C++ Programming for Engineers

Petar Jervic
Francisco Luro
Andrew Moss (course-resp)

May 2, 2012

Overview

- For-loops motivation
- While vs For-loops
- Datatypes for new problems
- Problems

For loops

Why another loop construct ?

- Recurring problem/solution
  - Print all the elements in a list
  - Convert to lower case each character in a string
  - Search for the smallest value in a list
  - ...

```
int i=0,max;
cin » max;
while(i < max) {
  cout « "i" « endl;
  i++;
}
```

Same behavior, new syntax

```
int i=0,max;
cin » max;
while(i < max) {
  cout « "i" « endl;
  i++;
}
// can access i here...
```

```
int max;
cin » max;
for (int i=0; i < max; i++) {
  cout « "i" « endl;
}
// cannot access i outside the body of the loop
```

New syntax

Syntax consists of 4 parts

- counter definition
  - for(int i=0; i < max; i++) { }
- termination condition
  - for(int i=0; i < max; i++) { }
- counter update
  - for(int i=0;i < max; i++) { }
- body of the loop
  - for(int i=0;i < max; i++) { }

While vs For loop

Unknown number of iterations

- While (preferred)

```
char c = ' ';
while(c != 'q') {
  cin » c;
  // do something...
}
```

- For loop

```
for (char c = ' ' ; c != 'q' ; cin » c ) {
  // do something...
}
```

While vs For loop

Known number of iterations

- While

```
int i=0;
while(i < myString.length()) {
  myString[i] = tolower(myString[i]);
  i++;
}
```

- For loop (preferred)

```
for (int i=0 ; i < myString.length(); i++) {
  myString[i] = tolower(myString[i]);
}
```

More on syntax

Each place holder in the for-loop admits several definitions and increments

```
for (int i=0,j=0; i < N; i+=1, j+=2)
  cout « "(" « i « "," « j « ")" « endl;
```

(0, 0)
(1, 2)
(2, 4)
(3, 6)
...

Real world problems

Before loops, exercises were

- small problems
- repetition of code
- known size of the problem in advance

Now

- we can process many elements
- with less code
- unknown of the size of the problem

But...

- we need more general datatypes (like strings)
  - int a,b,c,d,e,f,g,....;

Some additional definitions

We have used strings as collection of characters

```
string S = "Hello";
cout « S;
cout « S[0] « S[1] « S[2] « S[3] « S[4];
```

What about collections of integers and floats ?

```
// array of ints
int A[4];
cin » A[0] » A[1] » A[2] » A[3];
cout « A[3] « A[2] « A[1] « A[0];
```

Some additional definitions (2)

Formally, the array definition

- Known size at compile time

```
// constant expression!
datatype variableName1[255];
```

- Known size at runtime: (* is not multiplication in this context)

```
// dynamic memory allocation
datatype *variableName2 = new datatype[expression];
```

- The usage is the same syntax

```
variableName1[3] = 5;
variableName2[3] = 5;
```

Some additional definitions (3) - Examples

Definition of collections of elements (Arrays)

- Known size (3 elements vector)

```
float vec[3];
vec[0] = 0;
vec[1] = 2;
vec[2] = vec[0];
```

- Unknown size / Runtime (N-vector)

```
int N;
cin » N;
float *vec = new float[N];
vec[0] = 0;
vec[5] = vec[0];
vec[N-1] = 0;
```

Problem 1

- Apply a simple linear function ($y = ax$) to an array of N numbers within [0,100]
  - Read N, the amount of values to process.
  - Read the constant a.
  - Allocate a dynamic array to hold the values
  - Generate N random floats and save them into the array
  - Compute the function
  - Print both arrays (before and after the function)

```
Input: a = 2.0, N = 5
Output:
5 98 1 8 4
10 196 2 16 8
```

Problem 1 (code)

```
float a,N;
cin » a » N;
float *vec1 = new float[N];
float *vec2 = new float[N];
for (int i=0; i < N; i++) {
  vec1[i] = rand()%100;
  cout « vec1[i] « " ";
  vec2[i] = a * vec1[i];
}
cout « endl;
for (int i=0; i < N; i++)
  cout « vec2[i] « " ";
cout « endl;
```

Problem 2

- Using for loops, generate the following pattern:
- Example for N = 4

```
12344321
123  321
12    21
1      1
```

Problem 2 (cont)

Given N = 4,

```
12344321
123  321
12    21
1      1
```

Analysing the output

- How many lines ?
- How many elements in the first row ?
  - How do they behave ?
- How many elements in the second ?
- How many blanks per line ?

Problem 3 (cont)

```
12344321
123  321
12    21
1      1
```

One possible view: (assuming fixed font size)

- pattern made of N lines, each with:
  - 1 sequence of increasing numbers
  - 1 sequence of 0 or more spaces
  - 1 sequence of decreasing numbers

Problem 3 (cont)

Given N=4,

```
12344321
123  321
12    21
1      1
```

Translates to,

```
for (int i=0; i<N; i++) {
  for (int j=1; j <=(N-i); j++)
    cout « j;
  for (int j=0; j < 2*i; j++)
    cout « ' ';
  for (int j=(N-i); j > 0; j-)
    cout « j;
  cout « endl;
}
```

Problem 3 (cont)

What about the following pattern ?

```
********
***    ***
**        **
*          *
```