6.9 Functions with string/vector parameters

Functions commonly modify a string or vector. The following function modifies a string by replacing spaces with hyphens.

Figure 6.9.1: Modifying a string parameter, which should be pass by reference.

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

// Function replaces spaces with hyphens
void StrSpaceToHyphen(string& modStr) {
   unsigned int i;    // Loop index

   for (i = 0; i < modStr.size(); ++i) {
      if (modStr.at(i) == ' ') {
         modStr.at(i) = '-';
      }
   }
}

int main() {
   string userStr;  // Input string from user

   // Prompt user for input
   cout << "Enter string with spaces: " << endl;
   getline(cin, userStr);

   // Call function to modify user defined string
   StrSpaceToHyphen(userStr);

   // Output modified string
   cout << "String with hyphens: ";
   cout << userStr << endl;

   return 0;
}
```

```
Enter string with spaces:
Hello there everyone.
String with hyphens: Hello-there-everyone.

...

Enter string with spaces:
Good bye  now   !!!
String with hyphens: Good-bye--now---!!!
```

Feedback?

The string serves as function input and output. The string parameter must be pass by reference, achieved using & (yellow highlighted), so that the function modifies the original string argument (userStr) and not a copy.

zyDE 6.9.1: Modifying a string parameter: Spaces to hyphens.

1. Run the program, noting correct output.
2. Remove the & and run again, noting the string is not modified, because the s by value and thus the function modifies a copy. When done replace the &
3. Modify the function to also replace each '!' by a '?'.

Load default template...

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

// Function replaces spaces with hyphens
void StrSpaceToHyphen(string& modStr) {
   unsigned int i;  // Loop index

   for (i = 0; i < modStr.size(); ++i) {
      if (modStr.at(i) == ' ') {
         modStr.at(i) = '-';
      }
   }
}

int main() {
   string userStr;  // Input string from u

   // Prompt user for input
   cout << "Enter string with spaces: " <<
   getline(cin, userStr);
```

```
Hello there everyone!!!
```

Run

Feedback?

Sometimes a programmer defines a vector or string parameter as pass by reference even though the function does not modify the parameter, to prevent the performance and memory overhead of copying the argument that would otherwise occur.

The keyword ***const*** can be prepended to a function's vector or string parameter to prevent the function from modifying the parameter. Programmers commonly make a large vector or string input parameter pass by reference, to gain efficiency, while also making the parameter const, to prevent assignment.

The following illustrates. The first function modifies the vector so defines a normal pass by reference (highlighted yellow). The second function does *not* modify the vector but for efficiency uses constant pass by reference (highlighted orange).

Figure 6.9.2: Normal and constant pass by reference vector parameters in a vector reversal program.

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

void ReverseVals(vector<int>& vctrVals) {
   unsigned int i;  // Loop index
   int tmpVal;      // Temp variable for swapping

   for (i = 0; i < (vctrVals.size() / 2); ++i) {
      tmpVal = vctrVals.at(i); // These statements swap
      vctrVals.at(i) = vctrVals.at(vctrVals.size() - 1 -
i);
      vctrVals.at(vctrVals.size() - 1 - i) = tmpVal;
   }
}

void PrintVals(const vector<int>& vctrVals) {
   unsigned int i;  // Loop index

   // Print updated vector
   cout << endl << "New values: ";
   for (i = 0; i < vctrVals.size(); ++i) {
      cout << " " << vctrVals.at(i);
   }
   cout << endl;
}

int main() {
   const int NUM_VALUES = 8;             // Vector size
   vector<int> userValues(NUM_VALUES);   // User values
   int i;                                // Loop index

   // Prompt user to populate vector
   cout << "Enter " << NUM_VALUES << " values..." << endl;
   for (i = 0; i < NUM_VALUES; ++i) {
      cout << "Value: ";
      cin >> userValues.at(i);
   }

   // Call function to reverse vector values
   ReverseVals(userValues);

   // Print reversed values
   PrintVals(userValues);

   return 0;
}
```

```
Enter 8 values...
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values:  80 70 60 50 40 30 20
10
```

Feedback?

A reader might wonder why all input parameters are not defined as constant pass by reference parameters: Why make local copies at all? The reason is efficiency. For parameters involving just a few memory locations, making a local copy enables the compiler to generate more efficient code, in part because the compiler can place those copies inside a tiny-but-fast memory inside the processor called a register file—further details are beyond our scope.

In summary:

- Define a function's output or input/output parameters as pass by reference.
    - But create output parameters sparingly, striving to use return values instead.

- Define input parameters as pass by value.
    - Except for large items (perhaps 10 or more elements); use constant pass by reference for those.

PARTICIPATION ACTIVITY | 6.9.1: Constants and pass by reference.

How should a function's vector parameter `ages` be defined for the following situations?

1) ages will always be small (fewer than 10 elements) and the function will not modify the vector.

- ○ Constant and pass by reference.
- ○ Constant but not pass by reference.
- ○ Pass by reference but not constant.
- ○ Neither constant nor pass by reference.

2) ages will always be small, and the function will modify the vector.

- ○ Constant and pass by reference.
- ○ Constant but not pass by reference.
- ○ Pass by reference but not constant.
- ○ Neither constant nor pass by reference.

3) ages may be very large, and the function will modify the vector.

- ○ Constant and pass by reference.
- ○ Constant but not pass by reference.
- ○ Pass by reference but not constant.
- ○ Neither constant nor pass by

reference.

4) ages may be very large, and the function will not modify the vector.

- ○ Constant and pass by reference.
- ○ Constant but not pass by reference.
- ○ Pass by reference but not constant.
- ○ Neither constant nor pass by reference.

Feedback?

PARTICIPATION ACTIVITY | 6.9.2: Vector parameters.

Define a function's vector parameter `ages` for the following situations. Assume ages is a vector of integers. Example: ages will always be small (fewer than 10 elements) and the function will not modify the vector: `const vector<int> ages`.

1) ages will always be small, and the function will modify the vector.

```
void MyFct (
                    ) {
```

Check   Show answer

2) ages may be very large, and the function will modify the vector

```
void MyFct (
                    ) {
```

Check   Show answer

3) ages may be very large, and the function will not modify the vector.

```
void MyFct (
                    ) {
```

Check Show answer

Feedback?

CHALLENGE ACTIVITY | 6.9.1: Use an existing function.

Use function GetUserInfo to get a user's information. If user enters 20 and Holly, sample program output is:

```
Holly is 20 years old.
```

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

void GetUserInfo(int& userAge, string& userName) {
   cout << "Enter your age: " << endl;
   cin >> userAge;
   cout << "Enter your name: " << endl;
   cin >> userName;
}

int main() {
   int userAge;
   string userName;

   /* Your solution goes here  */
    GetUserInfo(userAge, userName);

   cout << userName << " is " << userAge << " years old." << endl;

   return 0;
}
```

Run ✓ All tests passed

✓ Testing with inputs: 20 Holly

Your output
```
Holly is 20 years old.
```

✓ Testing with inputs: 40 Andy

Your output
```
Andy is 40 years old.
```

Feedback?

CHALLENGE ACTIVITY

6.9.2: Modify a string parameter.

Complete the function to replace any period by an exclamation point. Ex: "Hello. I'm Miley. Nice to meet you." becomes:

```
"Hello! I'm Miley! Nice to meet you!"
```

```

void MakeSentenceExcited(string& sentenceText) {

   /* Your solution goes here  */
   for(int i = 0; i< sentenceText.size(); i++){
      if (sentenceText.at(i)== '.'){
         sentenceText.at(i) = '!';
         }
   }

}

int main() {
   string testStr;

   getline(cin, testStr);
   MakeSentenceExcited(testStr);
   cout << testStr;

   return 0;
}
```

Run

✓ All tests passed

✓ Testing with: Alright. I understand.

Your output

```
Alright! I understand!
```

✓ Testing with: Howdy friend. Until later...

Your output

```
Howdy friend! Until later!!!
```

✓ Testing with: .Hello.

Your output

```
!Hello!
```

✓ Testing with:

Your output *Your program correctly produced no output*

Feedback?

CHALLENGE ACTIVITY 6.9.3: Modify a vector parameter.

Write a function SwapVectorEnds() that swaps the first and last elements of its vector parameter. Ex: sortVector = {10, 20, 30, 40} becomes {40, 20, 30, 10}.

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

/* Your solution goes here  */
void SwapVectorEnds(vector<int>& sortVector){
   int tmp;
   tmp = sortVector.at(0);
   sortVector.at(0) = sortVector.at(sortVector.size()-1);
   sortVector.at(sortVector.size()-1) = tmp;
}

int main() {
   vector<int> sortVector(4);
   unsigned int i;
   int userNum;

   for (i = 0; i < sortVector.size(); ++i) {
      cin >> userNum;
      sortVector.at(i) = userNum;
   }
```

Run ✓ All tests passed

✓ Testing with inputs: 10 20 30 40

Your output
```
40 20 30 10
```

✓ Testing with inputs: 11 12 13 14

Your output
```
14 12 13 11
```

Feedback?