Nalongsone Danddank Student ID : 14958950 StarID: jf3893pd

Email: [nalongsone.danddank@my.metrostate.edu](mailto:nalongsone.danddank@my.metrostate.edu)\

**Metropolitan State University**

**ICS-365-01 —Organization of Programming Languages**

**Homework #6**

1. Compare the tombstone and lock-and-key methods of avoiding dangling pointers, from the points of view of safety and implementation cost.

Answer:

**Tombstones** take more memory. They are costly in both time and space, and . Because tombstones are never deallocated, their storage is never reclaimed. Every access to a heap- dynamic variable through a tombstone requires one more level of indirection, which requires an additional machine cycle on most computers. Apparently none of the designers of the more popular languages have found the additional safety to be worth this additional cost, because no widely used language uses tombstones.

While in the **lock-&-key pointer** requires add cpu time on each pointer assugnment to copy key as well as pointer. And pointer arithmetic could overwrite key in the heap. They Their values are represented as ordered pairs (key, address), where the key is an integer value. Heap-dynamic variables are represented as the storage for the variable plus a header cell that stores an integer lock value. When a heap-dynamic variable is allocated, a lock value is created and placed both in the lock cell of the heap-dynamic variable and in the key cell of the pointer that is specified in the call to new. Every access to the dereferenced pointer compares the key value of the pointer to the lock value in the heap-dynamic variable. If they match, the access is legal; otherwise the access is treated as a run-time error. Any copies of the pointer value to other pointers must copy the key value. Therefore, any number of pointers can reference a given heap- dynamic variable. When a heap-dynamic variable is deallocated with dispose, its lock value is cleared to an illegal lock value. Then, if a pointer other than the one specified in the dispose is dereferenced, its address value will still be intact, but its key value will no longer match the lock, so the access will not be allowed.

1. What is a data type?

Answer: The data types of a language are a large part of what determines that language’s style and usefulness. Along with control structures, they form the heart of a language. It defines a collection of data values and a set of predefined operations on those values. The main factor in determining the ease with which they can perform this task is how well the data types available in the language being used match the objects in the real world of the problem being addressed.

1. What is an abstract data type?

Answer: It is an enclosure that includes only the data representation of one specific data type and the subprograms that provide the operations for that type. An abstract data type are not defined in terms of concrete instances used in implementations. For instance, List, Stack, Queue, Set, Map, Steam.

1. How are floating-point types represented?

Answer: they are only approximations for many real values. For instance, “*Pi*” or “*e*” cannot be correctly represented in floating-point notation. Because on most computers, floating-point numbers are stored in binary which exacerbates the problem.

1. What languages support complex data type?

Answer: Fortran, Python, and Languages that support a complex type include operations for arithmetic on complex values.

1. Why is Java not strongly typed?

Answer: Although, java types can be explicitly cast, which could result in a type error but there are no implicit ways type errors can go undetected. For example, expressions are strongly java, but an arithmetic operator with one floating-point operand and one integer operand is legal. The value of the integer operand is coerved to floating-point, and a floating-point operation takes place.

1. What significant justification is there for the -> operator in C and C++?

Answer: the significant justification for the -> operation in C and C++ is writ-ability, and it is easier to write p -> q than (\*p).q

1. What are the arguments for and against Java’s implicit heap storage recovery, when compared with the explicit heap storage recovery required in C++? Consider real-time systems.

Answer: Implicit heap storage recovery eliminates the creation of dangling pointers through explicit deallocation operations, such as delete. The disadvantage of implicit heap storage recovery is the execution time cost of doing the recovery, often when it is not even necessary and there is no shortage of heap storage.

1. In what way is static type checking better than dynamic type checking?

Answer: There are two reasons that static type checking is better than dynamic type checking:

1. anything done at compile time leads to better overall efficiency, simply because production programs are often executed but far less often compiled.

2. type checking uncovers program errors, and the earlier errors are found the less costly it is to remove them.

1. Explain how coercion rules can weaken the beneficial effect of strong typing.

Answer: they are allowing many potential type errors to be masked by simply coercing the type of an operand from its incorrect type given in the statement to an acceptable type, rather than reporting it as an error.