John Pennig

Homework 8

1. Chapter 6 Review Questions:

9. Static Array: A static array is one in in which subscript ranges are statically bound and storage allocation is done before run time. The advantage is that static arrays are the most efficient.

Fixed Stack-Dynamic Array: A fixed stack-dynamic array are ones in which the subscript ranges are statically bound, but allocation of storage is done at declaration elaboration time. The advantage is these arrays are space efficient.

Fixed Heap-Dynamic Array: A fixed heap-dynamic array is similar to a fixed stack-dynamic array that subscript ranges and storage binding are both fixed after storage is allocated. However, fixed heap-dynamic arrays have both bindings done when they are requested by the program and the storage is allocated from the heap. The advantage is flexibility, the array size fits the problem.

Heap-Dynamic Array: A heap-dynamic array is one where subscript ranges, and storage is dynamic and can change during the arrays lifetime. The advantage is flexibility, the array can grow and shrink during execution.

28. List comprehensions come from an idea derived from set notation.

29. Union: A union is a type whose variables may store different type values at different times during the execution.

Free Union: Free unions are unions which do not have type checking and do not enforce types allowing for programmers to have complete freedom in their use.

Discriminated Union: A discriminated union is one that has a type indicator and type checking, the indicator is called a discriminant.

32. The two common problems with pointers are dangling pointers, and lost heap-dynamic variables called garbage. Dangling pointers is when a pointers variable has been deallocated. An example would be if multiple pointers were pointing to the same address, one of the pointers influence the address in a way not expected by the other pointers. Now the other pointers are dangling. Lost heap-dynamic variables are when variables are no longer accessible to the variable in a program and no longer need to be allocated in a program. Lost heap-dynamic variables are when a pointer is set to a newly created heap-dynamic variable and then that pointer is set to another variable. That first heap-dynamic variable is no longer accessible.

35. Reference types are better for formal definitions in C++ because a reference type variable is a constant variable that is always dereferenced and after initialization the reference type can never be set to reference any other variable.

37. The eager method for is often called the reference counter method. The method has a counter that says how many more times the memory is referenced. Every time it is referenced, it decrements the counter until 0 where it is then reclaimed immediately. The lazy method is called the mark-sweep process. It allocates storage when needed and deallocates when only necessary as well.

2. Explain with example, the tombstone and the lock-and-key approach to the dangling pointer problem + Chapter 6 Problem Set Question 4.

The tombstone approach to the dangling pointer problem involves attaching a special cell called a tombstone that itself is a pointer to the heap-dynamic variable. If the variable is deallocated, the tombstone remains and is set to nil so that the program is able to recognize is the variable has been reallocated.

The locks-and-keys approach to the dangling pointer problem is where the pointers themselves are ordered pairs consisting of a key and address. When a new heap-dynamic variable is allocated, a lock value is placed in both the key address of the pointer and the lock cell of the variable. Every reference compares the key values to make sure they match.

Question 4 Problem Set: Tombstones are very safe as they prevent a pointer from ever pointing to a dereferenced variable. However, the implementation cost is very high in time and space. Tombstones are never deallocated, so their space is never reclaimed, and every pointer through a tombstone causes another level of indirection. Lock-and-keys safety wise cause run-time errors if an access is to a deallocated variable. The implantation cost is adding in an additional key to both the pointer and heap-dynamic variables. However, when an item is dereferenced, the key is changed to a different value so that the old pointer throws an error still, but the space can be used for another variable.

3. Write short essays on each of the following topics. The essays should be broad, but also address the specific questions:

a. Programming Language issues around basic types. What are the issues to be considered when a language provides support for primitive data types? How do these apply to all the kinds of primitive types discussed in the text? (~400 words)

There are many issues that must be addressed around basic types in a programming language in order for that programming language to be useful in modern computing. These basic types include primitive data types such as characters, numerics, and Booleans that are the building blocks for more complex types critical to a powerful programming language. To deal with these types, we must also solve the problems including memory management, type checking, and performance with all types.

One of the primary issues in basic types is the memory management of these types. Different types may require varying amounts of memory for a primitive type. An example of this is Booleans which can only be either true or false may only need 1 bit where a float may require 4 bytes to properly store a value. The efficient management of memory is especially critical for the primitive data types that can severely bog down a system if you gave 16 bytes for every type. It is also important to be consistent in the storage allocation for types to be efficient such as a regular float will typically be assigned less storage than a double will. This leads us to our next problem with precision and overflow. If there is not enough storage allocated to float types, this can cause precision issues that can cause critical problems. Most floating-point numbers are approximations, but it is still important that they are as close to accurate as possible. If they are not allocated enough storage, it can cause incorrect rounding and overflow leading to a potentially very incorrect result. Overflow occurs if a type does not have enough storage to perform arithmetic on it and causes the memory to overflow and lose values. These issues must be clearly taken care of for the primitive data types to ensure a useful programming language.

Another major issue for basic types includes type checking, safety, and conversion. Type checking is a major issue for error detecting to make sure the program Is operating as the programmer intended. Type checking and ensuring that the type is able to perform the correct arithmetic such as addition with two integers is important in all computing. Another major issue is conversion such as the capability to perform addition with an integer and float, but again type checking must find errors such as not allowing addition between an integer and a character type to ensure type safety. Type checking, safety, and conversion is critical to all programming languages.

Programming languages face many challenges in primitive data types, including memory management, precision and overflow, type checking, conversion, and safety. These are important issues that language developers must take into account to make an efficient and useful modern language.

b. Programming Language issues around Strings. What are the various ways in which languages provide the string type? Compare and contrast the case of C, C++ and Java. Discuss the design issues and all the issues that arise when string is provided as a type, including the problem of security. (~500 words)

A string type is one that includes a sequence of characters and are one of the most used data types in programming. Strings have two very important design issues that such as should strings be a special character array or a primitive type and should strings be static or dynamic length. Different languages may implement strings in a vastly differing way such as between C, C++, and Java. When comparing these 3 languages, we can see the differences in how the string types are represented, the ability to manipulate, and management of these strings.

Strings are not represented as a primitive type in both C, and C++. Instead in these languages, string data is usually stored in arrays of single characters and are referenced as arrays in the language. These languages do support operations through their standard libraries that are critical to perform operations such as assignment, catenation, substring reference, comparison, and pattern matching. These languages terminate their character strings with a special character often 0 to represent the end of the string. Functions on strings in C, and C++ continue their operations until this special character appears and then they complete. These string manipulation functions are inherently unsafe and have led to many programming errors such as not guarding against overflow. The strings in C are considered limited dynamic length strings because they are allowed to have a varying length up to a fixed maximum.

It is encouraged however in C++ to use the string class from the standard library rather than char arrays like C. The standard string class from the standard library offers a higher-level abstraction from char arrays like in C. The string class allows for dynamic memory management and handles allocation, deallocation, and resizing automatically.

In Java, strings are supported by using the string class for immutable strings and the stringbuffer class whose values are changeable and similar to arrays of single characters. Java strings are static meaning that they have a fixed length when they are created, while stringbuffers are dynamic much like the string class from C++.

The issues that arise when designing strings for a programming language such as security and implementation are important to consider. The security of strings is a critical issue when you may be taking in user input or performing operations on the strings. In C, strings are efficient in memory allocation and access but can cause critical errors such as overflows and memory leaks. These errors happen because the strings are considered limited dynamic and may not have enough storage after concatenation if the programmer is not careful. In C++, this issue is mainly fixed due to the dynamic length strings, but other issues can arise if programmers perform string manipulation functions without validating the strings to insure correct input. In Java, since strings are immutable, they cannot be manipulated but are still open to other issues such as when handling sensitive information since strings are stored in plain text. There is always a balance that language designers must consider when implementing strings in languages so they must take into account security, storage, and manipulation to ensure a useful data type.

c. Programming Language issues around arrays and associative arrays. Discuss the various design issues for arrays, choices for each issue, and the implications of making particular choices on these issues. Compare and contrast C++ and Java. What are associative arrays, and how do the common languages provide support for these? (~600 words)

An array can be defined as a homogenous aggregate of data elements where each individual element can be identified by its position in the aggregate relative to the first element. There are many issues that a language designer must consider when planning to implement arrays into a language such as design issues, and implications that certain choices make for the whole of the language.

There are many design issues that must be solved to implement arrays effectively in a language. These issues are what types are legal for subscripts, are subscripting expressions in element references range checked, are subscript ranges bound, when is array memory allocated, are ragged or rectangular multidimensional arrays allowed or both, can arrays be initialized when they have their storage allocated, and what kind of slices are allowed? Different languages handle these issues in different ways leading to varying uses of arrays in languages.

To reference a particular element in an array, most languages use a subscript or index. The subscript is used to specify which element in the array the programmer is referencing starting at 0. Subscripts can either be static if the references are all constants, otherwise they can be dynamic. Most languages use the syntax of the array name with the specific subscript in brackets directly after the name to reference a specific element. In C++ and Java, arrays start at index 0. C++ does not implicitly check that indexes are within range which can cause errors, but Java does.

The answer to when array memory is allocated can be complex. It can vary from language to language and depend on the type of array. There are 4 main arrays, static, fixed stack-dynamic, fixed heap-dynamic, and heap-dynamic. Static arrays have subscript ranges statically bound and memory allocation static and before run time. Fixed stack-dynamic arrays have static subscript ranges, but the storage allocation is typically done during runtime. Fixed heap-dynamic arrays are similar to fixed stack-dynamic arrays in which subscript and storage allocation is static but only once the program requests them during execution. Heap-dynamic arrays subscript range and storage is completely dynamic and can change several times during execution. In C++, arrays declared that include the static modifier are static and if they do not include the static modifier are then fixed stack-dynamic arrays. C++ also supports fixed heap-dynamic arrays by using operators new and delete to manage heap storage and arrays can be treated as pointers to the heap. In Java, all non-generic arrays are fixed heap-dynamic arrays. Many languages initialize arrays at the same time their memory is allocated. C++ and Java both allow initialization of their arrays.

Each language may also have a different implementation or rules for allowing rectangular, or jagged arrays. Rectangular arrays are multidimensional arrays where the number of rows and columns have the same number of elements. Jagged arrays are arrays where the number of rows and columns do not have the same number of elements. These multidimensional arrays have arrays in their arrays. Both C++, and Java support jagged arrays but not rectangular arrays. Both languages use separate brackets to specify each dimension.

Slices of an array can be referred to as a substructure of an array. Slices can be seen as a specific part of an already existing array. If arrays cannot be manipulated in a language, then that language has no need for slices. Languages such as C++ and Java do not have any inherent support for slices.

An associative array can be defined as an unordered collection of data elements that are indexed by an equal number of values called keys. These keys are user-defined and must be stored in the structure of associative arrays. Each element in these arrays are stored as a key and a value. These associative arrays are also commonly referred to as maps, dictionaries, or hashes. C++, and Java provide support for these associative arrays through their standard libraries.