Assignment 5

Jordan Dube

1. [20 points] Determine the symbol table and output for the following code. Please note the use of format specifiers and use an ASKII lookup table if necessary.

# include <stdio.h> // line 1

int a = 101;   
char b = 'z';

int q(char b) {

int a = 89;

printf ("%c\n",a);   
printf("%d\n",b); // line 7

return a;

}

void p() {

double b = 2.6;

printf("%c\n",a);

printf("%f\n",b); // line 13

a = q(a);

}

int main() {

char a = 'L';

printf("%c\n",b); // line 17

p();

return 0;

}

1. [6 pts] Using static scoping determine the symbol table at lines: 7, 13, and 17
   1. Line 7

|  |  |  |
| --- | --- | --- |
| NAME | BINDINGS | |
| a | Int = 89, local to q | Int = 101, global |
| b | Char = ‘z’, global |  |
| q | Int function |  |
|  |  |  |

* 1. Line 13

|  |  |  |
| --- | --- | --- |
| NAME | BINDINGS | |
| a | Int = 101, global |  |
| b | Double = 2.6,  local to p | Char = ‘z’, global |
| q | Int function |  |
| p | Void function |  |

* 1. Line 17

|  |  |  |
| --- | --- | --- |
| NAME | BINDINGS | |
| a | Char = ‘L’,  local to main | Int = 101, global |
| b | Char = ‘z’, global |  |
| q | Int function |  |
| p | Void function |  |
| main | Int function |  |

1. [6 pts] Using dynamic scoping determine the symbol table at lines: 17, 13, and 7
   1. Line 17

|  |  |  |
| --- | --- | --- |
| NAME | BINDINGS | |
| a | Char = ‘L’,  local to main | Int = 101, global |
| b | Char = ‘z’, global |  |
| q | Int function |  |
| p | Void function |  |
| main | Int function |  |

* 1. Line 13

|  |  |  |
| --- | --- | --- |
| NAME | BINDINGS | |
| a | Char = ‘L’,  local to main | Int = 101, global |
| b | Double = 2.6,  local to p | Char = ‘z’, global |
| q | Int function |  |
| p | Void function |  |
| main | Int function |  |

* 1. Line 7
     + Note: q does not return before line 8, so the value of q(a) is not known at line 7, hence the value of a in the scope of “local to main” is simply q(a) at this point.

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | BINDINGS | | |
| a | Int = 89, local to q | Char = q(a),  local to main | Int = 101, global |
| b | Char = a, local to q | Double = 2.6,  local to p | Char = ‘z’, global |
| q | Int function |  |  |
| p | Void function |  |  |
| main | Int function |  |  |

1. [4 pts] Using static scoping determine the output (assume execution begins at main())

z

e

2.6

Y

101

1. [4 pts] Using dynamic scoping determine the output (assume execution begins at main())

z

L

2.6

Y

76

1. [8 points] Define the concept of binding and explain the difference between static and dynamic binding.
   1. The concept of binding refers to the process of associating an attribute (properties that determine the meaning of the name to which they are associated) with a name. The process of binding can occur at various stages of the program running, such as load time or execution time. This is known as the binding time, which also includes the time it takes to compute the actual attribute.
   2. Static binding is the binding processes that execute before runtime (they bind static attributes to names), while dynamic binding happens during the execution of the program (they bind dynamic attributes to names).
2. [4 points] Name and define the three storage allocation methods defined in class

* Static objects are given an absolute address that is retained throughout   
  the program’s execution.
* Stack objects are allocated and deallocated in last-in, first-out order,   
  usually in conjunction with subroutine calls and returns.
* Heap objects may be allocated and deallocated at arbitrary times.

1. [4 points] Name and define the three types of memory management discussed in class.

* Manual management -> allocating memory and calling free/delete responsibility is solely on programmer. ERROR PRONE.
* Garbage collection -> an algorithm is continuously running on the available memory to check for any unused values (garbage). If found, frees the unused memory. Only problem is extra overhead memory & resources are needed to keep the collector running.
* Reference counting -> A counter keeps track of calls to an object. When the counter gets to 0, the object is no longer needed, and the memory location can be freed. Can be automated by counting references at compile time and inserting free operations as needed.

1. [8 points] Define the symbol table, environment, and memory in the context of binding.

* Symbol Table: A data abstraction to keep track of names and the attributes that are bound to them. It can be thought of as a function that expresses the binding of attributes to names. It is constantly updated by inserting and deleting names and their attributes as the program progresses, and the symbol table is discarded after the process completes.
* Environment: The binding of names to storage locations is called the environment.
* Memory: Binding of storage locations to values.

1. [5 points] Default arguments in function definitions, such as:

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

present a problem for overload resolution. Describe the problem.

* Having 2 parameters can result in the wrong print statement being called. Since base will always be 10 in the definition, it provides the same arguments that one can have in a 1 parameter overload. Because of this, the compiler cannot resolve the ambiguity in the case that the user uses a call such as print(1). Since the default value of the “base” parameter is defined, either the above print function OR the standard 1 parameter print function can be used, and the compiler cannot choose since they are both valid.

1. [5 points] Are default arguments reasonable in the presence of overloading? Justify your answer.

* No, it is not reasonable to use default arguments in the presence of overloading because if you know you are overloading, chances are you already have a function defined that has the default argument omitted. This will result in an error when the compiler tries to resolve the overloading. It is much safer to simply have a conditional branch in your method that switches functionality based on the value of a separate argument rather than use a default value in the declaration. Forbidding the use of default arguments in overloading makes the coding process easier on both the programmer AND the computer.

1. [8 points] Many programming languages (including C, C++, and Java) prohibit the redeclaration of variable names in the same scope. Discuss the reasons for this rule. Could not variables be overloaded the same way functions are in a language like C++ or Java?

* Variable names cannot be redeclared in the same scope because they are semantically different. Functions and operators cannot be redefined during runtime, but variables can change their value (and be cast to different data types). If we wanted to print the value stored in a, but a was defined as a char and an int, the compiler would have trouble determining which a to print. Since variables are bound and they STORE values, it is a lot harder to comfortably reference overloaded names without any side effects causing the program to output incorrect values. Functions and operators modify the values of functions, but they do not store any values (but they do often return values). They should not be overloaded because the symbol table is defined by scope, so having 2 variables with the same name would throw an exception when overload resolution tried to execute. If the compiler did not have an error, then the wrong values could be assigned to the unintended variable, and debugging would be a nightmare.

1. [8 points] What is a dangling reference? Can dangling references exist in C++? In Java? Why or why not?
   1. A dangling reference is a location that has been deallocated from the   
      environment but can still be accessed by a program.
   2. Yes, since the programmer can dynamically allocate memory and use pointers, then calling free or destroying a local variable can result in a pointer that points to a deallocated block of memory.
   3. No, Java does not allow dangling references because both pointers and memory management are abstracted from the programmer. When an object is no longer referenced, Java has a garbage collector which frees the object’s block of memory, thereby leaving no pointers pointing to that memory location.
2. [15 points] Name and define all the type constructors discussed during the “D10 – Data Types” lecture.

* Cartesian Product: AKA record or cross product, allows different data types to be stored together. (STRUCTS in C/C++). Can be stored by names or position, can be nested. Usually stored sequentially in memory, with the compiler keeping track of the offset used to access each field in a Cart. Prod. Can be called a tuple, which can have differing data types separated by commas. (CLASSES in Java, with optional interface implementations). Can be abstract and/or denotational.
* Union: Formed by taking the set theoretic union of the sets of their   
  values. 2 types: discriminated: type is specified for each value; undiscriminated: type is not specified, but assumptions must be made. Useful in reducing memory requirements because data items can be stored in overlapping memory.
* Arrays: Mapping from index to component. Sometimes called sequence types. Implementation is efficient, with elements sequentially ordered in memory and accessed using offset. Can be stored using pointers as well. Either in row-major, column-major, or row-pointer. Some languages allow declaration of array type without size, but most allow a size to be specified.
* Pointer Type: constructs the set of all addresses that refer to a specified type. They are implicit in languages that perform automatic memory management (Java). They can be used in the construction of recursive type.
* Recursive Type: A type that uses itself in its declaration. Represent data whose size and structure is not known in advance and may change as computation proceeds.

1. [4 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, Ada.

* Java regards Boolean as not ordinal or numeric. As such, it cannot convert a Boolean value to an integral value.
* C regards Bool as an integral type, so it can be converted to an integral value, where the int 0 is False and the int 1 is True. (Used \_Bool in Assignment 4 with these enumerations). The bool datatype is included with <stdbool.h> in the c99 library.

1. [4 points] What are advantages and disadvantages for Booleans to be convertible to integers?

* An advantage to using Booleans is readability. Someone looking over your shoulder can look at your output and if they see a 0, that could mean false OR it could be some value that takes some reading to interpret. Seeing TRUE makes it a lot easier to get confirmation. For the readability case, you can easily read true, and if you also wanted to do bitwise and/or/not operations, it would be easy without having to do any strcmp or similar string manipulation. Boolean also maps to a narrow range, while using ints can in some cases cause corruption (such as when function returns 2 when 2 is not a true/false value).
* A disadvantage is that it takes up more room in memory to store TRUE vs. simply storing 1 or 0. More abstractions can slow down some programmers. Programmers might also have to write more lines to actually do anything with the results, such as if the result of an operation is more than 1, then map it to TRUE, when one could simply return the result and use it later.

1. [7 points] Name and define the two types of type equivalence

* Structural equivalence: two data types are the same if they have the same structure. This means the same fields in the same order. This also can mean that the 2 equivalent objects are constructed from the same constructors. Not as rigid because the actual fields of objects might not be comprehensive in regards to information stored (i.e. a[10] vs. a[20] from D11 slides).
* Name Equivalence: two types are the same only if they have the same name. 2 objects are equivalent if they have the same type name, case specific. Variable declarations must have the same name. Allows for constructors to use different orders of fields yet have the same name.

1. [10 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)
2. Dynamic: type information is maintained and checked at runtime.
3. Static: types are determined from the text of the program and checked by the translator.
   1. Advantages for static typed:
      1. Allows for strong typing, where types are checked before runtime, and errors are thrown before execution.
      2. Reduces the amount of code to be checked by the compiler.
      3. Reduces the number of runtime errors (thereby saving CPU cycles) that cause execution to be interrupted.
      4. Helps remove ambiguity and resolve overloaded operators/functions.

Disadvantage:

* + 1. Can reject safe programs due to super strict type checking rules.
  1. Advantages for dynamically typed:
     1. Less resources spent at compile time, meaning a potentially shorter turnaround time.
     2. Smaller source code/easier readability due to less type specification/clutter.

Disadvantage:

1. Much weaker typed, which can cost time/sanity due to unforeseen type conflicts which impact program performance.