```
Script started on 2022-05-20 08:08:52-05:00 [TERM="xterm" TTY="/dev/pts/5" COLUMNS=
a_vitale7@ares:~$ pwd
/home/students/a_vitale7
a_vitale7@ares:~$ cat applyfnc.info
/*************************************************************************
 *                                                                       *
 *  NAME:  Antonino Vitale                       CLASS:  CSC122-W01      *
 *                                                                       *
 *  Lab:  Why can't you just do what I TELL you to?                      *
 *                                               Level:  3               *
 *                                                                       *
 *  Description:                                                         *
 *                                                                       *
 *     This program shows examples of how the genarr.h library is used   *
 *     for manipulating entire arrays with functions by passing          *
 *     functions as arguments.                                           *
 *                                                                       *
 *************************************************************************/
a_vitale7@ares:~$ cat applyfnc.cpp
#include <iostream>

#include <string>

#include <cstring>

#include "genarr.h"



using namespace std;



const size_t MAX = 16;



template < typename DataType > inline void output(const DataType& data, ostream& os

    os << data << endl;

}

template < class ArrT > inline void output(const ArrT& arr, long size, ostream& os

    os << "{ ";

    for (long i = 0; i < size; i++) {

        os << arr[i] << ((i != size - 1) ? (", ") : (" }"));

    };

    os << endl;

}



//make into functions print, add, uppercase

// for applying

template < typename DataType > void print(DataType& data) { cout << data << " | ";

template < typename DataType > void addfour(DataType& x) { x += 4; return; }

template < typename DataType > void uppercase(DataType& data) { data = static_cast<

// for accumulating

template < typename DataType > DataType add(DataType& x, DataType& y) { return x +

template < typename DataType > DataType subtract(DataType& x, DataType& y) { return

string append(const string& x, const string& y) { return x + ' ' + y; }

//make into objects print, add, sum

template < typename DataType > class example_print

{

public:

    example_print(void) { return; }

    ostream& operator() (DataType output, ostream& os = std::cout) { return os << o

};

template < typename DataType > class example_add

{

    DataType addend = 0;

public:

    example_add(DataType add = 0) { addend = add; }

    example_add(const example_add& add) { addend = add.addend; }
```

```
    void operator() (DataType& add_to) { add_to += addend; return; }
};
template < typename DataType > class example_sum
{
    DataType sum = 0;
public:
    example_sum(DataType s = 0) : sum{ s } { }
    DataType operator ( ) (void) const { return sum; }
    DataType operator ( ) (DataType data) { sum += data; return sum; }
    DataType operator ( ) (DataType* arr) { for (auto& i : arr) { sum += i; } retu
    void reset(DataType s = 0) { sum = s; return; }
    friend ostream& operator<<(ostream& out, const example_sum& s) { out << s.sum;
};

int main()
{
    short output_MAX = 3; //outputs always output the first three of an array, to s
    cout << "\n\tIntegers" << endl;
    short int_total;
    short int_arr[MAX];
    for (unsigned short i = 0; i < MAX; i++) {
        int_arr[i] = i;
    }
    //  Integers
    //Print
    cout << "print() ";
    apply_fauto(int_arr, print<short>);
    cout << endl;
    output(int_arr, output_MAX);
    //Add 4
    cout << "add() ";
    apply_fauto(int_arr, addfour<short>);
    cout << endl;
    output(int_arr, output_MAX);
    //Obj Print
    cout << "example_print() ";
    apply_cauto(int_arr, example_print<short>());
    cout << endl;
    output(int_arr, output_MAX);
    //Obj Add 4
    cout << "example_add(4) ";
    apply_cauto(int_arr, example_add<short>(4));
    cout << endl;
    output(int_arr, output_MAX);
    //Obj Sum
    cout << "example_sum() ";
    apply_cauto(int_arr, example_sum<short>());
    cout << endl;
    output(int_arr, output_MAX);
    int_total = accumulate_cauto(int_arr, add<short>, static_cast<short>(0)); //Ex
    cout << "total: ";
    output(int_total);

    cout << "\n\tDoubles" << endl;
```

```
double double_total;
double double_arr[MAX];
for (unsigned short i = 0; i < MAX; i++) {
    double_arr[i] = i + 0.1*i + 0.01;
}
//  Doubles
//Print
cout << "print() ";
apply_fauto(double_arr, print<double>);
cout << endl;
output(double_arr, output_MAX);
//Add 4
cout << "add() ";
apply_fauto(double_arr, addfour<double>);
cout << endl;
output(double_arr, output_MAX);
//Obj Print
cout << "example_print() ";
apply_cauto(double_arr, example_print<double>());
cout << endl;
output(double_arr, output_MAX);
//Obj Add 4
cout << "example_add(4.0) ";
apply_cauto(double_arr, example_add<double>(4.0));
cout << endl;
output(double_arr, output_MAX);
//Obj Sum
cout << "example_sum() ";
apply_cauto(double_arr, example_sum<double>());
cout << endl;
output(double_arr, output_MAX);
double_total = accumulate_cauto(double_arr, add<double>, 0.0); //Extra
cout << "total: ";
output(double_total);

cout << "\n\tCharacters" << endl;
double char_total;
char char_arr[MAX];
for (unsigned short i = 0; i < MAX; i++) {
    char_arr[i] = static_cast<char>(97 + i);
}
//  Characters
//Print
cout << "print() ";
apply_fauto(char_arr, print<char>);
cout << endl;
output(char_arr, output_MAX);
//Add 4
cout << "add() ";
apply_fauto(char_arr, addfour<char>);
cout << endl;
output(char_arr, output_MAX);
//Uppercase
cout << "uppercase() ";
```

```
apply_fauto(char_arr, uppercase<char>);
cout << endl;
output(char_arr, output_MAX);
//Obj Print
cout << "example_print() ";
apply_cauto(char_arr, example_print<char>());
cout << endl;
output(char_arr, output_MAX);
//Obj Add 4
cout << "example_add(4) ";
apply_cauto(char_arr, example_add<char>(4));
cout << endl;
output(char_arr, output_MAX);
//Obj Sum
cout << "example_sum() ";
apply_cauto(char_arr, example_sum<char>());
cout << endl;
output(char_arr, output_MAX);
char_total = accumulate_cauto(char_arr, add<char>, ' '); //Extra
cout << "total: ";
output(char_total);

cout << "\n\tStrings" << endl;
string str_arr[MAX] = {"zero", "one", "two", "three", "four", "five", "six", "s
//  Strings
//Print
cout << "print() ";
```

```
apply_fauto(str_arr, print<string>);
cout << endl;
output(str_arr, output_MAX);
//Add 4
cout << "add() ";
apply_fauto(str_arr, addfour<string>);
cout << endl;
output(str_arr, output_MAX);
//Obj Print
cout << "example_print() ";
apply_cauto(str_arr, example_print<string>());
cout << endl;
output(str_arr, output_MAX);
//Obj Add 4
//terminate called after throwing an instance of 'std::logic_error' what():  ba
/* removing due to error
cout << "example_add(\"4\") ";
apply_cauto(str_arr, example_add<string>("4"));
cout << endl;
output(str_arr, output_MAX);
*/
//Obj Sum
cout << "example_sum() ";
sentence = accumulate_cauto(str_arr, append, string()) + '.';
cout << endl;
cout << "Sentence: ";
output(sentence);
```

```
        cout << "\nEnd of program" << endl;
        return 0;
}
a_vitale7@ares:~$ cat genarr.h
#ifndef GENARIC_ARRAY_PROCESSING_FUNCTIONS_TEMPLATE_LIBRARY
#define GENARIC_ARRAY_PROCESSING_FUNCTIONS_TEMPLATE_LIBRARY

//      The Apply and Accumulate functions
template < typename ArrT, typename FuncT > void apply_cauto(ArrT& arr, FuncT func)
    for (auto& i : arr) {
        func(i);
    }
    return;
}
template < typename ArrT, typename FuncT > void apply_cmanual(ArrT& arr, size_t si
    for (unsigned long i = 0; i < size; i++) {
        func(arr[i]);
    }
    return;
}
template < typename ArrT, typename DataType > void apply_fauto(ArrT& arr, void fun
    for (auto& i : arr) {
        func(i);
    }
    return;
}
```

```
template < typename ArrT, typename DataType > void apply_fmanual(ArrT& arr, size_t
    for (unsigned long i = 0; i < size; i++) {
        (*func)(arr[i]);
    }
    return;
}

template < typename ArrT, typename FuncT, typename ClassFuncT > ClassFuncT accumul
    ClassFuncT temp = arr[0];
    for (auto& i : arr) {
        temp = func(temp, i);
    }
    return temp;
}
template < typename ArrT, typename FuncT, typename ClassFuncT > ClassFuncT accumul
    ClassFuncT temp = arr[0];
    for (unsigned long i = 1; i < size; i++) {
        temp = func(temp, arr[i]);
    }
    return temp;
}
template < typename ArrT, typename FuncT, typename ClassFuncT > ClassFuncT accumul
    ClassFuncT temp = arr[0];
    for (auto& i : arr) {
        temp = func(temp, i);
    }
    return temp;
```

```
}

template < typename ArrT, typename FuncT, typename ClassFuncT > ClassFuncT accumula

    ClassFuncT temp = arr[0];

    for (unsigned long i = 1; i < size; i++) {

        temp = func(temp, arr[i]);

    }

    return temp;

}



#endif

a_vitale7@ares:~$ CPP applyfnc.cpp genarr.h
applyfnc.cpp***


a_vitale7@ares:~$ ./applyfnc.out

        Integers
print() 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
{ 0, 1, 2 }
add()
{ 4, 5, 6 }
example_print() 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
{ 4, 5, 6 }
example_add(4)
{ 8, 9, 10 }
example_sum()
{ 8, 9, 10 }
total: 256

        Doubles
print() 0.01 | 1.11 | 2.21 | 3.31 | 4.41 | 5.51 | 6.61 | 7.71 | 8.81 | 9.91 | 11.0
{ 0.01, 1.11, 2.21 }
add()
{ 4.01, 5.11, 6.21 }
example_print() 4.01 5.11 6.21 7.31 8.41 9.51 10.61 11.71 12.81 13.91 15.01 16.11
{ 4.01, 5.11, 6.21 }
example_add(4.0)
{ 8.01, 9.11, 10.21 }
example_sum()
{ 8.01, 9.11, 10.21 }
total: 268.17

        Characters
print() a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p |
```

```
{ a, b, c }
add()
{ e, f, g }
uppercase()
{ E, F, G }
example_print() E F G H I J K L M N O P Q R S T
{ E, F, G }
example_add(4)
{ I, J, K }
example_sum()
{ I, J, K }
total: 81

        Strings
print() zero | one | two | three | four | five | six | seven | eight | nine | ten
{ zero, one, two }
add()
{ zero, one, two }
example_print() zero one two three four five six seven eight nine ten eleven twelve
{ zero, one, two }
example_sum()
Sentence: zero zero one two three four five six seven eight nine ten eleven twelve

End of program
a_vitale7@ares:~$ ./applyfnc.out

        Integers
print() 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
{ 0, 1, 2 }
add()
{ 4, 5, 6 }
example_print() 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
{ 4, 5, 6 }
example_add(4)
{ 8, 9, 10 }
example_sum()
{ 8, 9, 10 }
total: 256

        Doubles
print() 0.01 | 1.11 | 2.21 | 3.31 | 4.41 | 5.51 | 6.61 | 7.71 | 8.81 | 9.91 | 11.0
{ 0.01, 1.11, 2.21 }
add()
{ 4.01, 5.11, 6.21 }
example_print() 4.01 5.11 6.21 7.31 8.41 9.51 10.61 11.71 12.81 13.91 15.01 16.11
{ 4.01, 5.11, 6.21 }
example_add(4.0)
{ 8.01, 9.11, 10.21 }
example_sum()
{ 8.01, 9.11, 10.21 }
total: 268.17

        Characters
print() a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p |
```

```
{ a, b, c }
add()
{ e, f, g }
uppercase()
{ E, F, G }
example_print() E F G H I J K L M N O P Q R S T
{ E, F, G }
example_add(4)
{ I, J, K }
example_sum()
{ I, J, K }
total: 81

        Strings
print() zero | one | two | three | four | five | six | seven | eight | nine | ten
{ zero, one, two }
add()
{ zero, one, two }
example_print() zero one two three four five six seven eight nine ten eleven twelve
{ zero, one, two }
example_sum()
Sentence: zero zero one two three four five six seven eight nine ten eleven twelve

End of program
a_vitale7@ares:~$ cat applyfnc.tpq
/*************************************************************************
 *                                                                       *
 *     How many template functions did you write? How many instantiations *
 *     of these functions are in your program's compiled binary? How many *
 *     existed during the compilation of your program?                   *
 *          I wrote applys and accumulates for both class object methods  *
 *          and functions to be used as arguments.                       *
 *          I wrote double the nessacary functions because the manual    *
 *          allows the function to repeat to a certain stop point, and   *
 *          the auto will automatically do the entire array.             *
 *          In total there are eight functions in the library.           *
 *                                                                       *
 *     What are the 'names' of the instantiations of your apply function  *
 *     in this program? Your accumulate function?                        *
 *          "apply_", then either "f" for a funnction or "c" for a class  *
 *          method, then either "auto" to automatically do the entire    *
 *          array or "manual" to a specified end.                        *
 *                                                                       *
 *     Does the size of your .out file change if you comment out one of   *
 *     the calls to the apply? Why/Why not? (Hint: 'ls -l *.out' at the $ *
 *     prompt will show the size of all the .out files on your account.)  *
 *          yes? the argument function is not called an amount of times   *
 *          equal to the size of an array?                               *
 *                                                                       *
 *                                                                       *
 *************************************************************************/
a_vitale7@ares:~$ exit
exit
```

```
Script done on 2022-05-20 08:11:20-05:00 [COMMAND_EXIT_CODE="0"]
```