This exam is closed book and notes. There are 4 parts for a total of 100 points. Answer them all on the question sheets. Incomprehensible answers get zero points!

**Part 1 (20 points)**

Complete the C++ function declared `char *reverse(const char *pch)`; so:

1. (4 points) It will call function `size_t strlen(const char *)` to determine the length of the C-string whose address is the value of parameter `pch`, and save this length in variable `mylength`. (Remember that a “C-string” is a null-terminated array of char.)

2. (4 points) It will allocate a dynamic array of char just big enough to hold a C-string with the length determined above, and store its address in the variable named `pnewch`.

3. (10 points) It will store into this dynamic array the C-string obtained by **reversing the order of the characters of the given C-string** (from `pch[]`). (Example: Reversing "ABC123Z" gives "Z321CBA".)

4. (2 points) The function should return the address of the reversed string.

```cpp
#include <cstring>  // Supplies strlen()
#include <cstdlib>  // Supplies size_t
using namespace std;
char * reverse( const char *pch )
{
    size_t mylength;
    char * pnewch;

    mylength = strlen( pch );       //Req. 1 is met.
    pnewch = new char[ mylength + 1]; //Req. 2 is met.
    for( int i = 0; i < mylength; i++ )
    {
        pnewch[ i ] = pch[ mylength - i - 1];
    }   //The chars. are copied into the new array in reversed order.
    pnewch[ mylength ] = '\0';       //New array is now a C-string.
                                       //Req. 3 is met.
    return pnewch;                    //CORRECTED 3/9/04 //Req. 4 is met.
}
```
**Part 2 (20 points)**

The following defines a class that implements a dynamically allocated partially filled array of floats data structure:

```cpp
#include <cstdlib> // Supplies size_t
using namespace std;

class Part2Array {
public:
    Part2Array() //Constructor.
        { data = new float[13];
            capacity = 13;
            used = 0;
        }
    void remove(size_t i);
    //Precondition: 0<=i<=used-1
    //Postcondition: The entry in position i has been removed and all used
    // entries right of position i are shifted left so the used entries
    // continue to be stored in a prefix of the array pointed to by data.
    void reserve(size_t new_capacity);
    //Precondition: new_capacity >= used
    //Postcondition: The capacity of the dynamically allocated array is
    // now new_capacity; and the original contents are preserved.
    ....
    //Various data access functions are not shown.
private:
    float * data;  //Invariant: data!=NULL.
    size_t used;  //Invariant: The array prefix used is data[0..(used-1)].
    size_t capacity;  //Invariant: The number of allocated elements=capacity.
};
```

Write complete and correct implementations of the member functions (a) remove and (b) reserve.

```cpp
void Part2Array::remove(size_t i)  
{  
    for(size_t j = i+1; j<used; j++)  
        { data[j-1] = data[j]; }  
    used--;  
}
```

```cpp
void Part2Array::reserve(size_t nc)  
{  
    float *t = new float[nc];  
    for(size_t j = 0; j<used; j++)  
        { t[j] = data[j]; }  
    delete [] data;  
    capacity = nc;  
    data = t;  
}  
```
Part 3 (25 points)

The node type for a singly-linked-list of floats is defined by:

```c
struct node
{
    float data;
    node *fore; //fore==the address of the next node if any, NULL if not.
};
```

Suppose we have one singly-linked-list whose head and tail node addresses are respectively stored in variables HEAD and TAIL. Suppose cursor points to some node, not the last.

```c
node * HEAD;
node * TAIL;
node * cursor;
```

Assume all nodes are dynamically allocated and should be returned to the “heap” or free-store when they are deleted. Complete the code below to delete (from this singly-linked list) the node just after the node pointed to by cursor:

But first answer here (for 5 points, in English, not C++): What has to be done in the special case when the node to be deleted is the last node in the list?

"The value of TAIL must be changed to the address of the new last node, which is the node that was next-to-last in the list before the deletion."

```c
assert( cursor != NULL && cursor != TAIL );
```

(write the code below..)

```c
node *t;
t = cursor->fore; //t==addr of node N to delete.
cursor->fore = t->fore; //fore of previous node==
    // addr of node after N,
    // or NULL if none.
if( t->fore == NULL )
{ TAIL = cursor; } //See the "answer" above.
delete t; //return deleted node to the free-store.
```
Part 4 (35 points, 7 questions)

1. Draw a “memory picture” or data structure diagram (with boxes, variable names, arrows and values) for what is in memory after the code sequence below finishes running:

```
int I = 5;
int J = 63;
int K;
int * P;
int * Q;
P = &I;
Q = &J;
K = *Q;
*Q = *P;
P = Q;
Q = new int;
*Q = 98;
```

Notice how integer values and pointer value symbols are drawn INSIDE the box representing each variable!!

This clarifies graphically that the VALUE of a variable IS INSIDE the variable, and the computer changes the value WITHOUT changing the identity of the variable.

If you made changes as you worked out this problem (which indeed you SHOULD do), mark clearly what your final picture is! (Ambiguities get 0 points).

2. Why don’t linked lists make use of an operation like “reserve” described in Part 2?

A new linked list node is allocated to provide the memory to contain a new element each time an element is added to a linked list. The “reserve” operation allocates memory for elements to be added in the future, which haven’t been added yet. A “reserve” operation or any other that allocates memory for future elements is not needed for linked lists because linked lists do not allocate memory for future elements before those elements are added.

3. Bozo Simpleton, who didn’t understand how to apply “.h” files, wrote a class definition for class BoSim near the top of the same file in which he wrote the implementations of class BoSim’s member functions. He named that file “BoSim.cxx”.

He eventually corrected some syntax errors and got this implementation file to compile. He then tried to compile the “main” module (in a file named main.cxx) that declared some variables of type BoSim. Since the compiler reported errors, he wrote the definition of class BoSim near the top of main.cxx. Finally main.cxx was compiled, the linking was successful, and the program seemed to run OK.

Explain why what Bozo did is poor software development practice, in terms of what might go wrong in the future.

In the future, Bozo or someone else might edit changes to the BoSim class definition he coded in one of main.cxx or BoSim.cxx but not the
other. Future compilations of both these .cxx files might be successful, but when they are linked and run, the code from the two .cxx files will be based on different definitions of the BoSim class. Bugs that are more or less difficult to detect and track down might result.

If Bozo followed the good practice of putting class definitions in header files only, then every future compilation will use the physically same definition of the BoSim class.

4. This question is about the Project 2 :infile <filename> command. Suppose the file was opened successfully by using istream myfile(filename);. Each line from the file is read using myfile.getline(...) as the condition-expression in a while( ){} or if( ){} statement. The return value of myfile.getline(...) will evaluate to false in two situations. Explain each situation:

Situation 1:

Lines are read from the file one by one. After the last line has been read, the value will be false the next time myfile.getline(...) is run. Project 2 should detect this situation and exit since it is the normal way for the run to finish.

Situation 2:

When the last line read has more than the input array size number of characters including the newline, the return value will be false. This situation results from incorrect input data in the file provided by the user. Project 2 should print an error message and exit.

This explanation is based on the assumption that the input array size was correctly coded as the 2nd argument to myfile.getline(...).

5. In this course’s project and laboratory assignments, what is build.sh for?

To contain and automate running of the g++ compile and link commands, so the program can be repeatedly rebuilt after edits without you repeatedly typing those commands.

6. Explain one advantage of using a C-string, instead of a partially filled char array together with a size_t “used” variable.

The end of a C-string is indicated by the reserved “null” character. Therefore, when a program processes the string’s characters sequentially, it can detect the end of the string directly, without counting down or comparing to a separate variable holding the length.

7. Choose one operation that takes constant time on a linked list but not on a partially filled array (1 pt.). Explain why the amount of time used for this operation is constant with the linked list but is not constant with the partially filled array (4 pt.).

Operation: Inserting a new element just after a position indicated by a pointer to the previous position. The amount of time is constant because
the computer only does one new node allocation, one copy of the data into the new node, and then a (small) constant number of modifications of pointer fields. For a partially filled array, the existing elements after the given position must all be copied one position ahead. The number of these copy operations will be the number of such elements, which is not constant. Also, the array will have to re-allocated and copied completely if it was full before the operation was invoked.