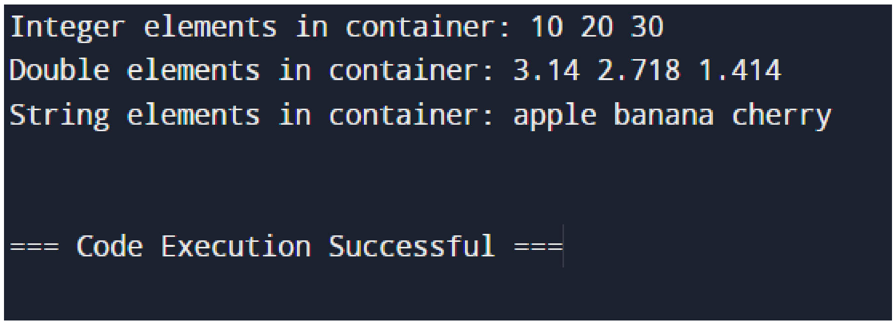
Palindrome String

```cpp
#include <iostream>
#include <string>
using namespace std;

template <typename T>
bool isPalindrome(T str) {
    int n = str.length();
    for(int i = 0; i < n / 2; i++) {
        if(str[i] != str[n - i - 1]) {
            return false;
        }
    }
    return true;
}

int main() {
    std::string str1 = "madam";


    if(isPalindrome(str1))
        cout << str1 << " is a palindrome." << endl;
    else
        cout << str1 << " is not a palindrome." << endl;


    return 0;
}
```

here's **a basic implementation** of a C++ class called Container that can store elements of any type (integer, double, or string). It includes member functions to perform operations such as setters, getters, insertion, removal, and searching.

```cpp
#include <iostream>
#include <string>

const int MAX_SIZE = 100; // Maximum size of the container
using namespace std;

class Container {
private:
    int intSize;
    int doubleSize;
    int stringSize;
    int intElements[MAX_SIZE];
    double doubleElements[MAX_SIZE];
    string stringElements[MAX_SIZE];

public:
    Container() : intSize(0), doubleSize(0), stringSize(0) {}

    // Setter for integer elements
    void setIntElements(const int* newElements, int newSize) {
        intSize = min(newSize, MAX_SIZE);
        for (int i = 0; i < intSize; ++i) {
            intElements[i] = newElements[i];
        }
    }

    // Setter for double elements
    void setDoubleElements(const double* newElements, int newSize) {
        doubleSize = min(newSize, MAX_SIZE);
        for (int i = 0; i < doubleSize; ++i) {
            doubleElements[i] = newElements[i];
        }
    }

    // Setter for string elements
    void setStringElements(const std::string* newElements, int newSize) {
        stringSize = min(newSize, MAX_SIZE);
        for (int i = 0; i < stringSize; ++i) {
            stringElements[i] = newElements[i];
        }
    }

    // Getter for integer elements
    int* getIntElements() {
        return intElements;
    }

    // Getter for double elements
    double* getDoubleElements() {
        return doubleElements;
    }

    // Getter for string elements
    string* getStringElements() {
        return stringElements;
    }

    // Getter for sizes
    int getIntSize() const {
        return intSize;
    }

    int getDoubleSize() const {
        return doubleSize;
    }

    int getStringSize() const {
        return stringSize;
    }

    // Insertion for integer elements
    void insertInt(int item) {
        if (intSize < MAX_SIZE) {
            intElements[intSize++] = item;
        } else {
            cout << "Container is full. Cannot insert." << endl;
        }
    }

    // Insertion for double elements
    void insertDouble(double item) {
        if (doubleSize < MAX_SIZE) {
            doubleElements[doubleSize++] = item;
        } else {
            cout << "Container is full. Cannot insert." << endl;
        }
    }

    // Insertion for string elements
    void insertString(const std::string& item) {
        if (stringSize < MAX_SIZE) {
            stringElements[stringSize++] = item;
        } else {
            cout << "Container is full. Cannot insert." << endl;
        }
    }

    // Removal for integer elements
    bool removeInt(int item) {
        for (int i = 0; i < intSize; ++i) {
            if (intElements[i] == item) {
                for (int j = i; j < intSize - 1; ++j) {
                    intElements[j] = intElements[j + 1];
                }
                intSize--;
                return true;
            }
        }
        return false;
    }

    // Removal for double elements
    bool removeDouble(double item) {
        for (int i = 0; i < doubleSize; ++i) {
            if (doubleElements[i] == item) {
                for (int j = i; j < doubleSize - 1; ++j) {
                    doubleElements[j] = doubleElements[j + 1];
                }
                doubleSize--;
                return true;
            }
        }
        return false;
    }

    // Removal for string elements
    bool removeString(const std::string& item) {
        for (int i = 0; i < stringSize; ++i) {
            if (stringElements[i] == item) {
                for (int j = i; j < stringSize - 1; ++j) {
                    stringElements[j] = stringElements[j + 1];
                }
                stringSize--;
                return true;
            }
        }
        return false;
    }

    // Searching for integer elements
    bool searchInt(int item) const {
        for (int i = 0; i < intSize; ++i) {
            if (intElements[i] == item) {
                return true;
            }
        }
        return false;
    }

    // Searching for double elements
    bool searchDouble(double item) const {
        for (int i = 0; i < doubleSize; ++i) {
            if (doubleElements[i] == item) {
                return true;
            }
        }
        return false;
    }

    // Searching for string elements
    bool searchString(const std::string& item) const {
        for (int i = 0; i < stringSize; ++i) {
            if (stringElements[i] == item) {
                return true;
            }
        }
        return false;
    }
};

int main() {
    Container container;

    container.insertInt(10);
    container.insertInt(20);
    container.insertInt(30);

    cout << "Integer elements in container: ";
    int* intElements = container.getIntElements();
    for (int i = 0; i < container.getIntSize(); ++i) {
        cout << intElements[i] << " ";
    }
    cout << endl;

    container.insertDouble(3.14);
    container.insertDouble(2.718);
    container.insertDouble(1.414);

    cout << "Double elements in container: ";
    double* doubleElements = container.getDoubleElements();
    for (int i = 0; i < container.getDoubleSize(); ++i) {
        cout << doubleElements[i] << " ";
    }
    cout << endl;

    container.insertString("apple");
    container.insertString("banana");
    container.insertString("cherry");

    cout << "String elements in container: ";
    string* stringElements = container.getStringElements();
    for (int i = 0; i < container.getStringSize(); ++i) {
        cout << stringElements[i] << " ";
    }
    cout << endl;

    return 0;
}
```



```
Integer elements in container: 10 20 30
Double elements in container: 3.14 2.718 1.414
String elements in container: apple banana cherry


=== Code Execution Successful ===
```

Sample code making a class reusable across data types

```
-----------------------------------  Pair.h file -------------------------------
-------------------
#ifndef PAIR_H
#define PAIR_H
#include<iostream>
using namespace std;
```

```
template <typename T1, typename T2>
class Pair
{
    template<typename T1, typename T2>
    friend ostream& operator<<(ostream &out, Pair<T1, T2>&);
public:
    Pair();
    //Pair(const Pair&);
    Pair(T1, T2);
    void print();
    ~Pair();
private:
    T1 value1;
    T2 value2;

};
#endif
```

Cpp

```
template <typename T1, typename T2>
Pair<T1, T2>::Pair()
{
    value1=0;
    value2=0;
}
template <typename T1, typename T2>
Pair<T1, T2>::~Pair()
{
    cout<<"\ndestructor of Pair"<<endl;
}

template <typename T1, typename T2>
Pair<T1, T2>::Pair(T1 x, T2 y)
{
    cout<<"Parameterized constructor called"<<endl;
    value1= x;
    value2=  y;

}
//template <typename T1, typename T2>
//Pair<T1, T2>::Pair(const Pair& pr)
//{
```

```
//    cout<<"Copy constructor called"<<endl;
//    value1=pr.value1;
//    value2=pr.value2;
//}
template <typename T1, typename T2>
void Pair<T1, T2>::print()
{
    if(value1 && value2)
       cout<< value1<<" "<<value2<<endl;
}
template<typename T1, typename T2>
ostream& operator<<(ostream &out, Pair<T1, T2>& p)
{
       out<<p.value1<<" "<<p.value2;
    return out;
}
```

Driver

```
------------------------------------ Driver.cpp -----------------------
#include "Pair.cpp"
using namespace std;
int main()
{

    Pair<int, double> * p= new Pair<int, double>(1, 2.2);
    Pair<float, string> p2(2.8, "Hello");

    cout<<(*p)<<endl;
    cout<<p2<<endl;

    delete p;
    return 0;
}
```

```
Parameterized constructor called
Parameterized constructor called
1 2.2
2.8 Hello
destructor of Pair
destructor of Pair
```

You are required to demonstrate the working of a class MyList that manages a dynamically growing array of cstrings, Partial class definition, driver program (main) and required output is given below. Implement the class MyList such that the given main program runs successfully. Make sure that your program doesn't consume extra space and there shouldn't be any memory leakage or exceptions in your code.

```
#include <iostream>
#include <cstring>
using namespace std;

class MyList {
    char** arr;
    int size;

public:
    MyList();
    ~MyList();

    friend ostream& operator<<(ostream& out, const MyList& m);
    friend void operator+(const char* str, MyList& list);
    MyList operator+(const MyList& rt) const;
};

MyList::MyList() : arr(nullptr), size(0) {}

MyList::~MyList() {
    for (int i = 0; i < size; ++i) {
        delete[] arr[i];
    }
    delete[] arr;
}

MyList MyList::operator+(const MyList& rt) const {
    int newSize = size + rt.size;
    MyList sumList;
    sumList.size = newSize;
    sumList.arr = new char*[newSize];

    for (int i = 0; i < size; ++i) {
        int len = strlen(arr[i]) + 1;
        sumList.arr[i] = new char[len];
        strcpy(sumList.arr[i], arr[i]);
    }
    for (int i = size, j = 0; i < newSize; ++i, ++j) {
        int len = strlen(rt.arr[j]) + 1;
```

```cpp
        sumList.arr[i] = new char[len];
        strcpy(sumList.arr[i], rt.arr[j]);
    }

    return sumList;
}

ostream& operator<<(ostream& out, const MyList& m) {
    out << "[ ";
    for (int i = 0; i < m.size; ++i) {
        out << m.arr[i];
        if (i < m.size - 1) {
            out << ", ";
        }
    }
    out << " ]";
    return out;
}

void operator+(const char* str, MyList& list) {
    char** newArr = new char*[list.size + 1];

    for (int i = 0; i < list.size; ++i) {
        int len = strlen(list.arr[i]) + 1;
        newArr[i] = new char[len];
        strcpy(newArr[i], list.arr[i]);
    }

    int len = strlen(str) + 1;
    newArr[list.size] = new char[len];
    strcpy(newArr[list.size], str);

    for (int i = 0; i < list.size; ++i) {
        delete[] list.arr[i];
    }
    delete[] list.arr;

    list.arr = newArr;
    list.size++;
}

int main() {
    MyList list1;
    "Hello" + list1;
    "World" + list1;
    cout << list1 << endl;

    MyList list2;
    "C++" + list2;
    "Programming" + list2;

    cout << list2 << endl;

    MyList list3 = list1 + list2;
    cout << list3 << endl;

    return 0;
}
```

See output from final.pdf