

The do Loop (4.3)

The `while` loop's condition test is the first thing that occurs in its execution.

The `do` loop (or `do-while` loop) has its condition tested only after at least one execution of the statements.

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

The do Loop

This means that the `do` loop should be used only when the statements must be executed before there is any knowledge of the condition.

This also means that the `do` loop is the least used loop.

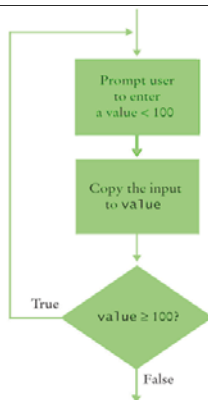
C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

The do Loop

What problems require something to have happened before the testing in a loop?

Getting valid user input is often cited.

Here is the flowchart for the problem in which the user is supposed to enter a value less than 100 and processing must not continue until they do.



C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

The do Loop

Here is the code:

```
int value;
do
{
    cout << "Enter a value < 100";
    cin >> value;
} while (value >= 100);
```

In this form, the user sees the same prompt each time until they enter valid input.

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

The do Loop

In order to have a different, "error" prompt that the user sees only on *invalid* input, the initial prompt and input would be before a `while` loop:

```
int value;
cout << "Enter a value < 100";
while (value >= 100);
{
    cout << "Sorry, that is larger than 100\n";
    << "Try again: ";
    cin >> value;
}
```

Notice what happens when the user gives valid input on the first attempt: nothing – good.

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Nested Loops (4.4)



For each hour, 60 minutes are processed – a nested loop.

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Nested Loops

- Nested loops are used mostly for data in tables as rows and columns.
- The processing **across the columns** is a **loop**, as you have seen before, "nested" inside a loop for **going down the rows**.
- Each row is processed similarly so design begins at that level. After writing a loop to process a generalized row, that loop, called the "inner loop," is placed inside an "outer loop."

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

Nested Loops

EX: Write a program to produce a table of powers.
The output should be something like this:

x^1	x^2	x^3	x^4
1	1	1	1
2	4	8	16
3	9	27	81
...
10	100	1000	10000

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

Nested Loops

- The first step is to solve the "(inner)" loop.
- There are four columns and in each column we display the power. Using x to be the number of the row we are processing, we have (in pseudo-code):

```
for n from 1 to 4
{
    print  $x^n$ 
}
```

* You would test that this works in your code before continuing. If you can't correctly print one row, why try printing lots of them?

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

Nested Loops

Now, putting the inner loop into the whole process we have:

(don't forget to indent, nestedly)

```
print table header
for x from 1 to 10
{
    print table row
    print endl
}
```

outer loop
inner loop

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

The Complete Program for Table of Powers

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main()
{
    const int NMAX = 4;
    const double XMAX = 10;

    // Print table header
    for (int n = 1; n <= NMAX; n++)
    {
        cout << setw(10) << n;
    }
    cout << endl;
    for (int n = 1; n <= NMAX; n++)
    {
        cout << setw(10) << "x ";
    }
    cout << endl << endl;
}
```

ch04/powtable.cpp

when use power function base must be a real & (double)

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

The Complete Program for Table of Powers

```
// Print table body
for (double x = 1; x <= XMAX; x++)
{
    // Print table row
    for (int n = 1; n <= NMAX; n++)
    {
        cout << setw(10) << pow(x, n);
    }
    cout << endl;
}

return 0;
}
```

inner loop
outer loop

The program run would be:

1	2	3	4
x	x	x	x
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

More Nested Loop Examples

The loop variables can have a value relationship. In this example the inner loop depends on the value of the outer loop.

```
for (i = 1; i <= 4; i++)
{
    for (j = 1; j <= i; j++)
        cout << " * ";
    cout << endl;
}
```

The output will be:

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

Copyright © 2008 by John Wiley & Sons. All rights reserved.

More Nested Loop Examples

```
for (i = 1; i <= 4; i++)
    for (j = 1; j <= i; j++)
        cout << " * ";
cout << endl;
```

j is the number of "columns" (or the line's length), which depends on the line number, i

j stops at: i i i i
1 2 3 4

line num.	i	1	2	3	4
i 1	*				
i 2	*	*			
i 3	*	*	*		
i 4	*	*	*	*	

i represents the row number or the line number

Copyright © 2008 by John Wiley & Sons. All rights reserved.

More Nested Loop Examples

In this example, the loop variables are still related, but the processing is a bit more complicated.

```
for (i = 1; i <= 3; i++)
{
    for (j = 1; j <= 5; j++)
    {
        if (i + j % 2 == 0) { cout << " * "; }
        else { cout << " "; }
    }
    cout << endl;
}
```


The output will be:

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

i=1 j=1 2 3 4 5
i+j: 2 even 3 not even 4 even 5 not even
i=2 j=1 2 3 4 5
i+j: 3 not even 4 even 5 not even 6 even 7 not even

Copyright © 2008 by John Wiley & Sons. All rights reserved.

Processing Input – When and/or How to Stop? (4.5)

- We need to know, when getting input from a user, when they are done.
- One method is to hire a sentinel (as shown)  or more correctly choose a value whose meaning is STOP!
- As long as there is a known range of valid data points, we can use a value not in it.

Copyright © 2008 by John Wiley & Sons. All rights reserved.

Processing Input – When and/or How to Stop?

EX: • We will write code to calculate the average of some salary values input by the user.

How many will there be?

That is the problem. We can't know.

But we can use a **sentinel value**, as long as we tell the user to use it, to tell us when they are done.

- Since salaries are never negative, we can safely choose -1 as our sentinel value.

Copyright © 2008 by John Wiley & Sons. All rights reserved.

Processing Input – When and/or How to Stop?

- In order to have a value to test, we will need to get the first input before the loop. The loop statements will process each non-sentinel value, and then get the next input.
- Suppose the user entered the sentinel value as the first input. Because averages involve division by the count of the inputs, we need to protect against dividing by zero. Using an **if-else** statement from Chapter 3 will do.

Copyright © 2008 by John Wiley & Sons. All rights reserved.

The Complete Salary Average Program

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

ch04/sentinel.cpp

```
int main()
{
    double sum = 0;
    int count = 0;
    double salary = 0;

    // get all the inputs
    cout << "Enter salaries, -1 to finish: ";
    cin >> salary;
    while (salary != -1)
    {
        // process input
        sum = sum + salary;
        count++;

        // get next input
        cin >> salary;
    }
}
```

© 2008 for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

The Complete Salary Average Program

```
// process and display the average
if (count > 0)
{
    double average = sum / count;
    cout << "Average salary: " << average << endl;
}
else
{
    cout << "No data" << endl;
}

return 0;
}
```

A program run:

```
Enter salaries, -1 to finish: 10 10 40 -1
Average salary: 20
```

© 2008 for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Using Failed Input for Processing

- Sometimes it is easier and a bit more intuitive to ask the user to "Hit Q to Quit" instead of requiring the input of a sentinel value.
- Sometimes picking a sentinel value is simply impossible – if any valid number is allowed, which number could be chosen?

© 2008 for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Using Failed Input for Processing

- In the previous chapter we used `cin.fail()` to test if the most recent input failed.
- Note that if you intend to take more input from the keyboard after using failed input to end a loop, you must reset the keyboard with `cin.clear()`.

© 2008 for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Using Failed Input for Processing

If we introduce a bool variable to be used to test for a failed input, we can use `cin.fail()` to test for the input of a 'Q' when we were expecting a number:

© 2008 for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Using Failed Input for Processing

```
cout << "Enter values, Q to quit: ";
bool more = true;
while (more)
{
    cin >> value;
    if (cin.fail())
    {
        more = false;
    }
    else
    {
        // process value here
    }
}
cin.clear() // reset if more input is to be taken
```

© 2008 for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Using Failed Input for Processing

- Using a `bool` variable in this way is disliked by many programmers.

Why?

- `cin.fail` is set when `>>` fails. This allows the use of an input *itself* to be used as the test for failure.
- Again note that if you intend to take more input from the keyboard, you must reset the keyboard with `cin.clear`.

© for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

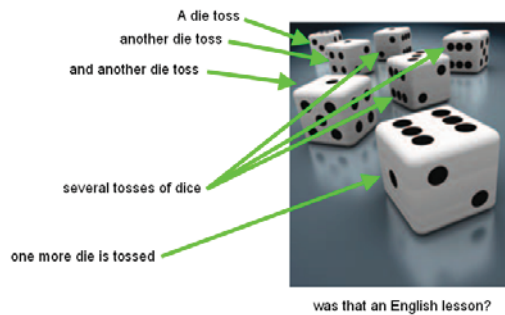
Using Failed Input for Processing

Using the input attempt directly we have:

```
cout << "Enter values, Q to quit: ";  
while (cin >> value)  
{  
    // process value here  
}  
cin.clear();
```

© for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

Random Numbers and Simulations (4.6)



© for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

Simulations

A *simulation program* uses the computer to simulate an activity in the real world (or in an imaginary one).

© for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

Simulations

- Simulations are commonly used for
 - Predicting climate change
 - Analyzing traffic
 - Picking stocks
 - Many other applications in science and business

© for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

Randomness for Reality (Simulating)

- Programmers must model the "real world" at times.
- Consider the problem of modeling customers arriving at a store.

Do we know the rate?

Does anyone?

How about the shopkeeper!

© for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved