Nested Loops

Chapter 4, 5

String Algorithms, Sort Algorithms

Consider the following programs

- Printing:

1

12

123

1234

…

- Finding a substring within another string
- Printing multiplication table
- Sorting an array of random numbers

- All Require a loop nested in another loop to accomplish

Nested Loop Example

```
int i, j;
for(i = 0; i < 3; i++)
{
    for(j = 0; j < 2; j++)
    {
        cout << "(" << i << "," << j << ") ";
    }
    cout << endl;
}
```

- What is the output of the above code?

(0, 0) (0,1)

(1, 0) (1,1)

(2, 0) (2,1)

Nested Loop Example

```
int i, j;
for(i = 0; i < 3; i++)
{
   for(j = 0; j < 2; j++)
   {
      cout << "(" << i << "," << j << ") ";
   }
   cout << endl;
}
```

- Inner loop executes multiple times for each iteration of the outer loop
- For every iteration of the outer loop, the inner loop is initialized as if it's the first execution

Iterations of Nested Loop

```
int i, j, k;
for(i = 0; i < 2; i++)
{
   for(j = 0; j < 2; j++)
   {
      for(k = 0; k < 2; k++)
      {
         …
      }
   }
}
```

i	j	k
0	0	0
		1
	1	0
		1
1	0	0
		1
	1	0
		1

Example 8.1

- Print the following pyramid using nested loops:

1

12

123

1234

…

123456789

Strategy for Nested Loops

- Nested loops is inherently more difficult to write code for than non-nested loops
  - Loops inside of loops, inside of other loops
  - Iterations within iterations

- With the right strategy the nested loops become easy to solve

Strategy for Nested Loops

- What is a nested loop actually executing?
  - In general it can be said the inner loop represents an algorithm that will be repeated by the outer loop
  - The outer loop will often set a new set of initial values/constraints for the inner algorithm
  - Often the outer loop may use the result from the inner algorithm to perform some other action between the next iteration

- How does this help us?

Strategy for Nested Loops

- Given the inner algorithm is to be repeated by the outer algorithm, the approach should be the following:
  1. Solve the inner algorithm first for a specific example
  2. Generalize the inner algorithm (make initialization, and conditions variable instead of constants)
  3. Form outer loop by determining how many times the inner algorithm will be repeated
  4. Place the inner loop inside the outer loop
  5. Using the variables and known execution pattern of the outer loop, set the variables of the inner loop

Applying to Example 8.1

1

1 2

1 2 3

1 2 3 4

- What is the inner algorithm?
  - Printing a line of numbers until a threshold is reached
  - Forgetting about the pyramid, lets solve printing 1234 algorithmically

Application of Steps 1,2

1.) Solve the inner algorithm first

```
for(j = 1; j <= 4; j++)
    cout << j;
```

Here we solve for printing 1234

2.) Generalize the inner algorithm

```
for(j = 1; j <= n; j++)
    cout << j;
```

Allows for printing of any line of numbers

Application of Steps 3,4

3.) Form outer loop by determining how many times the inner algorithm will be repeated

```
for(i = 1; i <= 9; i++)
{
   cout << endl;
}
```

There are 9 lines to be printed, therefore we repeat the inner algorithm 9 times. Print newline after each.

4.) Place the inner loop inside the outer loop

```
for(i = 1; i <= 9; i++)
{
   for(j = 1; j <= n; j++)
      cout << j;
   cout << endl;
}
```

Application of Step 5

5.) Using the variables and known execution pattern of the outer loop, set the variables of the inner loop

```
for(i = 1; i <= 9; i++)
{
   for(j = 1; j <= i; j++)
      cout << j;
   cout << endl;
}
```

The start of each line is always at 1 and counts up to whatever line we are on. Therefore, the threshold *n* becomes *i*. Thus on line 2, the inner algorithm will count up to 2 and stop.

Example 8.2

- Find the start location of a substring within another string. If the string cannot be found then return -1;

Example String:

“Sometimes you need to find a string inside another”

Find substring “time” should return 4

Find substring “doesn’t exist” should return -1

Application of Steps

- What is the inner algorithm that the outer loop relies on?
  - Matching if one string is equivalent for another

1.) Solve the inner algorithm first

Suppose we want to compare the two strings below; write an algorithm to do this

word	**t**	**i**	**m**	**a**
	0	1	2	3
cmpWord	**t**	**i**	**m**	**e**
	0	1	2	3

Application of Step 1

word	**t**	**i**	**m**	**a**
	0	1	2	3
cmpWord	**t**	**i**	**m**	**e**
	0	1	2	3

1.) Solve the inner algorithm first

```
match = true;
for(j = 0; j < cmpWord.length() && j < word.length(); j++)
{
    if(cmpWord[j] != word[j])
    {
        match = false;
        break;
    }
}
```

Application of Step 2

2.) Generalize the inner algorithm

```
match = true;
for(j = 0, k = strW; j < cmpWord.length() && k < word.length(); j++, k++)
{
    if(cmpWord[j] != word[k])
    {
        match = false;
        break;
    }
}
```

- Knowing that it is not always the case we start comparing word at index 0, we generalize the start of the comparison
  - Accomplished by using two counters instead of one

Application of Step 3

3.) Form outer loop by determining how many times the inner algorithm will be repeated

```
for(i = 0, match = false; i <= word.length() - cmpWord.length() &&
        !match; i++)
{

}
```

- Algorithm starts comparison always at the start of *word*
- Algorithm goes up till word.length() – cmpWord.length() since if there are only 3 letters left in word and there are 4 letters in cmpWord, it can be said that cmpWord cannot be contained within word
- The algorithm will also stop if a match occurs

Application of Step 4

4.) Place the inner loop inside the outer loop

```
for(i = 0, match = false; i <= word.length() - cmpWord.length() &&
        !match; i++)
{
   for(j = 0, k = strW; j < cmpWord.length() && k < word.length();
          j++, k++)
   {
      if(cmpWord[j] != word[k])
      {
        match = false;
        break;
      }
   }
}
```

Application 5

```
for(i = 0, match = false; i <= word.length() - cmpWord.length() &&
        !match; i++)
{
   for(j = 0, k = i; j < cmpWord.length() && k < word.length();
          j++, k++)
   {
      if(cmpWord[j] != word[k])
      {
        match = false;
        break;
      }
   }
}
```

- Changed strW to i, since the outer loop dictates where the comparison starts

Application 5

```
for(i = 0, match = false; i <= word.length() - cmpWord.length() &&
        !match; i++)
{
   for(j = 0, k = i; j < cmpWord.length(); j++, k++)
   {
      if(cmpWord[j] != word[k])
      {
        match = false;
        break;
      }
   }
}
```

- k < word.length() is made redundant by i <= word.length() - cmpWord.length()
- Therefore it is removed

Final Code

```
for(i = 0, match = false; i <= word.length() - cmpWord.length() &&
!match; i++)
{
  match = true;
  for(j = 0, k = i; j < cmpWord.length(); j++, k++)
  {
    if(cmpWord[j] != word[k])
    {
      match = false;
      break;
    }
  }
}
if(match)
   idx = i - 1;
else
  idx = -1;
```

idx represents the location of the substring for the final output

- We add a final if to reduce the i by one since it will be overshot
- Otherwise, if no match occurs we set idx = -1

Unguided Example 8.3

- Using loops, print a multiplication table that looks like the one shown below:

```
Multiplication Table:
       1   2   3   4   5   6   7   8   9
   1   1   2   3   4   5   6   7   8   9
   2   2   4   6   8  10  12  14  16  18
   3   3   6   9  12  15  18  21  24  27
   4   4   8  12  16  20  24  28  32  36
   5   5  10  15  20  25  30  35  40  45
   6   6  12  18  24  30  36  42  48  54
   7   7  14  21  28  35  42  49  56  63
   8   8  16  24  32  40  48  56  64  72
   9   9  18  27  36  45  54  63  72  81
Press any key to continue . . .
```

- Hint: for formatting lookup the function setw() which controls padding between numbers. Include iomanip library.

Application of Step 1, 2

1.) Solve the inner algorithm first

Printing a line of multiplication with a given number, i.e. 3*1, 3*2, 3*3, …

```
cout << setw(3) << 3;   // Get the repeat of first column
for(j = 1; j <= 9; j++)
   cout << setw(3) << 3*j;
cout << endl;
```

2.) Generalize the inner algorithm

```
cout << setw(3) << n;   // Get the repeat of first column
for(j = 1; j <= 9; j++)
   cout << setw(3) << n*j;
cout << endl;
```

Application of Step 3

3.) Form outer loop by determining how many times the inner algorithm will be repeated

```
for(i = 1; i <= 9; i++)
{

}
```

- We print 9 lines so this code is a trivial counter up to 9

Application of Step 4

4.) Place the inner loop inside the outer loop

```
// Print Table
for(i = 1; i <= 9; i++)
{
   cout << setw(3) << n;   // Get the repeat of first column
   for(j = 1; j <= n; j++)
      cout << setw(3) << n*j;
   cout << endl;
}
```

Application of Step 5

5.) Using the variables and known execution pattern of the outer loop, set the variables of the inner loop

```
// Print Table
for(i = 1; i <= 9; i++)
{
   cout << setw(3) << i;   // Get the repeat of first column
   for(j = 1; j <= i; j++)
      cout << setw(3) << i*j;
   cout << endl;
}
```

- Change *n* to *i*, so that *i* controls the number being multiplied

Finalize

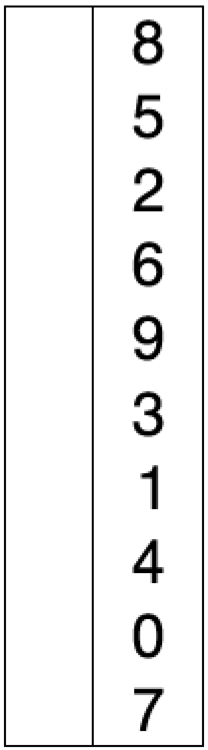
- We are still missing the top line of our table therefore we add some additional code to print it

```
cout << "Multiplication Table:" << endl;

// Repeat of Top Line
cout << setw(3) << " ";
for(i = 1; i <=9; i++)
   cout << setw(3) << i;
cout << endl;

// Print Table
for(i = 1; i <= 9; i++)
{
   cout << setw(3) << i;   // Get the repeat of first column
   for(j = 1; j <= 9; j++)
     cout << setw(3) << i*j;
   cout << endl;
}
```

Example 8.4 (Selection Sort)

- Write a program to do the following
  1. Create a 20 element array of integers
  2. Fill the array with random numbers between 0 to 100
  3. Print the random array
  4. Sort the array using the selection sort method
  5. Print the sorted array

Visualization of Selection Sort



- Selection sort starts at the first index and finds the lowest number in the array
  - When the lowest number is found the numbers are swapped placing the smallest number in the first position
- The index then moves to the right and then the lowest number is searched for again excluding the numbers already sorted
- Executes in $\theta(n^2)$ where n is the number of elements

Animation Courtesy of Wikipedia

	8	5	2	6
divide				
sIdx				
iMin				

	8	5	2	6
divide				
sIdx				
iMin				

	8	5	2	6
divide				
sIdx				
iMin				

Swap

	8	5	2	6
divide				
sIdx				
iMin				

	2	5	8	6
divide				
sIdx				
iMin				

	2	5	8	6
divide				
sIdx				
iMin				

	2	5	8	6
divide				
sIdx				
iMin				

Swap

	2	5	8	6
divide				
sIdx				
iMin				

	2	5	8	6
divide				
sIdx				
iMin				

	2	5	8	6
divide				
sIdx				
iMin				

Swap

	2	5	8	6
divide				
sIdx				
iMin				

- List is now in order, sorting is complete

	2	5	6	8
divide				
sIdx				
iMin				

Components of Algorithm

- Swap two numbers
- Search for the Lowest Number
- Move the divide index

Swapping two Numbers in an Array

x	8	5	2	6
idx	0	1	2	3

- Swap x[0] with x[2]
- Bad Method (Loses Value)

```
x[0] = x[2];
x[2] = x[0]
```

- Must Introduce Third Variable

```
tmp = x[0];
x[0] = x[2];
x[2] = tmp;
```

Searching Array

x	8	5	2	6
idx	0	1	2	3

- Find the index of the Lowest Number
  - Requires two indices
    - Track Lowest #
    - Track Location of Search

```
for(j = 0, lowIdx = 0; j < 4; j++)
{
  if(x[j] < x[lowIdx])
    lowIdx = j;
}
```

Moving the index

x	8	5	2	6
idx	0	1	2	3

- Index moves from the start of the array to n-2, where n is the length
  - If the index reaches n-1 (in this case 3), the array can be assumed to be sorted

```
for(idx = 0; idx < 4 - 1; idx++)
{

}
```

Back to Example 8.4

- Write a program to do the following
  1. Create a 20 element array of integers
  2. Fill the array with random numbers between 0 to 100
  3. Print the random array
  4. Sort the array using the selection sort method
  5. Print the sorted array

Tasks 1-3

1. Create a 20 element array of integers

```
int x[20];
```

2. Fill the array with random numbers between 0 to 100

```
srand((unsigned int)time(NULL));
for(i = 0; i < 20; i++)
   x[i] = rand() % 101;
```

3. Print the random array

```
for(i = 0; i < 20; i++)
   cout << x[i] << " ";
cout << endl;
```

Task 4 – Sorting, App Step 1

1.) Inner algorithm

- Search for lowest number and swap

```
for(j = 0, lowIdx = 0; j < 20; j++)
{
  if(x[j] < x[lowIdx])
       lowIdx = j;
}
tmp = x[0];
x[0] = x[lowIdx];
x[lowIdx] = tmp;
```

Application of Step 2

2.) Generalize Inner Algorithm

```
for(j = strIdx, lowIdx = strIdx; j < 20; j++)
{
  if(x[j] < x[lowIdx])
       lowIdx = j;
}
tmp = x[strIdx];
x[strIdx] = x[lowIdx];
x[lowIdx] = tmp;
```

- Swapping and searching does not start at the same location each time

Application of Step 3

3.) Form outer loop by determining how many times the inner algorithm will be repeated

- Moving the index

```
for(i = 0; i < 20; i++)
{

}
```

Application of Step 4

4.) Place the inner loop inside the outer loop

```
for(i = 0; i < 20; i++)
{
  for(j = strIdx, lowIdx = strIdx; j < 20; j++)
  {
      if(x[j] < x[lowIdx])
         lowIdx = j;
  }
  tmp = x[strIdx];
  x[strIdx] = x[lowIdx];
  x[lowIdx] = tmp;
}
```

Application of Step 5

5.) Using the variables and known execution pattern of the outer loop, set the variables of the inner loop

```
for(i = 0; i < 20; i++)
{
  for(j = i, lowIdx = i; j < 20; j++)
  {
      if(x[j] < x[lowIdx])
         lowIdx = j;
  }
  tmp = x[i];
  x[i] = x[lowIdx];
  x[lowIdx] = tmp;
}
```

- Outer loop controls start of search
  - Replace *strIdx* with *i*