

Examination 2 Programming Languages and Systems Concepts,
CSI 511
Fall 2001 (December 17, 2001)

1 Rules of the Exam

This examination is open book and notes. Calculators are permitted. Networked devices are strictly prohibited. The questions are marked as to their relative value, the exam will be scored out of 100% but is worth 25 points toward your course grade. Relax and try to do what you can.

2 The Problem Set

1. Readings (10 %) Note: Word for word copying from the readings will not get credit, answer IN YOUR OWN WORDS:

- (a) What is step-wise refinement and what problem does Niklaus Wirth apply it to?(5 %)

Step-wise refinement is a method of successive approximation widely used in engineering and science. The approach uses a simple preliminary design and iterative extends it until it is fully featured. Wirth applied this approach to the 8 queens problem

- (b) Dijkstra claims go-to statements are bad for higher level languages, because they make programs hard to understand and are not more expressive than go-to free constructs. Jacopini had an automated translation system to take programs containing go-to's and create go-to free programs. Does this solve the problem, why or why not?(5 %)?

Jacopini's approach does not solve the underlying readability problem. Jacopini's approach translates programs containing go-to statements into programs using so called "structured programming" constructs. This approach does not guarantee that the "structured" output is any more readable than the input, in fact it can be *harder* to read than the input.

2. Array Management (25 %):

- (a) Consider a three dimensional array declared as:

```
A : array [1..3,1..4,1..5] of integer;
```

using a Pascal like syntax. Assuming that integers are 32 bit values, and that the array starts at memory location 1000 (base 10) what is the offset of $A[1, 2, 3]$ if A is stored in row major order? What is the offset if column major order is used (5 %), show your work?

We can describe the problem's structure to look like:

$$\text{Address}(A[1, 2, 3]) = \text{Address}(A) + \text{Offset}([1, 2, 3])$$

We know $\text{Address}(A) = 1000$, the offset is what we need to compute.

Let's say that the i th dimension of the array has a lower bound L_i and an upper bound U_i , so the array is dimensioned as follows:

$$A[L_1..U_1, L_2..U_2, L_3..U_3]$$

We define the number of elements across each dimension as:

$$N_i = U_i - L_i + 1$$

Then we know the address of $A[d_1, d_2, d_3]$ is (for row major order):

$$\begin{aligned} \text{Address}(A[d_1, d_2, d_3]) &= \text{Address}(A[L_1, L_2, L_3]) + \\ &\quad \left[((d_1 - L_1) \times N_2 + (d_2 - L_2)) \times N_3 + (d_3 - L_3) \right] \times \text{sizeof}(\text{integer}) \end{aligned}$$

While for column major order we get:

$$\begin{aligned} \text{Address}(A[d_1, d_2, d_3]) &= \text{Address}(A[L_1, L_2, L_3]) + \\ &\quad \left[((d_3 - L_3) \times N_2 + (d_2 - L_2)) \times N_1 + (d_1 - L_1) \right] \times \text{sizeof}(\text{integer}) \end{aligned}$$

- Row Major Case: The adjacent addresses differ by one in the last index in this case. Thus, memory can be viewed as a block composed of 3 layers dimensioned 4×5 integers, with the layers having strips of 5 integers. The solution for this case is:

$$\begin{aligned} \text{Address}(A[1, 2, 3]) &= \text{Address}(A) + \text{Offset}([1, 2, 3]) \\ &= 1000 + \left[((1 - 1) \times 4 + (2 - 1)) \times 5 + (3 - 1) \right] \times \text{sizeof}(\text{integer}) \\ &= 1000 + [0 + 5 + 2] \times 4 = 1028 \end{aligned}$$

- Column Major Case: The adjacent addresses differ by one in the first index in this case. Thus, memory can be viewed as a block composed of 5 layers dimensioned 4×3 integers, with the layers having strips of 3 integers. The solution for this case is:

$$\begin{aligned} \text{Address}(A[1, 2, 3]) &= \text{Address}(A) + \text{Offset}([1, 2, 3]) \\ &= 1000 + \left[((3 - 1) \times 4 + (2 - 1)) \times 3 + (1 - 1) \right] \times \text{sizeof}(\text{integer}) \\ &= 1000 + [24 + 3 + 0] \times 4 = 1108 \end{aligned}$$

- (b) The Perl programming language has associative arrays, in C++ associative arrays are not part of the language, but they are supported as part of the STL library. STL supports associative arrays via the “map” container class. STL maps are implemented using binary search trees (red-black trees to be exact, a balanced variety of binary search tree). Consider an STL map containing N elements.

- Give the computational complexity of finding an element in a C++ map. Justify your answer (5 %).

The cost of looking up an item in a balanced binary search tree is proportional to height of the tree, so for a tree containing n items, the cost is $O(\log n)$ (both for comparisons and pointer evaluations) where the log is base 2.

ii. Give the cost of traversing a C++ map. Justify your answer (5 %).

The cost of traversing a binary search tree is proportional to the number of elements in the tree, so it is $O(n)$.

- (c) Fortran 90 supports array slicing, some high performance computing languages support slices in a more general sense. Suppose that for a matrix, we wanted to make a *diagonal* array slice across an $N \times N$ array of 64 bit (double precision) floating point values. If the array begins at location 2000 and the default Fortran ordering is used, give the calculation the compiler would use to access the i th element, where $1 \leq i \leq N$ (10 %).

Typically, a dummy array is created for array slicing, and the number of elements in the dummy array is N , and by definition of the diagonal of a matrix, the dummy array has the values:

$$\text{dummy}[i] = A[i, i] \quad 0 \leq i < N$$

Fortran happens to be column major, and the given array is square. The compiler can either precompute and store the pointers in a dummy array or evaluate the address calculation at run time:

$$\begin{aligned} \text{address}(\text{dummy}[i]) &= \text{address}(A[i, i]) \\ &= 2000 + (((i - 1) \times N) + (i - 1)) \times \text{sizeof}(\text{double}) \\ &= 2000 + 8(i \times N - N + i - 1)\text{Bytes} \end{aligned}$$

3. Object oriented computing (20 %): Consider the following C++ code using templates, which exploits polymorphism and operator overloading. The code is supposed to transpose a dynamically allocated two dimensional array, and use the transpose of the array. However, when running the program, the performance is extremely poor, and it gives the wrong results. Find and fix the bugs in this program.

```
template <class T> class Array2D {
private:
    unsigned int LENGTH; // Number of elements in array
    unsigned int nrows;
    unsigned int ncols;
    unsigned int rowsize;
    T *array;
public:
    Array2D( const int row, const int col ){
        this->LENGTH = row * col;
        this->nrows = row;
        this->ncols = col;
        this->array = new T[this->LENGTH];
    }
    Array2D( const Array2D<T> &x ){
        if (this != &x){
            if (this->T != NULL){
                delete [] this->array;
            }
            this->array = new T[x.LENGTH];
            this->nrows = x.nrows;
            this->cols = x.ncols;
            for (int i = 0; i < this->Length; ++i){
                this->array[i] = x.array[i];
            }
        }
    }
};
```

```

    }
    unsigned int len() { return LENGTH; }
    unsigned int rows() { return nrows; }
    unsigned int cols() { return ncols; }
    // 2D Indexing Function into 1D Data Storage Structure
    inline T& operator() ( const register int y, const register int x ) const {
        int pos = // The students fill this in
        return array[pos];
    }
}

void transpose_swap(Array2D<int> data, int i , int j){
    int tmp = data(j,i);
    data(j,i) = data(i,j);
    data(i,j) = tmp;
}

void transpose( Array2D<int> data ){
    for (int i = 0; i < data.rows(); ++i){
        for (int j = 0; j < i; ++j){
            transpose_swap(data, i, j);
        }
    }
}

extern void use_transpose(Array2d<int> A);

int main(){
    Array2D<int> A(1000,1000);

    for (int i = 0; i < data.rows(); ++i){
        for (int j = 0; j < data.cols(); ++j){
            cin >> A(i,j); // read in A(i,j) from the input
        }
    }
    transpose(A);
    use_transpose(A);
}

```

The problem is that call by value is being used for the parameter transmission to both the `transpose` and `transpose_swap` routines. This implicitly invokes the copy constructor, which laboriously initializes the newly allocated array (formal parameter) to match the actual parameter. To fix it, I'd do the following:

- Change the the lines of code specifying the formal parameters to the offending functions to have call by reference, so the lines that read (intervening lines omitted for clarity):

```

void transpose_swap(Array2D<int> data, int i , int j){
void transpose( Array2D<int> data ){

```

now pass the formal parameter `data` by reference and instead read:

```

void transpose_swap(Array2D<int> &data, int i , int j){
void transpose( Array2D<int> &data ){

```

This change is mandatory.

- As an optional change, the copy constructor should be made a private member to prevent its accidental invocation.

- One student noted that the private members of object `x` are accessed in the copy constructor. If we make the copy constructor private and change its body to output an error message in the case of accidental invocation, then this will be a non issue.

4. Parameter transmission and Scope rules (30 %):

(a) Consider the following program:

```

int a = 1;
int b = 2;
int c = 3;

program main()
  procedure p(int x, int y)
    procedure q(int x, int y)
      begin
        x = c + 1;
        print(a, b, c, x, y);
      end
    begin
      int c = 4;
      x = a + 5;
      y = b + 6;
      q(a, b);
      print(a, b, c, x, y);
    end
  end
begin
  int x = 8;
  int y = 9;
  p(x,y);
  printf(a, b, c, x, y);
end

```

i. Assume Static Scope: Give the output of this program for: Call by Value, Call by Reference, Call by Value-Result and Call by Name (10 %).

Some notes: due to nested scope rules, the `c` from `p`'s activation record is preferred to that of the `c` from `main` during the calls to `p` and `q`. This influences the value of `x` when `q` is invoked.

Calling Convention	Print in q	Print in p	print in main
Call by Value	1 2 4 5 2	1 2 4 6 8	1 2 3 8 9
Call by Reference	5 2 4 5 2	5 2 4 6 8	5 2 3 6 8
Call by Value-Result	1 2 4 5 2	5 2 4 6 8	5 2 3 6 8
Call by Name	5 2 4 5 2	5 2 4 6 8	5 2 3 6 8

ii. Assume Dynamic Scope: Give the output of this program for: Call by Value, Call by Reference, Call by Value-Result and Call by Name (10 %).

It happens that dynamic scope results in the same variables resolving as in the static scope case (due to the fact that the containing function invokes the contained function). However, please note that although static and dynamic scope have the same results *for this class of problem* this similarity does not hold in the general case (frequently the results may differ).

Calling Convention	Print in q	Print in p	print in main
Call by Value	1 2 4 5 2	1 2 4 6 8	1 2 3 8 9
Call by Reference	5 2 4 5 2	5 2 4 6 8	5 2 3 6 8
Call by Value-Result	1 2 4 5 2	5 2 4 6 8	5 2 3 6 8
Call by Name	5 2 4 5 2	5 2 4 6 8	5 2 3 6 8

(b) Consider the following program (assume C style scoping):

```

int x = 3;
int y = 4;
int m = 5;

int
mult_mod(int a, int b, int c){
    return (a * b) mod c;
}

int
main(){
    int product = multmod(x + 1, y + 2, m);
    print(product, x, y, m);
}

```

What is the output of this program using call by value, call by value result, call by reference and call by name (10 %)?

Here we stress call by name's *lazy* evaluation of its parameters, which causes `mult_mod` to return $(3 + 1 * 4 + 2) \bmod 5 = (3 + 4 + 2) \bmod 5 = 9 \bmod 5 = 4$, while call by value generates $((3 + 1) * (4 + 2) = (4 * 6) \bmod 5 = 4$ (which are coincidentally equal). Note that references cannot be generated to the results of expressions in C and C++ (we tried this in class, and found that for the pre increment and post increment operators, the reference was the address of the variable, but for expressions such as this, the expressions did not have addressable memory locations). Call by value-result should also fail for similar reasons (the result cannot be returned).

Calling Convention	output
Call by Value	4 3 4 5
Call by Name	4 3 4 5

5. Garbage Collection (20 %):

- (a) Consider a doubly linked list, which is circularly linked. The program managing this list has a single pointer to the head of the list. You may assume that there are no other pointer to any nodes in the list. Suppose that the user sets the pointer to NULL, to reset the list to be empty.
- i. Would mark and sweep approaches be able to reclaim this space, why or why not (5 %)?
Yes, because mark and sweep will recursively visit any unmarked descendants of a pointer or will skip an already visited pointer structure.
 - ii. Would reference counting approaches be able to reclaim this space, why or why not (5 %)?
No, because each nodes reference count would be greater than zero.
- (b) Your company hires a new programmer, Wile E. Coyote (Super Genius) who suggests that the next and previous pointers can be consolidated by using a trick he found on the web which works by using exclusive or to encode the pointers. The trick involves keeping the address of the head and the tail of the list directly recorded in pointers. The pointer manipulations seem to allow for correct traversal of the list, but he wants to know whether a mark and sweep or reference counting approach can handle the garbage collection properly. What do you think the answer is, and why (10 %)?

Both mark and sweep and reference counting (in fact all garbage collection tools) need to be able to decode pointers and identify their associated regions in memory. This is **not** possible when such unusual pointer handling is used. In particular the encoding would make mark and sweep's traversal of the list difficult (in fact I don't think a general purpose mark and sweep could handle this) so that legitimately allocated nodes would be incorrectly freed, leaving dangling references. The reference counting techniques would find it hard to get right, since pointer assignment is not done in a straight forward fashion. Normal pointer arithmetic (addition or subtraction of offsets typically) is tolerated by many garbage collectors. It might be possible to hand code a special purpose garbage collector for this scheme. However, Wile E. Coyote would be hard pressed to make it work, and even if he did get it working it might take very long to implement and debug, that this could be a real CLM (career limiting move).