typed language, the types are checked at runtime. Some advantages of static typing are the increased safety of your program since any program that doesn't immediately define their types will get called out immediately, the programmer can immediately see the data types without having to read for context within the program, the program will run a bit faster (even though it is not noticeable given modern hardware) since decisions are made before runtime, and function/method calls can be used for overloading more effectively without any outlying gimmicks. The disadvantages of a statically typed language are it can be a bit more tedious to set data values for every variable and function call for the programmer, some reasonable programs can be called out and throw errors even if they are safe just because some declaration is done incorrectly, it forces the programmer to foresee the usage of their variables and function definitions, which can slow down the whole process of programming, and an increase in troubleshooting time since programmers need more time to figure out if their compilation error was due to a data type not being defined or declared multiple times or if it was actually a compilation error where their code's functionality itself is throwing an error. Some advantages of a dynamically typed language are that you don't need to define data types when you declare them, it's easier to program without knowing how you will implement your code later on, more programs will work since everything gets checked at runtime, it is easier to find system loopholes while programming, and the programming languages are technically more simpler since you cannot implement function/method overloading as easily as a statically typed language. Some disadvantages of a dynamically typed language are that memory is allocated a bit more poorly since everything is organized and run and checked last minute, more code must be compiled at runtime, meaning that the program execution itself is a bit slower (even though it's very unnoticeable given modern hardware), errors are caught last minute and troubleshooting and debugging will typically take longer, and since there are no types, it can be harder to just read data types to understand the whole program's implementation and you may have to read the whole program to understand its functionality.

Q7
8 Points

The Boolean datatype may or may not be convertible to integral types, depending on the language. Compare the treatment of the Boolean type in at least two of the following languages with regard to this issue: Java, C, Python.

In Java, the Boolean datatype is not convertible to integral types since its treated as its own primitive data type alongside integers, doubles, and strings. In Python, the Boolean datatype is technically convertible to integral types since you can use it within an if statement without specifically checking if a value is True or False.

Q8
8 Points

What are advantages and disadvantages for Boolean's to be convertible to integers? Provide at least two.

Two advantages for Boolean's to be convertible to integers are that it is much easier to implement Boolean statements within mathematical equations within your code, such as counting how many right and wrong answers someone got based on the Boolean of tests as an example and it is easier to make that distinction that 0 is off or negative and 1 is on or positive. Two disadvantages for Boolean's to be convertible to integers are that it can be difficult to read in code since programmers may get it confused or be confused on its implementation, especially if they don't understand the assumed values of 0=off and 1=on, especially in Python where it does not need to be declared that "If pass_test" means that the code is seeking for a 'True' under that Boolean assignment and some programmers may accidentally believe it works exactly like an integer and use it, which will lead to some unexpected behavior on the program's functionality, which can be hard to troubleshoot, especially if the programmer does not understand the Boolean's usage in regard to using it with integers.

Q9
8 Points

Name and define the two types of type equivalence

The two types of type equivalence are structural equivalence and name equivalence. Structural equivalence refers to if the values within the variable are the same whereas name equivalence refers to if the name representing those values are the same or not.