# HW #2 (List of Integers & Ring)

- **Task 1: Design a modular program to solve the following problem.**

- **Write a C++ program that will read in a list of numbers, find the average of all numbers, the average of only positive, the average of only negative numbers, and the largest element.**
- **Your program should contain at least four functions – one to read in the list, one to write it out, one to get the averages, and one to find the largest element.**

# HW #2 (2)

**void ReadList (int Array[ ], int N)**

- **This will read in a list of N values, where N is already known.**

**void Avgs (int Array[ ], int N, int &Ave, int &AveP, int &AveN)**

- **Array is a one-dimensional array of integers and N is the number of elements in that array, Both of these are input parameters to the function. The function must calculate: 1) the average of the N integers in Array and return the result in Ave, 2) the average of the possible numbers (> 0) and return the result in AveP, and 3) the average of the negative numbers (< 0) and return the result in AveN.**

# HW #2 (3)

**int Large (int Array[ ], int N)**

- **Array is a one-dimensional array of signed integers and N is the number of elements in that array. The function returns the value of the largest element in the array.**

**void Display (int Array[ ], int N)**

- **This will display the list of values (nicely formatted) together with the averages and the largest value.**

# HW #2 (4)

- **Use the following test data (there are two sets):**

**A) 4 -30 0 7 42 -20 18 400 -123 -6**

**B) 2 17 -5 0 20 15 -16 -3 -2 14 -1 12 1 -5 -100 15 22 -5 68 -13**

- **Store this information in a file and have your program read this data from that file.**

# HW #2 (5)

- **Task 2: Write a Java function to find the largest subsequence from the input list of integers.**

- **A largest subsequence for a list of integers is a sub-list of integers with the largest possible sum among all the possible sub-lists. For example, for the list**

**4 -30 0 7 42 -20 18 400 -123 -6**

- **The largest subsequence is**

**0 7 42 -20 18 400**

# HW #2 (6)

- **Here is a very simple algorithm to find the largest sub-sequence for a list of size n:**

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        use a loop to add all numbers from i to j and
        save the result to sum.
        keep the largest sum and record the i and j
        values.
    } }
```

**Output the largest sum and all the numbers from i to j.**

# HW #2 (7)

- **Extra bonus: This part is optional. If you can get it done, you will receive extra points.**

- **As you may be able to see, to find the largest subsequence for a list of n numbers, the algorithm given is of $O(n^3)$ time complexity. If possible, find a new algorithm which is substantially faster than the previous algorithm and implement the new algorithm in your program.**

# HW #2 (8)

- **Task 3: In this problem, you are going to implement (using C++ or Java) two functions to build and delete a ring structure, respectively.**
- **You are given the following *RingNode* definition which holds a *char*:**

```
class RingNode { // if you prefer, use struct instead
public:
   char value;        // value stored in the node
   RingNode* next;// pointer to the next RingNode
};
```

# HW #2 (9)

- **You are also given the following function to print the ring, given a pointer to the beginning of the ring (given by *head*) and the size of the ring (given by *size*):**

```
// print the ring
// head: pointer to the beginning of the ring
// size: number of nodes in the ring
void print_the_ring(const RingNode* head, int size) {
    for (int i=0; i<size; i++) {
        cout << head->value << endl;
        head = head->next;
    } }
```

# HW #2 (10)

- **Implement a recursive read function to build a ring given a file stream with the prototype:**

```
int read_recursively (ifstream& fin, RingNode*& current);
```

- **This function takes the *fin* of *ifstream* as input and reads in the first line of the file to build a ring, i.e., it reads character by character of the line until *either* end of file *or* \n is encountered, whichever is earlier.**
- **You can assume that *fin* is opened correctly. The function takes in the RingNode pointer *current*, which has been initialized to NULL.**

# HW #2 (11)

- **The function returns the number of nodes in the ring, with *current* pointing to the node of the first character in the line. If there is no character before end-of-file or \n, then *current* should be NULL.**
- **For example, given the following codes:**

```
int main() {
    ifstream fin("input.txt");
    RingNode* start_point = NULL;
    int total = read_recursively(fin, start_point);
    print_the_ring(start_point, total);
// …
```

# HW #2 (12)

- **If the content of *input.txt* is just a line of *abcdef*, the ring should be like below:**

  **and the output of the program by calling print_the_ring(current, 6) is:**

**a**

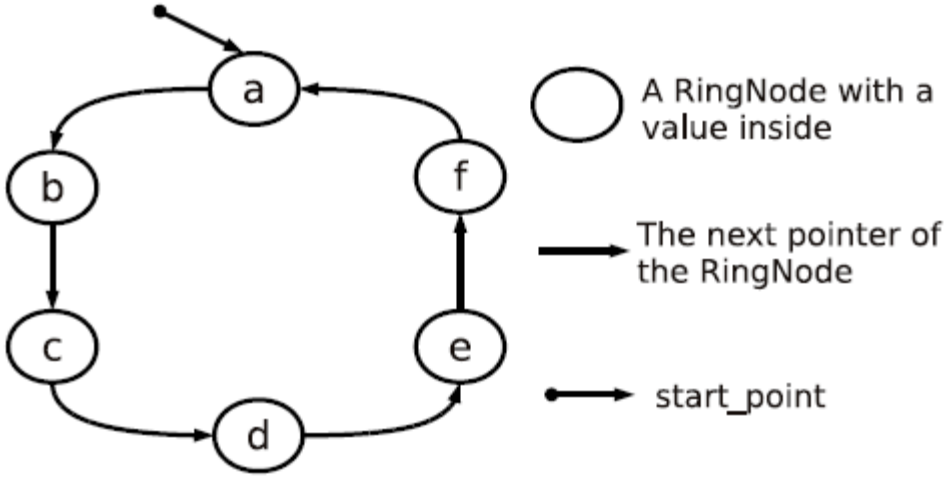
**b**

**c**

**d**

**e**

**f**



# HW #2 (13)

**3.A:**

- **Implement the recursive read function below. You may use**

**ifstream & ifstream::get(char & c);**

- **to read character from an input stream and**

**bool ifstream::eof();**

- **to determine the end-of-file status.**

# HW #2 (14)

```
// Recursive read in character by character of an input
    file
// to build a ring
// fin: input stream
// current: initialized to NULL and points to the first node
// upon exit
// returns the number of nodes in the ring

int read_recursively (ifstream& fin, RingNode*& current) {
```

# HW #2 (15)

**3.B:**

- **Implement a recursive delete function to destroy a ring with prototype**

**void delete_recursively (RingNode*& current);**

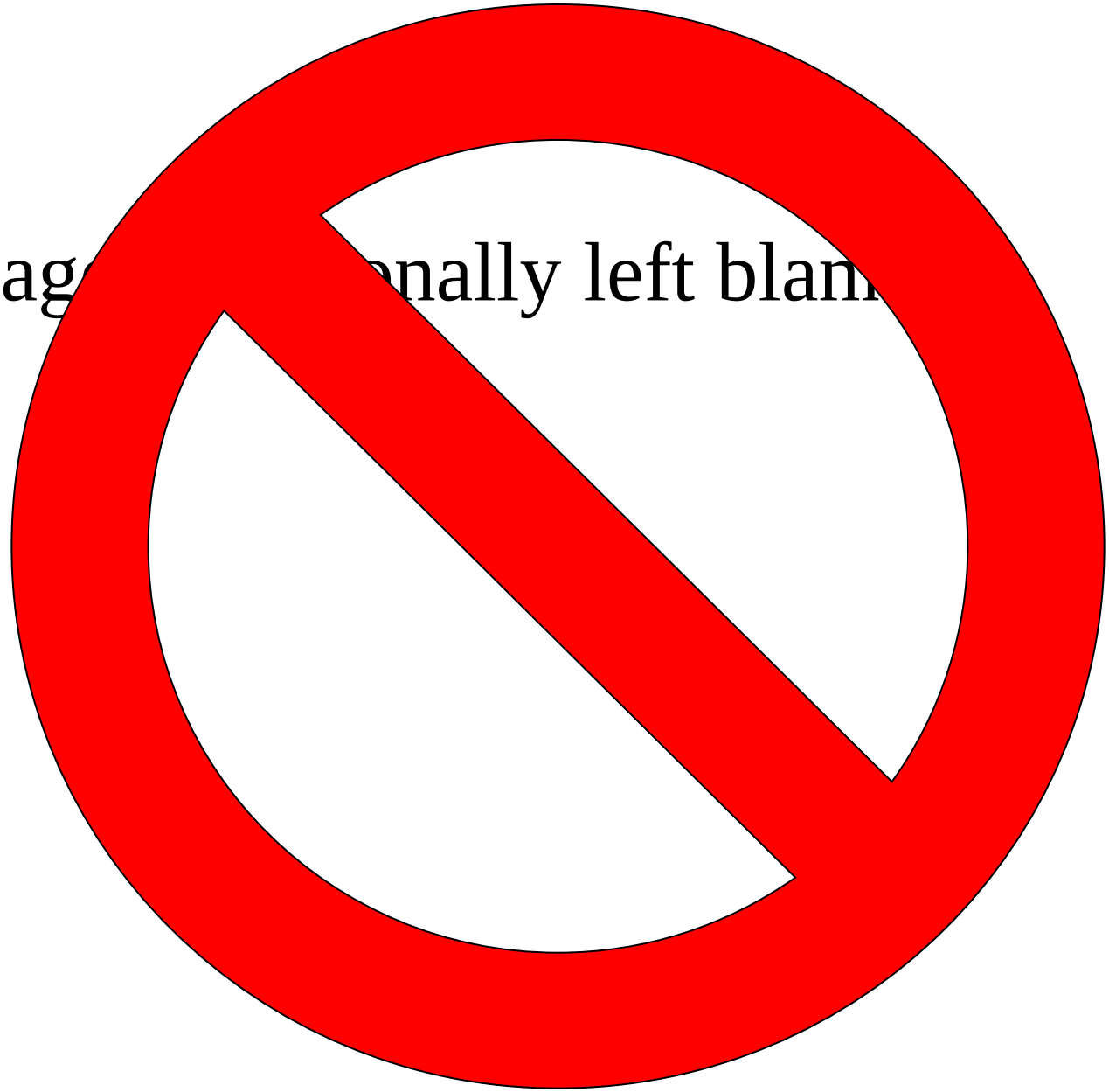
- **The input parameter *current* points to the first node in the ring, and upon function exit, the whole ring is deleted with *current* pointing to NULL. You can assume that *current* is either NULL or pointing to a valid RingNode when the function is firstly invoked.**

**// Recursively delete a ring**

**// current: pointing to the first node or NULL**

**void delete_recursively (RingNode*& current) {**

This pag nally left blan



# Classes: A First Look

```
#include <iostream.h>

#define SIZE 10

// Declare a stack class for characters
class stack {
    char stck[SIZE]; // holds the stack
    int tos;              // index of top-of-stack

public:
    void init();              // initialize stack
    void push(char ch); // push character on stack
    char pop();               // pop character from stack
}
```

```
// Initialize the stack
void stack::init() { tos = 0; }

// Push a character.
void stack::push(char ch) {
    if (tos==SIZE) { cout << “Stack if full”; return; }
    stck[tos] = ch;
    tos++; }

// Pop a character
char stack::pop() {
    if (tos==0) { cout << “Stack is empty”;
                  return 0; // return null on empty stack
                }
    tos--; return stck[tos]; }
```

```
main() {
    stack s1, s2; // create two stacks
    int i;
    // initialize the stacks
    s1.init();
    s2.init();

    s1.push(‘a);          s2.push(‘x’);
    s1.push(‘b’);         s2.push(‘y’);
    s1.push(‘c’);         s2.push(‘z’);

    for (i=0; i<3; i++) cout << “Pop s1: “ << s1.pop() << “\n”;
    for (i=0; i<3; i++) cout << “Pop s2: “ << s2.pop() << “\n”;

    return 0;
}
```