12.02 Templates Basics

Tuesday, March 28, 2023 2:29 PM

- Templates give capability to parameterize types in fn and classes, can define fn/classes w/ generic types that can be substituted for concrete types by the compiler
- begin w/ a simple ex:
  - Want to find max of 2 ints, 2 doubles, 2 chars, and 2 strings
  - Might wanna write overloaded fn:

```
int maxValue(int value1, int value2)
{
  if (value1 > value2)
    return value1;
  else
    return value2;
}

double maxValue(double value1, double value2)
{
  if (value1 > value2)
    return value1;
  else
    return value2;
}

char maxValue(char value1, char value2)
{
  if (value1 > value2)
    return value1;
  else
    return value2;
}

string maxValue(string value1, string value2)
{
  if (value1 > value2)
    return value1;
  else
    return value2;
}
```

  - 4 fn identical except for the diff types
  - Easy to apply generic thing:

```
GenericType maxValue(GenericType value1, GenericType value2)
{
  if (value1 > value2)
    return value1;
  else
    return value2;
}
```

  - The GenericType applies to all types such as int, double, char, and string
  - C++ lets define fn template w/ generic types

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

template<typename T>
T maxValue(const T value1, const T value2)
{
  if (value1 > value2)
    return value1;
  else
    return value2;
}

int main()
{
  cout << "Maximum between 1 and 3 is " << maxValue(1, 3) << endl;
  cout << "Maximum between 1.5 and 0.3 is "
    << maxValue(1.5, 0.3) << endl;
  cout << "Maximum between 'A' and 'N' is "
    << maxValue('A', 'N') << endl;
  cout << "Maximum between \"NBC\" and \"ABC\" is "
    << maxValue(string("NBC"), string("ABC")) << endl;

  return 0;
}
```

Automatic Check | Compile/Run | Reset | Answer | Choose a Compile

Execution Result:

```
command>cl GenericMaxValue.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>GenericMaxValue
Maximum between 1 and 3 is 3
Maximum between 1.5 and 0.3 is 1.5
Maximum between 'A' and 'N' is N
Maximum between "NBC" and "ABC" is NBC

command>
```

- Def for the fn template begins w/ keyword template followed by list of parameters, each param must be preceded by the interchangeable keyword typename or class in form <typename typeParameter> or <class typeParameter>
  - `template<typename T>`
  - This begins def of the fn template for maxValue, this line aka template prefix, where T is a type param (convention to use single capital letter to denote a type param)
-

Caution

The generic `maxValue` function can be used to return a maximum of two values of *any type*, provided that

- The two values have the same type;
- The two values can be compared using the > operator.

For example, if one value is `int` and the other is `double` (e.g., `maxValue(1, 3.5)`), the compiler will report a syntax error because it cannot find a match for the call. If you invoke `maxValue(Circle(1), Circle(2))`, the compiler will report a syntax error because the > operator is not defined in the `Circle` class.

- **Tip**

  You can use either `<typename T>` or `<class T>` to specify a type parameter. Using `<typename T>` is better because `<typename T>` is descriptive. `<class T>` could be confused with class definition.

- **Note**

  Occasionally, a template function may have more than one parameter. In this case, place the parameters together inside the brackets, separated by commas, such as `<typename T1, typename T2, typename T3>`.

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

  template<typename T>
  T maxValue(const T& value1, const T& value2)
  {
    if (value1 > value2)
      return value1;
    else
      return value2;
  }

  int main()
  {
    cout << "Maximum between 1 and 3 is " << maxValue(1, 3) << endl;
    cout << "Maximum between 1.5 and 0.3 is "
      << maxValue(1.5, 0.3) << endl;
    cout << "Maximum between 'A' and 'N' is "
      << maxValue('A', 'N') << endl;
    cout << "Maximum between \"NBC\" and \"ABC\" is "
      << maxValue(string("NBC"), string("ABC")) << endl;

    return 0;
  }
  ```

  Automatic Check | Compile/Run | Reset | Answer | Choose a Compi

  Execution Result:

  ```
  command>cl GenericMaxValuePassByReference.cpp
  Microsoft C++ Compiler 2019
  Compiled successful (cl is the VC++ compile/link command)

  command>GenericMaxValuePassByReference
  Maximum between 1 and 3 is 3
  Maximum between 1.5 and 0.3 is 1.5
  Maximum between 'A' and 'N' is N
  Maximum between "NBC" and "ABC" is NBC

  command>
  ```
  ```

Which of the following statements is incorrect?

- Templates provide the capability to parameterize types in functions and classes.
- With templates, you can define a function or a class with a generic type that can be substituted for a concrete type by the compiler.
- Templates facilitate developing reusable software.
- ✓ Templates improve performance.

Well done!

Templates do not improve performance.

Which of the statements is incorrect to invoke the following template function?

```
template<typename T>
T maxValue(const T& value1, const T& value2)
{
  if (value1 > value2)
    return value1;
  else
    return value2;
}
```

- cout << maxValue(1, 2)
- cout << maxValue(1.5, 2.5)
- cout << maxValue('A', 'B')
- cout << maxValue("AB", "AB")
- ✓ cout << maxValue(1.5, 2)

Well done!

This is incorrect because the two values are not of the same type.

You can invoke the following swap function using _____.

```
template<typename T>
void swap(T& var1, T& var2)
{
  T temp = var1;
  var1 = var2;
  var2 = temp;
}
```

- swap(1, 2)
- ✓ int v1 = 1; int v2 = 2; swap(v1, v2);
- int v1 = 1; int v2 = 2; swap(&v1, &v2);
- int v1 = 1; double v2 = 2; swap(v1, v2);

Good job!

See LiveExample 12.2.

12.3 Ex: A Generic Sort

Saturday, April 1, 2023 11:49 PM

- This section designs a generic sort fn
-
```
void selectionSort(double list[], int listSize)
{
  for (int i = 0; i < listSize; i++)
  {
    // Find the minimum in the list[i..listSize-1]
    double currentMin = list[i];
    int currentMinIndex = i;

    for (int j = i + 1; j < listSize; j++)
    {
      if (currentMin > list[j])
      {
        currentMin = list[j];
        currentMinIndex = j;
      }
    }

    // Swap list[i] with list[currentMinIndex] if necessary
    if (currentMinIndex != i)
    {
      list[currentMinIndex] = list[i];
      list[i] = currentMin;
    }
  }
}
```
- Ez to mod this fn to write new overloaded fn for sorting an array of int values, chars, strings, etc, just have to replace double w/ whatever u want
- Instead of write many overloaded sort fn, just write 1 template fn that works for any type
-
```
#include <iostream>
#include <string>
using namespace std;

template<typename T>
void sort(T list[], int listSize)
{
  for (int i = 0; i < listSize; i++)
  {
    // Find the minimum in the list[i..listSize-1]
    T currentMin = list[i];
    int currentMinIndex = i;

    for (int j = i + 1; j < listSize; j++)
    {
      if (currentMin > list[j])
      {
        currentMin = list[j];
        currentMinIndex = j;
      }
    }

    // Swap list[i] with list[currentMinIndex] if necessary;
    if (currentMinIndex != i)
    {
      list[currentMinIndex] = list[i];
      list[i] = currentMin;
    }
  }
}

template<typename T>
void printArray(const T list[], int listSize)
{
  for (int i = 0; i < listSize; i++)
```

```
}

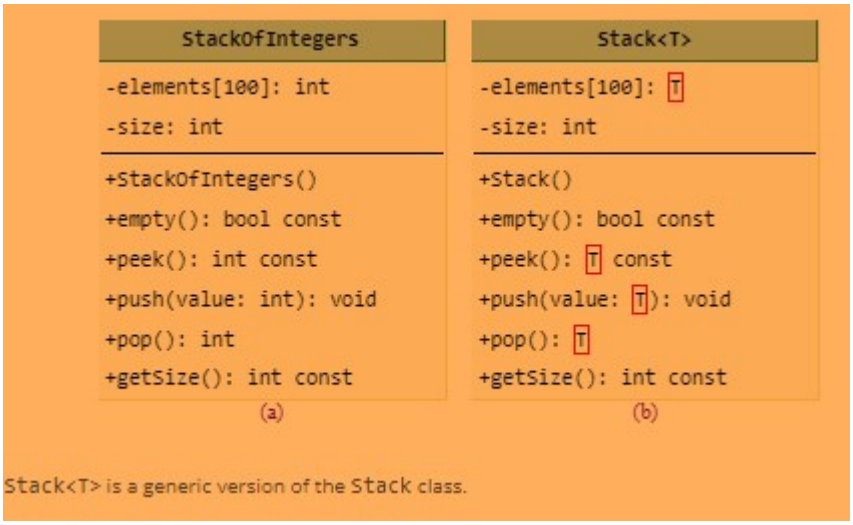
template<typename T>
void printArray(const T list[], int listSize)
{
  for (int i = 0; i < listSize; i++)
  {
    cout << list[i] << " ";
  }
  cout << endl;
}

int main()
{
  int list1[] = {3, 5, 1, 0, 2, 8, 7};
  sort(list1, 7);
  printArray(list1, 7);

  double list2[] = {3.5, 0.5, 1.4, 0.4, 2.5, 1.8, 4.7};
  sort(list2, 7);
  printArray(list2, 7);

  string list3[] = {"Atlanta", "Denver", "Chicago", "Dallas"};
  sort(list3, 4);
  printArray(list3, 4);
  return 0;
}
```

-

```
command>cl GenericSort.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>GenericSort
0 1 2 3 5 7 8
0.4 0.5 1.4 1.8 2.5 3.5 4.7
Atlanta Chicago Dallas Denver

command>
```

- This pgrm has 2 template fn
-

> **Tip**
>
> When you define a generic function, it is better to start with a nongeneric function, debug and test it, and then convert it to a generic function.

-

Given the printArray template function, which of the statements are correct to invoke it?

```
template<typename T>
void printArray(T list[], int arraySize)
{
  for (int i = 0; i < arraySize; i++)
  {
    cout << list[i] << " ";
  }
  cout << endl;
}
```

- int list[] = {1, 2, 3, 4}; printArray(list, 4);
- int list[] = {1, 2.5, 3, 4}; printArray(list, 4);
- double list[] = {1, 2, 3, 4}; printArray(list, 4);
- string list[] = {"Atlanta", "Dallas", "Houston", "Chicago"}; printArray(list, 4);
- ✓ All of the above.

Well done!

See LiveExample 12.3.

12.4 Class Templates

Sunday, April 2, 2023 5:42 PM

- U can define generic types for a class
- Can define template classes w/ type parameters for the class
- Type parameters can be used everywhere in the class where a reg type appears
-

```
#ifndef STACK_H
  #define STACK_H
  class StackOfIntegers{
  public:
    StackOfIntegers();
    bool empty() const;
    int peek() const;
   void push(int value);
    int pop();
    int getSize() const;
 private:
   int elements[100];
   int size;
 };
 StackOfIntegers::StackOfIntegers(){
   size = 0;
 }
 bool StackOfIntegers::empty() const{
   return size == 0;
 }
 int StackOfIntegers::peek() const{
   return elements[size - 1];
 }
 void StackOfIntegers::push(int value){
   elements[size++] = value;
 }
 int StackOfIntegers::pop(){
   return elements[--size];
 }
 int StackOfIntegers::getSize() const{
   return size;
 }
 #endif
```

-
-



- Can make this for not only int by using template class for any type of element

```
#ifndef STACK_H
#define STACK_H

FILL_CODE_OR_CLICK_ANSWER<FILL_CODE_OR_CLICK_ANSWER>
class Stack
{
public:
  Stack();
  bool empty() FILL_CODE_OR_CLICK_ANSWER
  T peek() const;
  void push(T value);
  T pop();
  int getSize() const;

private:
  T elements[100];
  int size;
};

FILL_CODE_OR_CLICK_ANSWER
FILL_CODE_OR_CLICK_ANSWER::Stack()
{
  size = 0;
}

template<typename T>
bool Stack<T>::empty() const
{
  return size == 0;
}

template<typename T>
T Stack<T>::peek() const
{
  return elements[size - 1];
}

template<typename T> // Implement push function
void FILL_CODE_OR_CLICK_ANSWER
{
  elements[size++] = value;
}

template<typename T>
T Stack<T>::pop()
{
  return elements[--size];
}

template<typename T>
int Stack<T>::getSize() const
{
  return size;
}

#endif
```

- Syntax is similar:
- `template<typename T>`
- Type param can be used in the class like any reg data type
- Constructors and fn are defined the same way for reg classes, but the constructors and fn themselves are templates, so have to put template prefix b4 the constructor and fn header in the implementation:

```
template<typename T>
Stack<T>::Stack()
{
  size = 0;
}

template<typename T>
bool Stack<T>::empty()
{
  return size == 0;
}

template<typename T>
T Stack<T>::peek()
{
  return elements[size - 1];
}
```

- Note that class name b4 the scope resolution operator :: is Stack<T>, not Stack
-

- **Tip**

  GenericStack.h combines class definition and class implementation into one file. Normally, you put class definition and class implementation into two separate files. For class templates, however, it is safer to put them together, because some compliers cannot compile them separately.

-
```
#include <iostream>
#include <string>
#include "GenericStack.h"
using namespace std;

int main()
{
  // Create a stack of int values
  Stack<int> intStack;
  for (int i = 0; i < 10; i++)
    intStack.push(i); // Push i into the stack

  while (!intStack.empty())
    cout << intStack.pop() << " ";
  cout << endl;

  // Create a stack of strings
  Stack<string> stringStack;
  stringStack.push("Chicago");
  stringStack.push("Denver");
  stringStack.push("London"); // Push London to the stack

  while (!stringStack.empty())
    cout << stringStack.pop() << " ";
  cout << endl;

  return 0;
}
```

Automatic Check | Compile/Run | Reset | Answer

Execution Result:

```
command>cl TestGenericStack.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>TestGenericStack
9 8 7 6 5 4 3 2 1 0
London Denver Chicago

command>
```

- To declare an object from a template class, have to specify a concrete type for the type param T, like
-
```
Stack<int> intStack;
```
- This declaration replaces the type parameter T w/ int, so intStack is a stack for int vals, object intStack is like any other object, the pgrm invokes the push fn on intStack to add ten int vals to the stack and displays the elements from the stack
-

- Note the code in lines 9–11:

```
while (!intStack.empty())
  cout << intStack.pop() << " ";
cout << endl;
```

  and in lines 23–25:

```
while (!stringStack.empty())
  cout << stringStack.pop() << " ";
cout << endl;
```

- These code fragments almost identical, diff is that 1st operates on intStack, 2nd operates on StringStack, can define a fn w/ a stack param to display the elements in the stack

```
1  #include <iostream>
2  #include <string>
3  #include "GenericStack.h"
4  using namespace std;
5
6  template<typename T>
7  void printStack(Stack<T>& stack)
8  {
9    while (!stack.empty())
10     cout << stack.pop() << " ";
11   cout << endl;
12 }
13
14 int main()
15 {
16   // Create a stack of int values
17   Stack<int> intStack;
18   for (int i = 0; i < 10; i++)
19     intStack.push(i);
20   printStack(intStack);
21
22   // Create a stack of strings
23   Stack<string> stringStack;
24   stringStack.push("Chicago");
25   stringStack.push("Denver");
26   stringStack.push("London");
27   printStack(stringStack);
28
29   return 0;
30 }
```

Automatic Check | Compile/Run | Reset | Answer

Execution Result:

```
command>cl TestGenericStackWithTemplateFunction.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>TestGenericStackWithTemplateFunction
9 8 7 6 5 4 3 2 1 0
London Denver Chicago

command>
```

- The generic class name Stack<T> is used as a param type in template fn (ln 7)

- **Note**

  C++ allows you to assign a *default type* for a type parameter in a class template. For example, you may assign `int` as a default type in the generic `Stack` class as follows:

  ```
  template<typename T = int>
  class Stack
  {
    ...
  };
  ```

  You now can declare an object using the default type like this:

  ```
  Stack<> stack;  // stack is a stack for int values
  ```

  You can use default type only in class templates, not in function templates.

- **Note**

  You also can use *nontype parameters* along with type parameters in a template prefix. For example, you may declare the array capacity as a parameter for the `Stack` class as follows:

  ```
  template<typename T, int capacity>
  class Stack
  {
    ...
  private:
    T elements[capacity];
    int size;
  };
  ```

  So, when you create a stack, you can specify the capacity for the array. For example,

  ```
  Stack<string, 500> stack;
  ```

  declares a stack that can hold up to `500` strings.

- **Note**

  You can define static members in a template class. Each template specialization has its own copy of a static data field.

- Which of the statements is incorrect for using the following template class?

```
template<typename T = int>
class Stack
{
  Stack();
  ...
};
```

  - `Stack<double> s;`
  - `Stack<int> s;`
  - `Stack<> s;`
  - ✓ `Stack s;`

  Fantastic!

  This statement is incorrect.

- Which of the statements is incorrect for using the following template class?

```
template<typename T, int capacity>
class Stack
{
  Stack();
  ...
private:
  T elements[capacity];
  int size;
};
```

  - `Stack<double, 40> s;`
  - `Stack<int, 50> s;`
  - `Stack<string, 50> s;`
  - ✓ `Stack<int, double> s;`

  Excellent!

  This statement is incorrect.

-
-

12.5 Improving the Stack Class

Sunday, April 2, 2023 7:04 PM

- This section implements a dynamic stack class
- Elements in Stack class are stored in array w/ fixed size (100), so cant add more than 100 elements
- Quick fix: change the number
  - But this waste space if only small stack
- Fix: add space dynamically when needed
- The size property in Stack<T> class reps the numb of elements in the stack
  - Add new property, capacity, reps the current size of the array for storing the elements
  - The no-arg constructor of Stack<T> makes an array w/ capacity 16
  - When u add a new element to the stack, may need to increase the array size in order to store the new element if the current capacity is full
  - Can't change size of array after its declared, but can make new bigger array, copy the elements from old to new, then delete the old array

```
#ifndef IMPROVEDSTACK_H
#define IMPROVEDSTACK_H

template<typename T>
class Stack
{
public:
  Stack(); // No-arg constructor
  Stack(const Stack&); // Copy constructor
  ~Stack(); // Destructor
  bool empty() const;
  T peek() const;
  void push(T value);
  T pop();
  int getSize() const;

private:
  T* elements;
  int size;
  int capacity;
  void ensureCapacity();
};

template<typename T>
Stack<T>::Stack(): size(0), capacity(16)
{
  elements = new T[capacity];
}

template<typename T>
Stack<T>::Stack(const Stack& stack)
{
  elements = new T[stack.capacity];
  size = stack.size;
```

```
  capacity = stack.capacity;
  for (int i = 0; i < size; i++)
  {
    elements[i] = stack.elements[i];
  }
}

template<typename T>
Stack<T>::~Stack()
{
  delete [] elements;
}

template<typename T>
bool Stack<T>::empty() const
{
  return size == 0;
}

template<typename T>
T Stack<T>::peek() const
{
  return elements[size - 1];
}

template<typename T>
void Stack<T>::push(T value)
{
  ensureCapacity();
  elements[size++] = value;
}

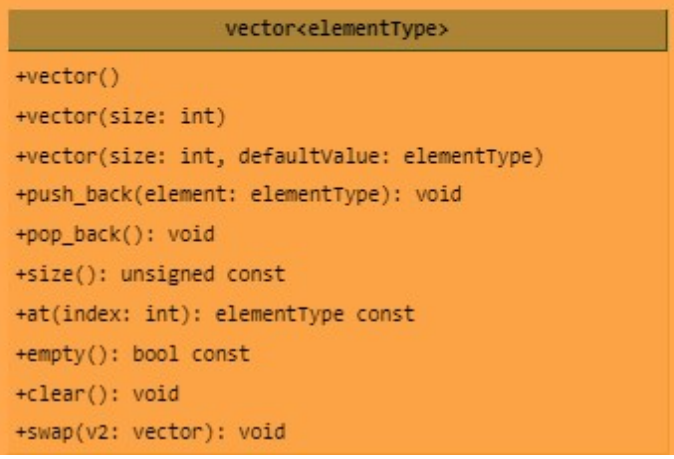
template<typename T>
void Stack<T>::ensureCapacity()
{
  if (size >= capacity)
  {
    T* old = elements;
    capacity = 2 * size;
    elements = new T[size * 2];

    for (int i = 0; i < size; i++)
      elements[i] = old[i];

    delete [] old;
  }
}

template<typename T>
T Stack<T>::pop()
{
  return elements[--size];
}

template<typename T>
int Stack<T>::getSize() const
{
  return size;
}

#endif
```

- Since internal array elements is dynamically made, a destructor must be given to properly destroy

the array to avoid mem leak (ln 42-46)

- Please note that the syntax to destroy a dynamically created array is
- ```
  delete [] elements; // Line 45
  delete [] old; // Line 79
  ```
- What happens if you mistakenly write the following?
- ```
  delete elements; // Line 45
  delete old; // Line 79
  ```
  - Pgrm will compile and run fine for a stack of primitive-type vals, but not correct for a stack of objects
  - The delete [] elements first calls the destructor on each object in the elements array and then destroys the array, but the statement delete element calls the destructor only on the first object in the array
- The Stack class can be further improved by storing the elements in a vector (next section)
- In the implementation of ImprovedStack.h, which of the following is true?

  size may increase or decrease.

  capacity never reduces.

  Inside Stack, a regular array is used to store elements.

  If the current capacity equals to size, capacity is doubled when a new element is added to Stack.

  ✓ All of the above.

  Excellent!

12.6 The C++ vector Class

Sunday, April 2, 2023 7:24 PM

- C++ has a generic vector class for storing a list of objects
- Can use an array to store a collection of data like strings and int vals
- C++ gives the vector class, which is more flexible than arrays, can be used like an array, but size auto grow as needed
- To make, use syntax
- ```
  vector<elementType> vectorName;
  ```
  - Ex:
  - ```
    vector<int> intVector;
    ```
  - Makes a vector to store int vals
  - ```
    vector<string> stringVector;
    ```
  - Which make a vector to store string objects
- **Figure 12.2** lists several frequently used functions in the vector class in a UML class diagram.

  Figure 12.2

  

- Can also make a vector w/ initial size, filled w/ default vals
  - Ex:
  - ```
    vector<int> intVector(10);
    ```
  - Makes a vector of initial size 10 w/ default vals 0
  - Vector can be accessed using the subscript operator [], like
  - ```
    cout << intVector[0];
    ```
  - Which displays the first element in the vector
-
- **Caution**

  To use the array subscript operator [ ], the element must already exist in the vector. Like array, the index is 0-based in a vector–i.e., the index of the first element in the vector is 0 and the last one is `v.size() - 1`. To use an index beyond this range would cause errors.
    ```

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

int main()
{
  vector<int> intVector; // Create a vector named intVector

  // Store numbers 1, 2, 3, 4, 5, ..., 10 to the vector
  for (int i = 0; i < 10; i++)
    intVector.push_back(i + 1);

  // Display the numbers in the vector
  cout << "Numbers in the vector: ";
  for (int i = 0; i < intVector.size(); i++)
    cout << intVector[i] << " ";

  vector<string> stringVector;

  // Store strings into the vector
  stringVector.push_back("Dallas");
  stringVector.push_back("Houston");
  stringVector.push_back("Austin");
  stringVector.push_back("Norman"); // Add Norman to the vector

  // Display the string in the vector
  cout << "\nStrings in the string vector: ";
  for (int i = 0; i < stringVector.size(); i++)
    cout << stringVector[i] << " ";

  stringVector.pop_back(); // Remove the last element

  vector<string> v2;
  v2.swap(stringVector);
  v2[0] = "Atlanta"; // Assign Atlanta to replace the first element in v2

  // Redisplay the string in the vector
  cout << "\nStrings in the vector v2: ";
  for (int i = 0; i < v2.size(); i++)
    cout << v2.at(i) << " ";

  return 0;
}
```

```
command>cl TestVector.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>TestVector
Numbers in the vector: 1 2 3 4 5 6 7 8 9 10
Strings in the string vector: Dallas Houston Austin Norman
Strings in the vector v2: Atlanta Houston Austin

command>
```

- Gotta keep the header class #include <vector>
- Also, string class is used, so gotta use #include <string>

The `size()` function returns the size of the vector as an `unsigned` (i.e., unsigned integer), not `int`. Some compilers may warn you because an unsigned value is used with a signed `int` value in variable `i` (lines 16, 29, 40). This is just a warning and should not cause any problems, because the unsigned value is automatically promoted to a signed value in this case. To get rid of the warning, declare `i` to be `unsigned int` in line 16 as follows:

```cpp
for (unsigned i = 0; i < intVector.size(); i++)
```

- In C++11, u can assign vals to a vector using a vector initializer, which like an array initializer
  - Ex:
  - ```cpp
    vector<int> intVector{1, 9};
    ```
    ```

- Which makes a vector w/ initial vals 1 and 9

- Can also assign vals to a vector after the vector made using syntax:
- ```
  intVector = {13, 92, 1};
  ```
- After executing this statement, intVector now has those vals, the old vals in intVector are destroyed
- Cant assign vals to an array after array is made using an array initializer
  - Ex:
  - ```
    int temp[] = {1, 9};
    temp = {13, 92, 1};
    ```
  - This causes syntax error
- Like array, can also use foreach loop to traverse all elements in a vector in C++, like
- ```
  for (int e: intVector)
  {
    cout << e << endl;
  }
  ```
- This displays all the vals in intVector
-
- Declare a `vector` named `scores` of twenty-five elements of type `int`.

  ```
  1  vector<int> scores(25);
  ```
-
- Given a vector a, declared to contain 34 elements, write an expression that refers to the last element of the vector.

  ```
  1  a.at(33)
  ```

  Excellent!

  Remarks and hints

  - Nice One!
-
  ```

- Given that a vector of `int` named a has been declared, assign 3 to its first element.

```
1  a[0] = 3;
```

Excellent!

Remarks and hints

- Nice One!

- Assume that the vector `arr` has been declared. In addition, assume that `VECTOR_SIZE` has been defined to be an `int` that equals the number of elements in `arr`.

Write a statement that assigns the next to last element of the vector to the variable `x` (`x` has already been declared).

```
1  x=arr[VECTOR_SIZE - 2];
```

Fantastic!

Remarks and hints

- Nice One!

- Given that a vector of `int` named a has been declared with 12 elements and that the `int` variable `k` holds a value between 0 and 6.

  Assign 9 to the element just after a[k].

  ```
  1  a[k+1] = 9;
  ```

  Nice work!

  Remarks and hints

  - Nice One!

- To declare a vector for holding int values, use _____.

  - ✓ `vector<int> v;`

- To add an int value 5 to a vector v of integers, use _____.

  - `v.add(5);`
  - `v.insert(5);`
  - ✓ `v.push_back(5);`
  - `v.append(5);`

- 
To obtain the size of the vector v, use _____.
v.getSize();
v.length();
v.getLength();
v.size();

12.7 Insertion and Deletion and Other Functions for a Vector

Sunday, April 2, 2023 9:17 PM

- U can use the insert and erase function to insert and delete elements in a vector
- Can insert an element at end of a vector using push_back fn, but to insert, need to use insert(p,element) where p is pointer to element in the vector
- Pointer to first element in a vector can be got by v.begin(), so pointer to ith element is v.begin() + I
- If want last element in vector, can use v.end(), so to point to last element in the vector is v.end()-1
- Can remove last element in a vector using pop_back fn, to remove element from anywhere in vector, use erase(p) fn
- Can use fns min_element, max_element, sort, random_shuffle, and find
-

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

template<typename T>
void print(const string& title, const vector<T>& v)
{
  cout << title << " ";
  for (int i = 0; i < v.size(); i++)
    cout << v[i] << " ";
  cout << endl;
}

int main()
{
  vector<int> v;
  for (int i = 0; i < 5; i++)
    v.push_back(i);

  v.insert(v.begin() + 1, 20); // Insert 20 at index 1
  v.erase(v.end() - 2); // Remove the second last element
  print("The elements in vector:", v);

  sort(v.begin(), v.end()); // Sort the elements in v
  print("Sorted elements:", v);

  random_shuffle(v.begin(), v.end()); // Shuffle the elements in v
  print("After random shuffle:", v);

  cout << "The max element is " <<
    *max_element(v.begin(), v.end()) << endl;

  cout << "The min element is " <<
    *min_element(v.begin(), v.end()) << endl;

  int key = 45;
  if (find(v.begin(), v.end(), key) == v.end())
    cout << key << " is not in the vector" << endl;
  else
    cout << key << " is in the vector" << endl;

  return 0;
}
```

-

```
command>cl VectorInsertDelete.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>VectorInsertDelete
The elements in vector: 0 20 1 2 4
Sorted elements: 0 1 2 4 20
After random shuffle: 20 1 4 2 0
The max element is 20
The min element is 0
45 is not in the vector

command>
```

- This pgrm makes a vector
-

- Assume v is a vector that has been declared and initialized.

  Write an expression whose value is the number of values that have been stored in v.

```
1  v.size();
```

- Assume v is a vector that has been declared and initialized. Write an expression whose value is `true` if there are any values stored in v.

```
1  v.size();
```

  Well done!

  Remarks and hints

  - Nice One!

- Assume v is a vector that has been declared and initialized.

  Write a statement that removes all the values stored in v.

```
1  v.clear();
2
```

  Fantastic!

  Remarks and hints

  - Nice One!

- Assume v is a vector of `int` that has been declared and initialized

  Write a statement that adds the value 42 to the vector.

  ```
  1  v.push_back(42);
  ```

  Well done!

  Remarks and hints

  - Nice One!

- Which of the following statements inserts number 5 to the beginning of vector v?

  - v.add(5)
  - v.add(0, 5)
  - v.insert(0, 5)
  - ✓ v.insert(v.begin(), 5)

  Nice work!

  See LiveExample 12.9.

- Which of the following statements deletes the first element from vector v?

  - v.delete(0)
  - v.erase(0)
  - ✓ v.erase(v.begin())
  - v.delete(v.begin())

  Nice work!

  See LiveExample 12.9.

- Which of the following statements returns the smallest element in vector v?

  ✗ min_element(v)

  min_element(v, v.begin(), v.end())

  ✓ *min_element(v, v.begin(), v.end())

  min_element(v, v.begin(), v.end() - 1)

  That's incorrect.

  See LiveExample 12.9.

12.8 Replacing Arrays Using the vector Class

Sunday, April 2, 2023 10:39 PM

- Vector can be used to replace arrays, they more flexible than arrays, but arrays are more efficient than vectors
-

Operation	Array	vector
Creating an array/vector	`string a[10]`	`vector<string> v`
Initializing an array/vector	`int a[] = {1, 2}`	`vector<int> v{1, 2}`
Assigning new values		`v = {12, 23}`
Accessing an element	`a[index]`	`v[index]`
Updating an element	`a[index] = "London"`	`v[index] = "London"`
Returning size		`v.size()`
Appending a new element		`v.push_back("London")`
Inserting e to ith position		`v.insert(v.begin() + i, e)`
Removing last element		`v.pop_back()`
Removing ith element		`v.erase(v.begin() + i)`
Removing all elements		`v.clear()`

- Both can be used to store list of elements
- Array more efficient if size of list is fixed, vector is resizable array
- The vector class has many member fn for accessing and manipulating a vector
- Using vectors is more flexible than using arrays, can always use vectors to replace arrays
  - Ex:
  - Deck of cards from b4

```
const int NUMBER_OF_CARDS = 52;
vector<int> deck(NUMBER_OF_CARDS);

// Initialize cards
for (int i = 0; i < NUMBER_OF_CARDS; i++)
  deck[i] = i;
```

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

const int NUMBER_OF_CARDS = 52;
string suits[4] = {"Spades", "Hearts", "Diamonds", "Clubs"};
string ranks[13] = {"Ace", "2", "3", "4", "5", "6", "7", "8", "9",
  "10", "Jack", "Queen", "King"};

int main()
{
  vector<int> deck(NUMBER_OF_CARDS);

  // Initialize cards
  for (int i = 0; i < NUMBER_OF_CARDS; i++)
    deck[i] = i;

  // Shuffle the cards
  srand(time(0));
  random_shuffle(deck.begin(), deck.end()); // Shuffle the cards

  // Display the first four cards
  for (int i = 0; i < 4; i++)
  {
    cout << ranks[deck[i] % 13] << " of " <<
      suits[deck[i] / 13] << endl;
  }

  return 0;
}
```

Compile/Run Reset Answer Choose a Compil

Execution Result:

```
command>cl DeckOfCardsUsingVector.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>DeckOfCardsUsingVector
6 of Spades
5 of Clubs
9 of Spades
4 of Hearts

command>
```

Recall that **LiveExample 8.1** creates a two-dimensional array and invokes a function to return the sum of all elements in the array. A vector of vectors can be used to represent a two-dimensional array. Here is an example to represent a two-dimensional array with four rows and three columns:

```cpp
vector<vector<int>> matrix(4); // four rows
for (int i = 0; i < 4; i++)
  matrix[i] = vector<int>(3);
matrix[0][0] = 1; matrix[0][1] = 2; matrix[0][2] = 3;
matrix[1][0] = 4; matrix[1][1] = 5; matrix[1][2] = 6;
matrix[2][0] = 7; matrix[2][1] = 8; matrix[2][2] = 9;
matrix[3][0] = 10; matrix[3][1] = 11; matrix[3][2] = 12;
```

The preceding code can be simplified using the following code:

```cpp
vector<vector<int>> matrix{{1, 2, 3},
  {4, 5, 6}, {7, 8, 9}, {10, 11, 12}};
```

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

int sum(const vector<vector<int>>& matrix)
{
  int total = 0;
  for (unsigned int row = 0; row < matrix.size(); row++)
  {
    for (unsigned column = 0; column < matrix[row].size(); column++)
    {
      total += matrix[row][column];
    }
  }

  return total;
}

int main()
{
  vector<vector<int>> matrix{
    {1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12}};

  cout << "Sum of all elements is " << sum(matrix) << endl;

  return 0;
}
```

Automatic Check | Compile/Run | Reset | Answer | Choose a Compiler:

Execution Result:

```
command>cl TwoDArrayUsingVector.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>TwoDArrayUsingVector
Sum of all elements is 78

command>
```

The `sum` function returns the sum of all elements in the vector. The size of the vector can be obtained from the `size()` function in the `vector` class. So, you don't have to specify the vector's size when invoking the `sum` function. The same function for two-dimensional array requires two parameters as follows:

```cpp
int sum(const int a[][COLUMN_SIZE], int rowSize)
```

Using vectors for representing two-dimensional arrays simplifies coding.

- Which of the following statements is correct?

  - vector<int> v = { 1, 2 };
  - int a[] = {1, 2};
  - v[0] = 3;
  - a[0] = 4;
  - ✓ All of the above.

  Good job!

  See Table 12.1.

12.9 Case Study: Evaluating Expressions

Sunday, April 2, 2023 10:55 PM

- Stacks can be used to evaluate expressions
-


- Prob solved using 2 stacks, operandStack and operatorStack
- 2 phases:
  - Phase 1: Scanning Expression
  - Extract operands and put them in the operandStack, process ones by higher precedence
  - Phase 2: Clearing Stack
  - Repeatedly process the operator from the top of the operatorsStak unitl its empty
  -

```
#include <iostream>
#include <vector>
#include <string>
#include <cctype>
#include "ImprovedStack.h"

using namespace std;

// Split an expression into numbers, operators, and parenthese
FILL_CODE_OR_CLICK_ANSWER split(const string& expression);

// Evaluate an expression and return the result
int evaluateExpression(const string& expression);

// Perform an operation
void processAnOperator(
  Stack<int>& operandStack, Stack<char>& operatorStack);

int main()
{
  string expression;
  cout << "Enter an expression: ";
  getline(cin, expression);

  cout << expression << " = "
    << evaluateExpression(expression) << endl;
```

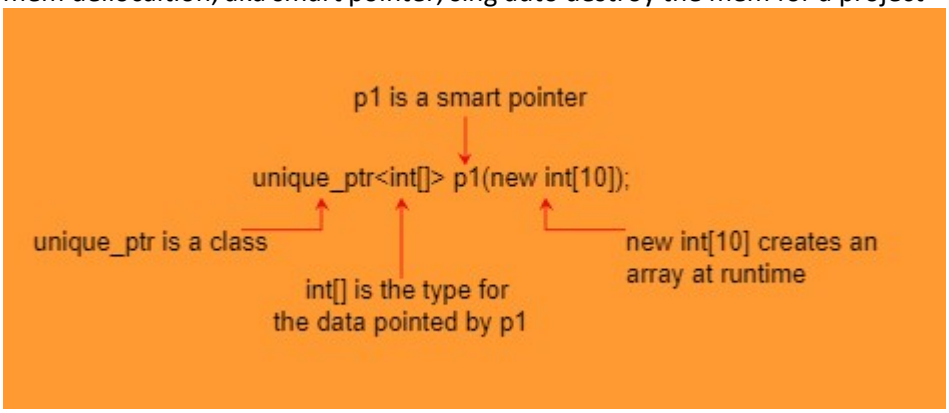
```
  getline(cin, expression);

  cout << expression << " = "
    << evaluateExpression(expression) << endl;

  return 0;
}

vector<string> split(const string& expression)
{
  vector<string> v; // A vector to store split items as strings
  string numberString; // A numeric string

  for (unsigned int i = 0; i < expression.length(); i++)
  {
    if (isdigit(expression[i]))
      numberString.append(1, expression[i]); // Append a digit
    else
    {
      if (numberString.size() > 0)
      {
        v.push_back(numberString); // Store the numeric string
        numberString.erase(); // Empty the numeric string
      }

      if (!isspace(expression[i]))
      {
        string s;
```

```
        s.append(1, expression[i]);
        v.push_back(s); // Store an operator and parenthesis
      }
    }
  }

  // Store the last numeric string
  if (numberString.size() > 0)
    v.push_back(numberString);

  return v;
}

// Evaluate an expression
int evaluateExpression(const string& expression)
{
  // Create operandStack to store operands
  Stack<int> operandStack;

  // Create operatorStack to store operators
  Stack<char> operatorStack;

  // Extract operands and operators
  vector<string> tokens = split(expression);

  // Phase 1: Scan tokens
  for (unsigned int i = 0; i < tokens.size(); i++)
  {
    if (tokens[i][0] == '+' || tokens[i][0] == '-')
    {
      // Process all +, -, *, / in the top of the operator stack
      while (!operatorStack.empty() && (operatorStack.peek() == '+'
       || operatorStack.peek() == '-' || operatorStack.peek() == '*'
       || operatorStack.peek() == '/'))
      {
        processAnOperator(operandStack, operatorStack);
      }

      // Push the + or - operator into the operator stack
      operatorStack.push(tokens[i][0]);
    }
    else if (tokens[i][0] == '*' || tokens[i][0] == '/')
    {
      // Process all *, / in the top of the operator stack
      while (!operatorStack.empty() && (operatorStack.peek() == '*'
        || operatorStack.peek() == '/'))
      {
        processAnOperator(operandStack, operatorStack);
      }

      // Push the * or / operator into the operator stack
```

```
      // Push the * or / operator into the operator stack
      operatorStack.push(tokens[i][0]);
    }
    else if (tokens[i][0] == '(')
    {
      operatorStack.push('('); // Push '(' to stack
    }
    else if (tokens[i][0] == ')')
    {
      // Process all the operators in the stack until seeing '('
      while (operatorStack.peek() != '(')
      {
        processAnOperator(operandStack, operatorStack);
      }

      operatorStack.pop(); // Pop the '(' symbol from the stack
    }
    else
    { // An operand scanned. Push an operand to the stack as integer
      operandStack.push(atoi(tokens[i].c_str()));
    }
  }

  // Phase 2: process all the remaining operators in the stack
  while (!operatorStack.empty())
  {
    processAnOperator(operandStack, operatorStack);
  }

  // Return the result
  return operandStack.pop();
}

// Process one opeator: Take an operator from operatorStack and
// apply it on the operands in the operandStack
void processAnOperator(
    Stack<int>& operandStack, Stack<char>& operatorStack)
{
  char op = operatorStack.pop();
  int op1 = operandStack.pop();
  int op2 = operandStack.pop();
  if (op == '+')
    operandStack.push(op2 + op1);
  else if (op == '-')
    operandStack.push(op2 - op1);
  else if (op == '*')
    operandStack.push(op2 * op1);
  else if (op == '/')
    operandStack.push(op2 / op1);
}
```

12.10 Using Smart Pointers for Auto Object Detection

Sunday, April 2, 2023 11:04 PM

- C++11 gives unique_ptr class from wrapping a pointer to perform auto object destruction
- Prblm w/ dynamic mem allocation is potential mem leak
- Can be caused by a pgrmer error, runtime exception if pgrm throws exception
- C++11 intro new templates called unique_ptr, functions as pointer w/ more features for auto mem dellocaition, aka smart pointer, sing auto destroy the mem for a project
- 

- This statement creates a smart pointer named p1 that points to a memory for an int array of 10 elements. Here are some detailed descriptions for the statement:
  - The unique_ptr class is defined in the memory header file, which must be included in order to use unique_ptr in the program.
  - int[] indicates that the smart pointer points to an array of int values.
  - p1 is the name of the smart pointer.
  - new int[10] is an expression that creates a dynamic memory for an array of 10 int values, which is passed to the constructor of the unique_ptr class to create smart pointer.
-

- Here are several more examples of creating smart pointers:

```
unique_ptr<double> p2(new double);
```

  This statement creates a smart pointer for a `double` value.

```
unique_ptr<Circle> p3(new Circle);
```

  This statement creates a smart pointer for a `Circle` objet.

- Smart pointers like normal pointers

  Smart pointers can be used just like a regular pointer. The following statemer assigns `5.5` to memory pointed by `p2`.

```
*p2 = 5.5;
```

  The following statement displays the area for a circle pointed by `p3`.

```
cout << p3->getArea(); // or cout << (*p3).getArea()
```

  LiveExample 12.13 gives an example that creates a new array which is the reversal of an original array.

  LiveExample 12.13 ReverseArrayUsingSmartPointer.cpp

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

unique_ptr<int[]> reverse(const int* list, int size)
{
  // Smart pointer for int[size]
  unique_ptr<int[]> result(new int[size]);

  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    result[j] = list[i];
  }

  return result;
}

void printArray(const unique_ptr<int[]>& list, int size)
{
  for (int i = 0; i < size; i++)
    cout << list[i] << " ";
}

int main()
{
  int list[] = {1, 2, 3, 4, 5, 6};
  unique_ptr<int[]> p = reverse(list, 6);

  printArray(p, 6);

  return 0;
}
```

Execution Result:

```
command>cl ReverseArrayUsingSmartPointer.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>ReverseArrayUsingSmartPointer
6 5 4 3 2 1

command>
```

The smart pointer p is passed by reference to `list` in the `printArray` function (line 29). The `printArray` function displays the array elements accessed through a smart pointer `list` (line 21). Note that the parameter `list` in the `printArray` is pass-by-reference for two reasons. First, it is more efficient to pass a `unique_ptr` object by reference. Second, a smart pointer is unique and it cannot be copied.

Ch 12 Summary

Sunday, April 2, 2023 11:10 PM

https://liangcpp.pearsoncmg.com/test/Exercise12_13.txt