Final Practice
CS1622
April 19, 2005

1. Types fall into two categories: *basic types*, or types built with *type constructors*. We use type expressions to express them, for example, `int` might be used to denote the basic integer type, and in a language with arrays, `array(-1..8) of int` might be used to denote an array of 10 integers with indices ranging from -1 to 8.

(a) Write type expression for an array of pointers to reals, where the index ranges from 1 to 100;
(b) a two-dimensional array of integers, whose rows are indexed from 0 to 9 and whose columns are indexed from -10 to 10;
(c) functions whose domains are functions from integers to pointers to integers and whose ranges are records consisting of an integer and a character.
(d) Write type expressions for the types of `foo` and `bar` in the following C declarations

```c
typedef struct {
    int a, b;
} CELL, *PCELL;

CELL foo[100];
PCELL bar(int x; CELL y) {...};
```

2. Translate the expression -(a+b) * (c+d) + (a+b+c) into three-address code.

3. Given the following C program:

```c
main() {
    int i;
    int a[10];
    i = 1;
    while (i <= 10) {
        a[i] = 0; i = i+1;
    }
}
```

(a) translate the executable statements to three-address code;
(b) convert the three-address code representation into a control flow graph.
4. Describe what happens at a procedure call, e.g., how is control transferred to the callee, how are activation records set up etc.

5. You are given the following C program:

```c
#include<stdio.h>

int a[10] = {9, 2, 17, 23, -9, 110, 0, 9, 2342, 3};

int partition(int p, int r) {
    int i, j, x, v;
    x = a[p];
    i = p-1;
    j = r+1;
    while (1) {
        do {
            j--;
        } while (a[j] > x);
        do {
            i++;
        } while (a[i] < x);
        if (i < j) {
            v = a[i];
            a[i] = a[j];
            a[j] = v;
        } else
            return j;
    }
}

void quicksort(int m, int n) {
    int i;
    if (n > m) {
        i = partition(m, n); /* partition array portion from [m.. n] */
        quicksort(m,i);
        quicksort(i+1,n);
    }
}

int main() {
    int i;
    quicksort(0,9);
    for (i=0; i<10; i++)
        printf("%d ", a[i]);
}
```

(a) The given array values will result in the following call sequence to procedure quicksort:
quicksort(0,9), quicksort(0,), quicksort(0,2), quicksort(1,2), quicksort(2,2). Draw the stack of activation records just when the call to quicksort(2,2) has been made.

(b) Draw the activation tree at this point.