If this lab is an Individual assignment, you must do all coded programs on your own. You may ask others for help on the language syntax, but you must organize and present your own logical solution to the problem. No lab is complete until the MyClass submits the signed pledge form associated with that lab. I realize that no coded programs will be graded until I turn in the sign & pledge form associated with that program; any late penalties will continue to compound until the pledge form is submitted.

If this lab is a team assignment, both team members may share logic as they program side by side on their own computers. Each person must type all of his/her own code as part of the learning process. Team assignments are never to be “You do this portion and I’ll do that portion” or “You do this lab and I’ll do the next lab”.

Some of the lab assignments will have short answer questions. These short answer questions will be spot checked and graded for completion, but not checked for accuracy. Once these labs are graded and returned, I encourage you to compare answers with another class member who has also had the lab graded and returned.

I/We realize that the penalty for turning in work that is not my own, or assisting others in doing so, can range from an "F" in the class to dismissal from Trinity University. I realize that it is a violation of academic integrity to share any portion of this lab with any person (outside my 2320 team & professor)!

Print Name _________________________________________ Time Required = ______.____ Hrs.
Signature ________________________________ (pledged)

OOP-4 Templates & Memory Management Print Only Pages 1-5
Individual Assignment
Answers To Questions 10 Points - Program 15 Points

Short Answer Questions – 5 Points
Electronic Solutions Of Short Answer Questions Will Not Be Accepted. Print A Copy Of Short Answer Question & Write Answers On Printed Copy

Dynamic Memory

1] ________________________________________________________________________________________ Write
   the line of code that will create a short integer, called No

2] ________________________________________________________________________________________ Write
   the line of code that will create a pointer to short integer, called NoPtr

3] ________________________________________________________________________________________ When we create a short integer, such as No above, what value is automatically placed in container No?

4] ________________________________________________________________________________________ When we create a a pointer to a short int, such as NoPtr above, what address is automatically placed in container NoPtr?

5] ________________________________________________________________________________________ Write
   the line of code that will create a short integer, called No and initialize it with 20.

6] ________________________________________________________________________________________ Write
   a print statement that will display the contents of short integer No as illustrated in the display below:

   | No = 20 |

7] ________________________________________________________________________________________ Write
   a print statement that will display the size of a short integer No as illustrated in the display below:

   | size of No = 2 |
8) Write a print statement that will display the address of where `No` is stored in memory (decimal format).

```
Address of No = 3471264
```

9) Write a print statement that will display the address of where `No` is stored in memory (hexadecimal format):

```
Address of No = 28fa48
```

10) Write a print statement that will display the address of where `No` is stored in memory (octal format):

```
Address of No = 10575224
```

11) Write the line of code that will create a pointer to a short integer, called `NoPtr` and make it point at container `No`.

```
NoPtr = &No;
```

12) Write the line of code that will display the address stored in `NoPtr` as illustrated in the display below:

```
Address stored in NoPtr = 3471264
```

13) Write the line of code that will display the size of `NoPtr` as illustrated in the display below:

```
size of NoPtr = 4
```

14) Write a print statement that will display the address of where `No` is stored in memory (hexadecimal format):

```
Address of No = 28fa48
```

15) Write a print statement that will display the short int pointed to by `NoPtr` as illustrated in the display below:

```
Short integer value Pointed to by NoPtr = 20
```

16) Write a statement that will dynamically allocate a short int container that is pointed to by `NoPtr`.

```
short int No new = malloc(2);
```

17) Write the line of code that will display the address stored in `NoPtr` as illustrated in the display below:

```
Address stored in NoPtr = 1586008
```

18) Write a statement that will place 127 in the short integer pointed to by `NoPtr`.

```
NoPtr = 127;
```

19) Write a print statement that will display the short int pointed to by `NoPtr` as illustrated in the display below:

```
Short integer value Pointed to by NoPtr = 27
```

20) Write a line of code that will return the short integer, currently pointed to by `NoPtr`, to the memory manager.

```// free
```
21] Write a statement that will dynamically allocate an array of five short integers that are pointed to by NoPtr.

22] Write a loop-block of code that will fill the short integer array, referenced by NoPtr, with values 50, 100, 150, 200,

23] Write a loop-block of code that will display the contents of the short integer array, referenced by NoPtr, as illustrated in the display below:

---|---------|---------|---------|---------|---------|
    |    5    |    10   |    15   |    20   |    25   |

24] Write a line of code that will return array of short integers, currently pointed to by NoPtr, to the memory manager.

```cpp
class Athlete {
public:
    Athlete(char NewName[] = "", int NewNo = 0, int NewSportNo = 0, double NewEquipmentValue = 0.0);
    Set(char NewName[] = "", int NewNo = 0, int NewSportNo = 0, double NewEquipmentValue = 0.0);
    ~Athlete(void);
private:
    char Name[26];
    long int No,
    SportNo;
    double EquipmentValue;
};

Athlete Sarah ("Sarah", 111, 1, 11.11);
```

25] Assume that you have class athlete. Write the line of code that will create an athlete pointer, called Sarah_Ptr;

26] Write a line of code that will point Sarah_Ptr to Sarah.

27] Write a line of code that allocate one Athlete to be referenced through Sarah_Ptr.

28] Write a line of code that will change the athlete, pointed to by Sarah_Ptr, to SarahLynn, 1111, 1, 99999.99.

29] Write a line of code to display the athlete, pointed to by Sarah_Ptr, to SarahLynn, 1111, 1, 99999.99; the message is to be "This Is Sarah".
30] Write a line of code to free the memory associated with pointer Sarah_Ptr.

31] Write the block of code that will create a block of code to create an athlete pointer, called Team_Ptr.

32] Write a line of code that allocate 100 Athletes to be referenced through Team_Ptr.

33] Write a line of code to free the memory associated with pointer Team_Ptr.

34] Write a line of code that will change the third team member, pointed to by Team_Ptr, to SarahLynn, 1111, 1, 99999.99.

35] Incorrectly referencing memory are ones that reference memory incorrectly.

36] In order to avoid dangling pointers, the constructor should either allocate the needed dynamic memory or set the pointer to NULL.

37] In order to avoid dangling pointers, the constructor should either allocate or set the pointer to NULL.

38] (Shallow/Deep) copy of Dynamic Variables Creates Memory Leak.

39] (Shallow/Deep) makes an actual copy of the data to which the pointers are pointing.

40] Shallow Copy only applies to objects which contain _?_; this shallow copy transfers pointer addresses, but fails to properly manage dynamic memory.

41] In only the space below, describe the **Shallow Copy** and the problems associated with it.

42] In only the space below, describe the **Deep Copy**.

---

**Templates**

1] are a feature of the C++ programming language that allow functions and classes to operate with generic types.

2] Write the prototype for function MySwap used in the template solution (2 lines - see slides).
Templates are not really functions. They are more of a design or pattern for what shall become a function if evoked.

The Compiler Shall Generate the necessary variations of this function during runtime.

Template functions must be placed in {.hpp/.cpp} files!

Template functions may have only one parameter.

---

**Programming Component 15 Points**

1) Make a copy of your project folder. Name the folder `TomH-ListType` (Use your first name and last initial).

```cpp
template <class InfoType>
class ListType
{
public:
    ListType (long int NewMax = 10);
    ~ListType(void);
    bool Empty (void);
    bool Full (void);
    void Display(char Message[] = "");
    bool Append (InfoType NewInfo);
private:
    InfoType *Info;
    long Max,
    ActNo;
};
```

2) The constructor and destructor for ListType were discussed in our slides. Create ListType.hpp and ListType.cpp. Add the Constructor and Destructor. Place function TestListType(void) in the .cpp file. You may choose to use it, or not.

3) Write the code for method Empty. When the Abstract Data Type (ADT) is EMPTY, the ActNo = 0 \rightarrow return true.

4) Write the code for method Full. When the Abstract Data Type (ADT) is FULL, the ActNo = Max \rightarrow return true.

5) Write the code for method Append. We are not going to use element 0 of the array Info. Append shall place the first item in Info[1] \rightarrow The ActNo will be 1.

```cpp
ListType <Student>
Class(4);

Student
John("John Doe", 234, MALE),
Sarah("Sarah Hicks", 345, FEMALE);
Class.Append(Sarah);
```

Should logically do the following:

```
<table>
<thead>
<tr>
<th>1</th>
<th>Sarah Hicks</th>
<th>345</th>
<th>Female</th>
</tr>
</thead>
</table>
ActNo = 1 | Max = 4
```

Append shall place the second item in Info[2] \rightarrow The ActNo will be 2.
Class.Append(John); |

Should logically do the following:

<table>
<thead>
<tr>
<th>2</th>
<th>John Doe</th>
<th>234</th>
<th>Male</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sarah Hicks</td>
<td>345</td>
<td>Female</td>
</tr>
</tbody>
</table>

ActNo = 2     Max = 4

Do not overflow the array. If function Append has room to store the NewInfo, place it in the structure and return SUCCESSFUL; otherwise return UNSUCCESSFUL. Do not include any print statements in the function.

6] Write the code for method Display.
Display the optional message, if any.
Display "==== FULL ====" if full.
Display "==== EMPTY ====" if empty.
If not empty, display a rectangular grid with only the valid data (records 1 – ActNo) in the following format:

<table>
<thead>
<tr>
<th>2</th>
<th>John Doe</th>
<th>234</th>
<th>Male</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sarah Hicks</td>
<td>345</td>
<td>Female</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
</tr>
</tbody>
</table>

Use cout << setw(50) << Info[Pos] ; to display the Info data!

Display the ActNo & Max.

Summary: The following block of code

```
ListType <Student>
Class(4);

Student
    John("John Doe", 234, MALE),
    Sarah("Sarah Hicks", 345, FEMALE);

Class.Append(Sarah);
Class.Append(John);
Class.Display("Class With Sarah & John");
```

Should produce the following:

<table>
<thead>
<tr>
<th>2</th>
<th>John Doe</th>
<th>234</th>
<th>Male</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sarah Hicks</td>
<td>345</td>
<td>Female</td>
</tr>
</tbody>
</table>

ActNo = 2     Max = 4
Your main program shall be:

```cpp
#include "Utilities.hpp"
#include "Student.hpp"
#include "ListType.hpp"

int main(int argc, char * argv[]) {

    ListType <int>
        IntNos;
    ListType <char>
        Chars(5);
    ListType <Student>
        Class(4);

    Student
        Jane("Jane Doe", 123, FEMALE),
        John("John Doe", 234, MALE),
        Sarah("Sarah Hicks", 345, FEMALE);

    puts("\n\n=======================================================================\n
");
    Class.Display("Class Should Be Empty");
    Class.Append(Sarah);
    Class.Display("Class With Sarah");
    Class.Append(John);
    Class.Display("Class With Sarah & John");

    puts("\n\n=======================================================================\n
");
    Chars.Display("Chars Should Be Empty");
    for (int ch = 'A'; ch <= 'F'; ch++)
        if (Chars.Append(ch))
            Chars.Display();
        else
            puts("Overflow Attempt - Chars Is Full");

    puts("\n\n=======================================================================\n
");
    IntNos.Display("IntNos Should Be Empty");
    for (int no = 1; no <= 6; no++)
        if (IntNos.Append(no * 5))
            IntNos.Display();
        else
            puts("Overflow Attempt - IntNos Is Full");

    getchar();
    return(0);
}
```

You may do as much, or as little, testing as you like. You may include TestListType(void) in your .cpp if you like.

Always document your programs as directed previously. I will not always include some documentation blocks for your functions, but I am doing so with this program in hopes that they illustrate the type of documentation you should do on future programs. You may copy & paste.

```
///////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////
// ListType(void) //
///////////////////////////////////////////////////////////////////////////////////
// Purpose : Do all that is necessary to create the ListType Class. Set the    //
// ActNo = 0 //
// Max = NewMax //
// Allocate NewMax + 1 Elements //
// Written By : Dr. Tom Hicks Environment : Windows 7 //
// Date : xx/xx/xxxx Compiler : Visual C++ //
///////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////
```
// Purpose: Do all that is necessary to destroy the ListType object.
// Written By: Dr. Tom Hicks
// Date: xx/xx/xxxx
// Environment: Windows 7
// Compiler: Visual C++

Empty

// Purpose: Explicitly return true if ActNo = 0; otherwise return false.
// Written By: Dr. Tom Hicks
// Date: xx/xx/xxxx
// Environment: Windows 7
// Compiler: Visual C++

Full

// Purpose: Explicitly return true if ActNo >= Max; otherwise return false.
// Written By: Dr. Tom Hicks
// Date: xx/xx/xxxx
// Environment: Windows 7
// Compiler: Visual C++

Append

// Purpose: Append the NewInfo to the next record in the array. Do not use
// record 0 of the Info array. Do not overflow the container.
// If there is room to append the new record, do so and return
// SUCCESSFUL; otherwise return UNSUCCESSFUL.
// Written By: Dr. Tom Hicks
// Date: xx/xx/xxxx
// Environment: Windows 7
// Compiler: Visual C++

Method Display

// Purpose: If there is a message, display it on a line.
// If Full display == FULL == on a line,
// If Empty display == EMPTY == on a line.
// If Not Empty display grid

3 | Data For Record 3
---|-------------------------
2 | Data For Record 2
---|-------------------------
1 | Data For Record 1
---|-------------------------
ActNo = 3 Max = 10

--- Skip A Blank Line ---
==

Program Output

==

Class Should Be Empty
==== EMPTY ====
ActNo = 0  Max = 4

Class With Sarah
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
</tbody>
</table>
ActNo = 1  Max = 4

Class With Sarah & John
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
</tbody>
</table>
ActNo = 2  Max = 4

Chars Should Be Empty
==== EMPTY ====
ActNo = 0  Max = 5

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
</tbody>
</table>
ActNo = 1  Max = 5

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
</tbody>
</table>
ActNo = 2  Max = 5

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
</tbody>
</table>
ActNo = 3  Max = 5

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
3 | C
2 | B
1 | A

ActNo = 4    Max = 5

==== FULL ====

5 | E
4 | D
3 | C
2 | B
1 | A

ActNo = 5    Max = 5

Overflow Attempt - Chars Is Full

=======================================================================

IntNos Should Be Empty

==== EMPTY ====

ActNo = 0    Max = 10

1 | 5

ActNo = 1    Max = 10

2 | 10
1 | 5

ActNo = 2    Max = 10

3 | 15
2 | 10
1 | 5

ActNo = 3    Max = 10

4 | 20
3 | 15
2 | 10
1 | 5
### What To Turn In

--- **No Lab Is Complete Until Both Are Complete** ---

1) **You sign & submit the Pledge form.**
   a) Make sure that all program files have a header box with a purpose that clearly defines what you are accomplishing in this lab.
   b) Make sure that each and every program function has a well-formed documentation box that clearly describes the purpose.
   c) Make sure that each and every program function header box has the appropriate Written By and Date.
   d) Review the Pledge statement
   e) Sign & Pledge
   f) Record the amount of time you think you spent on this lab
   g) Staple all pages of this lab. Fold in half length-wise (like a hot-dog). Put your name on the outside. Place it on the professor desk before the beginning of lecture on the day it is due. The penalty for late homework will not exceed 25% off per day.

2) **Place all programming code associated with this program, if any, in the Professor’s Code Drop Box**
   a) I do not accept programs by mail; do not submit labs via email!

--- **Comments** ---

A) Programs that do not compile are worth little, if anything.
B) If a print statement format is off, the penalties will often be less than the 25% per day late penalty; turn in the lab. You would not be happy if you went to Best Buy and purchased a large screen TV that did everything except show the picture; you would consider it pretty worthless. Most users consider software that does not work properly pretty useless as well. If the lab is not working correctly, credit will be small (if any); you might be better to accept a 25% (1 day) late penalty and turn in the lab working correctly!
C) Start all programs early so that you can get in contact with the professor if you have problems.
D) If you are turning in this lab late, you may
   - hand it to me if I am in the office
   - put it in the mailbox outside my office door
- slide it under the outer door to our suite (if locked)
- slide it under my office door. The sooner I get late labs, the sooner the late penalty meter quits clicking.

E] Backup your programs in at least three places. Put a copy on your Y drive. Put a copy on your flash drive. Put a copy on your personal computer. Send yourself a copy in your e-mail.