```
Script started on 2024-05-10 10:15:53-05:00 [TERM="xterm" TTY="/dev/pts/0" COLUMNS=
ee43254@ares:~$ pwd
/home/students/ee43254
ee43254@ares:~$ cat datalist.info
Name: Kyle Enkhzul

Class: CSC122-W01



Activity: How many items are in YOUR list?

Level: 9, 3.5 (base program) + 1.5 (automatic shrink),

    + 1.5 (automatic growth) + 2.5 (overloaded operators)



Description:



This lab allows the coder to show mastery of dynamic arrays, classes,

and libraries by creating a general 'template' or 'multipurpose' generic

data classes for whatever the programmer may need. It is meant to be

dynamic, memory efficient, and work for the programmers needs.

ee43254@ares:~$ show-code datalist.h


datalist.h:
```

```
 1  #ifndef DATALIST_H
 2  #define DATALIST_H
 3
 4  #include <iostream>
 5
 6  class GenericList {
 7    double *list;       // dynamic array of double data
 8    long physical_size; // maximum size of the list
 9    long logical_size;  // number of array positions filled
10
11  public:
12    // Constructor
13    GenericList(long max_size);
14
15    // Destructor
16    ~GenericList();
17
18    // Copy constructor
19    GenericList(const GenericList &other);
20
21    // Assignment operator
22    GenericList &operator=(const GenericList &other);
23
24    // Adds a value to the list
25    void add_value(double value);
26
27    // Checks if the list is full
28    bool full() const;
29
30    // Returns the number of values in the list
31    long get_size() const;
32
33    // Returns the value at the specified position
34    double get_value(long position) const;
35
36    // Returns a reference to the last value in the list
37    double get_last() const;
38
39    // Deletes the last value from the list
40    void delete_last();
41
42    // Outputs the values in the list
43    void output(std::ostream &outs) const;
44
45    // Overload the [] operator for element access
46    double& operator[](long index);
47
48    // Overload the += operator for adding values to the list
49    GenericList& operator+=(double value);
50
51    // Declare the << operator as a friend function
52    friend std::ostream& operator<<(std::ostream& os, const GenericList& list
53  };
54
55  #endif
```

```
ee43254@ares:~$ show-code datalist.cpp


datalist.cpp:
```

```
 1  #include "datalist.h"
 2
 3  #include <iostream>
 4  #include <cstdlib>
 5
 6  using namespace std;
 7
 8  // Initializes the object to an empty list with specified size.
 9  GenericList::GenericList(long max_size) :
10   list(new double[max_size]), physical_size(max_size), logical_size(0) {}
11
```

```
// Destructor to deallocate dynamic memory.
GenericList::~GenericList()
{
    delete[] list;
}

// Copy constructor.
GenericList::GenericList(const GenericList &other)
 : list(new double[other.physical_size]), physical_size(other.physical_siz
   logical_size(other.logical_size)
{
    for (long i = 0; i < logical_size; ++i)
    {
        list[i] = other.list[i];
    }
}

// Assignment operator.
GenericList &GenericList::operator=(const GenericList &other)
{
    if (this != &other)
    {
        delete[] list;
        physical_size = other.physical_size;
        logical_size = other.logical_size;
        list = new double[physical_size];
        for (long i = 0; i < logical_size; ++i)
        {
            list[i] = other.list[i];
        }
    }
    return *this;
}

// Overload the [] operator for element access
    double& GenericList::operator[](long index) {
    if (index < 0 || index >= logical_size) {
        cerr << "Error: Index out of range" << endl;
        exit(1);
    }
    return list[index];
}

// Define the operator+= member function inside the class
GenericList& GenericList::operator+=(double value) {
    add_value(value);
    return *this;
}

// Define the << operator as a friend function
ostream& operator<<(ostream& os, const GenericList& list) {
    for (long i = 0; i < list.logical_size; ++i) {
        os << list.list[i] << " ";
    }
    return os;
    }

// Precondition:  The list is not full.
// Postcondition:  The value has been added to the END of the list,
//                 if there was room. If the list is full, it will be resi
void GenericList::add_value(double value)
{
    if (full())
    {
        // Resize the list
        long new_physical_size = (physical_size == 0) ? 1 : physical_size
        double *new_list = new double[new_physical_size];
        for (long i = 0; i < logical_size; ++i)
        {
            new_list[i] = list[i];
        }
        delete[] list;
        list = new_list;
        physical_size = new_physical_size;
    }
    list[logical_size++] = value;
}

// Returns true if the list is full, false otherwise.
bool GenericList::full() const
{
    return (logical_size == physical_size);
}

// Returns the number of values in the list.
long GenericList::get_size() const
{
    return logical_size;
}

// Precondition:  0 <= position < get_size()
// Returns the value at specified position.
double GenericList::get_value(long position) const
{
    return ((position >= logical_size) || (position < 0)) ? (0.0) :
                (list[position]);
}

// Returns a copy of the last value in the list.
double GenericList::get_last() const
{
    if (logical_size == 0)
        return 0.0; // or NaN, depending on your preference
    return list[logical_size - 1];
}

// Deletes the last value from the list.
// If the logical size becomes significantly smaller than the physical size
```

```
120 // the list is resized to release the wasted elements.
121 void GenericList::delete_last()
122 {
123   if (logical_size > 0)
124     {
125      --logical_size;
126
127 // Check if the logical size is significantly smaller than the physical si
128      if (logical_size < physical_size / 2)
129      {
130  // Resize the list
131   long new_physical_size = (physical_size == 0) ? 0 : physical_size / 2;
132       if (new_physical_size == 0) {
133        delete[] list;
134         list = nullptr;
135     } else {
136       double *new_list = new double[new_physical_size];
137        for (long i = 0; i < logical_size; ++i)
138        {
139         new_list[i] = list[i];
140        }
141       delete[] list;
142       list = new_list;
143      }
144      physical_size = new_physical_size;
145     }
146   }
147 }
148
149 // Precondition:  If outs is a file output stream, then outs has
150 //                already been connected to a file.
151 // Postcondition:  Values are output one per line on the stream.
152 void GenericList::output(ostream &outs) const
153 {
154     for (long i = 0; i < logical_size; ++i)
155     {
156         outs << list[i] << '\n';
157     }
158 }
159
160 int main() {
161   const long MAX_SIZE = 5; // Maximum size of the list
162
163   // Create a list with maximum size MAX_SIZE
164   GenericList list(MAX_SIZE);
165
166   cout << "Creating a list with a maximum size of: " << MAX_SIZE << endl;
167
168   // Add some values to the list
169   list += 10.5;  // Using +=
170   list += 20.3;
171   list.add_value(15.7); // Using add_value
172   list.add_value(30.2);
173   list.add_value(25.9);
174
175    // Output the values in the list using <<
176    cout << "Values in the list: " << list << endl;
177
178    // Test element access using []
179    cout << "Value at index 2: " << list[2] << endl;
180
181    // Test deleting the last value
182    list.delete_last();
183    cout << "After deleting the last value: " << list << endl;
184
185    // Test full
186    cout << "Is the list full? " << (list.full() ? "Yes" : "No") << endl;
187
188      return 0;
189  }
ee43254@ares:~$ CPP datalist
datalist.cpp***


ee43254@ares:~$ ./datalist.out
Creating a list with a maximum size of: 5
Values in the list: 10.5 20.3 15.7 30.2 25.9
Value at index 2: 15.7
After deleting the last value: 10.5 20.3 15.7 30.2
Is the list full? No
ee43254@ares:~$ exit
exit

Script done on 2024-05-10 10:16:15-05:00 [COMMAND_EXIT_CODE="0"]
```