Learning Objectives

After the successful completion of this learning unit on Recursion (Gaddis Chapter 20) and Templates (Gaddis Chapter 16), you will be able to:

- To be able to identify the base case(s) and the general case in a recursive definition.
- To be able to write a recursive algorithm for a problem involving only simple variables.
- To be able to write a recursive algorithm for a problem involving structured variables.

- To be able to write a C++ function template.
- To be able to write code that instantiates a function template.
- To be able to write a user-defined specialization of a function template.

OR

- To be able to write a C++ Class template.
- To be able to write code that instantiates a Class template with various types such as int or strings.
- To be able to write a user-defined specialization of a Class template with exception handling routines.

Assignment 8 Part A: Recursive algorithms [20 points]

Among other possibilities, algorithms developed using recursion i.e., the ability of a function to call itself is an alternative control structure to repetition (looping) that can also be used to generate a recursive solution in various program logic implementation routines. Make sure to read and review Gaddis Text: Chapter 20 content on Recursion.

Rather than use a while statement, do-while statement, or for statement to execute a segment of code again, the program uses a selection statement (if or switch statement) to determine whether to repeat the code by calling the function again or to stop the process.

Each recursive solution has at least two cases: the base case and the general case. The base case is the one to which we have an answer; the general case expresses the solution in terms of a call to itself with a smaller version of the problem. Because the general case solves a smaller and smaller version of the original problem, eventually the program reaches the base case where an answer is known and the recursion stops.

Listed below are a few such programming challenges requiring function algorithms with recursive solutions. **Carefully read the program specifications for each recursive set of algorithms and *select any TWO from the three sets provided below.***

(a) Basic Recursion with Simple Types [10 points]

For the first recursive set of algorithms for this assignment ..write a program with two separate recursive void functions to process with an integer. Demonstrate the function algorithms in a driver program.

- a recursive void function that takes a single int argument n and displays the integer 1, 2, 3, ... , n.

```
void writeUp (/*in*/ int n); //a sample function prototype
```

- a recursive void function that takes a single int argument n and displays integers n, n-1, ... , 3, 2, 1.

```
void writeDown (/*in*/ int n); //a sample function prototype
```

Embed the two recursive functions* in a program and write a couple of drivers to test your functions. You may use the sample program interface provided : sample8a.cpp ↓ or create your own. Submit your final source code.

OR

(b) Recursion with String Types [10 points]

For the second recursive set of algorithms for this assignment ..write a program with two separate recursive functions to process characters in a string. Demonstrate the function algorithms in a driver program.

- a recursive void function that accepts a `string` object as its argument and prints the string in reverse order.

```
void strReverse(/*in*/string,/*in*/ int); //a sample function prototype
```

- a `bool` function that uses recursion to determine if a string argument is a palindrome. The function should return `true` if the argument reads the same forward and backward..

```
bool isPalindrome(/*in*/string); //a sample function prototype
```

Embed the two recursive functions* in a program and write a couple of drivers to test your functions. You may use the sample program interface provided : sample8b.cpp or create your own. Submit your final source code.

OR

(c) Recursion with Structured Variables [10 points]

For the third recursive set of algorithms for this assignment ..write a program with two separate recursive value-returning functions to process an array with limit valid elements. Demonstrate the function algorithms in a driver program.

- a recursive value-returning function to return the number of times a specified number occurs in an array.

```
int numTimes (/*inout*/ int *array,/*in*/ int size,/*in*/ int value); //a sample function prototype
```

- a recursive value-returning function to return the largest value in an array.

```
int findLargest (/*in*/ const int array[],/*in*/ int start,/*in*/ int end); //a sample function prototype
```

Embed the two recursive functions* in a program and write a couple of drivers to test your functions. You may use the sample program interface provided : sample8c.cpp or create your own. Submit your final source code.

Note: Points are provided only for functions that include recursive algorithms and meet program specifications.

Assignment 8 Part B: - Function Templates [20 points]

QuickSort Template

Create a template version of the QuickSort algorithm (see Gaddis Chapter 20.8 Focus on Problem Solving and Program Design: The Quick Sort Algorithm) that will work with any data type. Demonstrate the template with a driver function (main) to test it.

Prior to attempting this Function Templates assignment please review Gaddis Chapter 16.2 & 16.3, some helpful notes on Templates Function that will provide a few details with respect to the setup and use of function templates.

```
// This program demonstrates the QuickSort Algorithm.(click to download source code Pr20-11.cpp
          ⬇
         )
#include <iostream>
using namespace std;

// Function prototypes
void quickSort(int [], int, int);
int partition(int [], int, int);
void swap(int &, int &);

int main()
{
const int SIZE = 10; // Array size
int count; // Loop counter
int array[SIZE] = {7, 3, 9, 2, 0, 1, 8, 4, 6, 5};

// Display the array contents.
for (count = 0; count < SIZE; count++)
cout << array[count] << " ";
cout << endl;

// Sort the array.
quickSort(array, 0, SIZE - 1);

// Display the array contents.
for (count = 0; count < SIZE; count++)
cout << array[count] << " ";
cout << endl;
return 0;
}

//**************************************************
// quickSort uses the quicksort algorithm to *
// sort set, from set[start] through set[end]. *
//**************************************************

void quickSort(int set[], int start, int end)
{
int pivotPoint;

if (start < end)
{
// Get the pivot point.
pivotPoint = partition(set, start, end);
// Sort the first sub list.
quickSort(set, start, pivotPoint - 1);
// Sort the second sub list.
quickSort(set, pivotPoint + 1, end);
}
}

//************************************************************
// partition selects the value in the middle of the *
// array set as the pivot. The list is rearranged so *
// all the values less than the pivot are on its left *
// and all the values greater than pivot are on its right. *
//************************************************************

int partition(int set[], int start, int end)
{
int pivotValue, pivotIndex, mid;

mid = (start + end) / 2;
swap(set[start], set[mid]);
pivotIndex = start;
pivotValue = set[start];
for (int scan = start + 1; scan <= end; scan++)
{
if (set[scan] < pivotValue)
{
pivotIndex++;
swap(set[pivotIndex], set[scan]);
}
}
swap(set[start], set[pivotIndex]);
return pivotIndex;
}

//**************************************************
// swap simply exchanges the contents of *
// value1 and value2. *
//**************************************************

void swap(int &value1, int &value2)
{
int temp = value1;

value1 = value2;
value2 = temp;
}
```

For this programming segment of this assignment you may choose to (a) create a separate header file "quicksort.h" which should contain the three function definitions (quickSort, partition and swap) along with the inclusion of a generic function template <class T> above each function heading as well as the use of a type parameter to the function to specify a generic data type or (b) update the source code provided above to include the function template with necessary updates to the function parameters.

With the inclusion of the the function template you may test it with the following function call in main:

```
quickSort<int>(array, 0, SIZE - 1);
```

Turn in the entire program (single file 8d.cpp or with multiple files quicksort.h and 8d.cpp)

Here are a few general steps to follow if you choose to attempt this portion of the assignment.

STEP#1: Setup a new orderedpair class project in your compiler and add the following files: orderedpair.h, orderedpair.cpp and client.cpp.

STEP#2: Compile/run the orderedpair class project a few times to see program output. Review the source code carefully in the specification and implementation file and take note of the class structure, members, exception handling setup and then step through the client code to see how the client code is testing the class implementation. Note this is your original non templated version of the class which will be converted next into a templated version of the class. Close the project.

STEP#3: Create a new class project and call it templateClass. Now add the following files to your new templateClass project: orderedpair.h, orderedpair.cpp and client.cpp. From within the project select each of the files and rename it as follows tclass.h, tclass.cpp and client.cpp. Also rename the #include "orderedpair.h" to "tclass.h" in both the tclass.cpp and client.cpp. You can test compile your templateClass project to make sure it works well with the files renamed.

STEP#4: In your first test you may use int as the type to test your templated class effort. The class specification and implementation will need a few changes and I recommend following the vector example and the six steps in Templated Classes notes page as a guide to convert the current class into a templated class. To help you get started, here is a few examples of changes to be made using template syntax in the both the class specification and implementation file.

Place the template prefix before the class declaration and before each member function definition that occurs outside the class declaration.

```
Templated Class specification file example
namespace cs_pairs {
        template <class T>  //template prefix
        class OrderedPair {
```

```
Templated Class implementation file example
namespace cs_pairs {
        template <class T>
       OrderedPair<T>::OrderedPair(T newFirst, T newSecond)throw(DuplicateMemberError) {
        setFirst(newFirst);
        setSecond(newSecond);
    }
```

Place a <T> after each occurrence of the class name that occurs outside the class declaration.

For example, in the case of OrderedPair, that means that almost every time you see the word " OrderedPair " outside of the class declaration, you'll put a <T> after it.

```
OrderedPair<T>::OrderedPair
```

The only exception is when the word is not used as the name of a class, but rather as the name of a function, as is the case with the constructor.

Replace each occurrence of the original type such as "int" with "T"

```
OrderedPair<T>::OrderedPair(T newFirst, T newSecond)throw(DuplicateMemberError) {
```

The last three program statements in the tclass.cpp (templated class implementation) will look something like this...

```
    template <class T>
    const T OrderedPair<T>::DEFAULT_VALUE = T();
    template class OrderedPair<int>; // change to template class OrderedPair<string>;
} //namespace cs_pairs end
```

Once you have this first templated class working with ints next you will get to try working with strings by first setting up the client code to process with strings. I have provided two client code source code examples with changes made to facilitate testing the templated class with int and string data types that you may utilize in your project. One will process for int (intclient.cpp) and the other will process with strings (stringclient.cpp). You are welcome to write your own client code to test drive your templated class. Hope you get to see the benefits of templated container classes.

Turn in the entire project files (multiple files tclass.h, tclass.cpp and client.cpp)

Submit Your Work

You will be submitting several files for this assignment, so please ensure the correct files are uploaded for each part of this assignment.