Andrew Morris
4-11-16
CSCI 330
Assignment 6

Chapter 6:

1. **Review Questions:**

    1. What is a descriptor?
        a. A descriptor is the collection of the attributes of a variable

    2. What are the advantages and disadvantages of decimal data type?
        a. Easily translatable to decimal.
        b. Wastes memory and has a limited range

    3. What are the design issues for character string type?
        a. Should the type be an array or a primitive type
            i. C++ cstring vs stringstream
        b. Should the string be dynamic in length

    4. Define ordinal, enumeration, and subrange types.
        a. Ordinal
            i. The values can be put in a one-to-one correspondence with the positive integers
        b. Enumeration
            i. Named constants, all possible values are provided in the definition

        c. Subrange
            i. A sequence in an ordinal data type.
    7. In what ways are the user-defined enumeration types of C# more reliable than those of C++?
        a. C++ represents enumerated data types as integers. So Enumerated types can be used to operate with types that are not in the enumeration. Which can be undesirable.
    8. What are the design issues of arrays?
        a. Should the first elt be a 0 or 1.
        b. When does allocation take place
        c. Can multiple types populate a single array.
        d. Initialization of the array.
    9. Define static, fixed-stack dynamic, stack-dynamic, fixed heap-dynamic, and heap-dynamic arrays. What are the advantages of each?

   a. Static
      i. Allocation happens before run time, the size of the array cannot change.
   b. Fixed stack dynamic
      i. An array whose subscripts are static. The allocation is done when declared by the program.
   c. Stack dynamic
      i. The range can passed into the program, but the storage is allocated when the array is declared.
   d. Fixed heap dynamic
      i. The space is allocated on the heap for a known number of elts.
   e. Heap dynamic
      i. The space is allocated on the heap is not bounded, and can allocated at run time.

17. Define row major order and column major order
    a. [row][column] – each column is found by adding to the location of the zeroth row element
    b. [column][row] – each row element is found by adding to the zeroth elt in the column.
18. What is an access function for an array?
    a. A mathematical function that can give assigns the address of any array element to an index, based on pointer arithmetic.

2. Problem Set:

1. What are the arguments for and against representing Boolean values as single bits in memory?
   a. For: Incredibly space efficient
   b. Against: Access is special, therefore more expensive.
2. How does a decimal value waste memory space?
   a. Each single decimal value requires 4 bits. The maximum 4 digits can represent is 16 values (0 – 15). Therefore the decimal wastes about 1/3 of the available space.
5. What disadvantages are there in implicit dereferencing of pointers, but only in certain context? For example, consider the implicit dereference of a pointer to a record in Ada when it is used to reference a record field.
   a. A reference will be guaranteed to have some object associated when we pass through a reference. When we pass a pointer, if there is no object and the pointer is implicitly dereference, the pointer will be null.
7. What significant justification is there for the -> operator in C and C++?
   a. It is not significant justification, but “ptr->field” it is much more readable than “(*ptr).field”

(9) The union in C and C++ are separate from the records of those languages, rather than combined as they are in Ada. What are the advantages and disadvantages to these two choices?

- f. Separated
    - i. Good
        1. Do not need to check for types, easier to code
    - ii. Bad
        1. There is no type checking, so there can be some confusion about the contents of the union.
- g. Combined
    - i. Good
        1. Can support type checking
    - ii. Bad
        1. Have to determine type compatibility

10. Multidimensional arrays can be stored in row major order, as in C++, or in column major order, as in FORTRAN. Develop the access functions for both of these arrangements for three-dimensional arrays.
    - a. Row major
        - i. a[i,j,k] = &a[0,0,0] + (i * height + j * width + k) * sizeof(type)
    - b. Column major
        - i. a[i,j,k] = &a[0,0,0] + (k * width + j * height + i) * sizeof(type)
11. In the Burroughs Extended ALGOL language, matrices are stored as a single-dimensioned array of pointers to the rows of the matrix, which are treated as a single-dimensioned arrays of values. What are the advantages and disadvantages of such scheme?
    - a. Good
        - i. Allows easy access to row elements once the zeroth elt of a row is known
        - ii. Arrays larger than the computer's memory can be accessed.
    - b. Bad
        - i. Cannot access elements of a single column nearly as quickly as the row elements.

Part 2: Programming

Design a test program to determine type compatibility rules of C++ compiler. Write a report on your findings
Source (findings are in comments for each test):

```
#include <iostream>
#include <cstring>

using namespace std;
```

```
struct example {
        int x = 0;
        int y = 0;
};

int main() {
// int to float -> works
        int x = 1;
        float y = x;
// float to int -> works a = 2
        float z = 2.1;
        int a = z;
// int to char -> works
        int b = 100;
        char c = b;
// char to int -> works
        char d = 'a';
        int e = d;
// int to string ->ERROR "No viable conversion error"
        int f = 10;
        string g = f;
// string to int -> ERROR "No viable conversion error"
        string h = "S";
        int i = h;
// enumerated to int -> ERROR: expression not assignable
        enum days {mon, tue, wed, thur, fri, sat, sun};
        int j = 5;
        mon = j;
// int to enumerated -> Works
        int k = thur;
// struct to another struct of diff type
                // -> expected unqualified id;
        struct st1 {
                int x = 0;
                int y = 1;
        };
        struct st2 {
                int x = 2;
                int y = 4;
        };
        st1 = st2;
// float ptr to int number -> cannot initialize float* with an rvalue of type int *
        int m = 5;
        float * flptr = &m;
// int ptr to float number -> Cannot initialize a variable of type int * with and rvalue of type float
*.
        float n = 5.54;
        int * o = &n;

}
// float ptr to int number -> ERROR assigning to 'float *' from incompatible
    type 'int *'
        int m = 5;
        float * flptr;
        flptr = &m;
// int ptr to float number -> assigning to 'int *' from incompatible type
    'float *'
        float n = 5.54;
        int * o;
        o = &n;
        cout << "flptr =" << *(flptr) << endl;
        cout << "o =" << *(o) << endl;

}
```

```
Output:
dblindptr.cpp:26:9: error: no viable conversion from 'int' to 'string' (aka 'basic_string<char,
    char_traits<char>, allocator<char> >')
     string g = f;
         ^   ~
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/../
include/c++/v1/string:1333:5: note:
    candidate constructor not viable: no known conversion from 'int' to 'const
std::__1::basic_string<char> &'
    for 1st argument
  basic_string(const basic_string& __str);
  ^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/../
include/c++/v1/string:1348:31: note:
    candidate constructor not viable: no known conversion from 'int' to 'const value_type *'
    (aka 'const char *') for 1st argument
  _LIBCPP_INLINE_VISIBILITY basic_string(const value_type* __s);
                            ^
dblindptr.cpp:29:6: error: no viable conversion from 'string' (aka 'basic_string<char,
char_traits<char>,
    allocator<char> >') to 'int'
     int i = h;
       ^   ~
```

```
dblindptr.cpp:33:6: error: expression is not assignable
        mon = j;
        ~~~ ^
dblindptr.cpp:45:6: error: expected unqualified-id
        st1 = st2;
            ^
dblindptr.cpp:48:10: error: cannot initialize a variable of type 'float *' with an rvalue of type 'int
*'
        float * flptr = &m;
                ^       ~~
dblindptr.cpp:51:8: error: cannot initialize a variable of type 'int *' with an rvalue of type 'float
*'
        int * o = &n;
              ^   ~~
ptrtypetest.cpp:11:8: error: assigning to 'float *' from incompatible
      type 'int *'
        flptr = &m;
              ^ ~~
ptrtypetest.cpp:15:4: error: assigning to 'int *' from incompatible type
      'float *'
        o = &n;
          ^ ~~
```