CS111

Introduction to Computing Science

Previously

Previously

- whiles
- for
- do while

Today

- Processing input
- When to stop
- Nested loops

Processing Input – When and/or How to Stop?

- 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.

Processing Input – When and/or How to Stop?

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.

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.
- For averages we need the total sum, and the total number of inputs.

Pseudo code

- ask for input
- while input is not negative
  - update totals
  - ask for input
- compute average

The Complete Salary Average Program

```
int main()
{
   double sum = 0;
   int count = 0;
   double salary = 0;
   cout << "Enter salaries, -1 to finish: ";
   cin >> salary;
   while (salary != -1)
   {
        sum = sum + salary;
        count++;
        cin >> salary;
    }
   cout << "The average is: " << sum/count << endl;
   return 0;
}
```

Using Failed Input for Processing

- Sometimes it is easier and a bit more intuitive to ask the user to “Hit Q to Quit” instead or 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?

Using Failed Input for Processing

- Your program will fail if you for example enter a character ‘x’, when the program wants to read an integer.

```
int number;
cout << "Enter a number";
cin >> number;
```

- If you enter a character ‘x’ or a string “five” the input will fail.

- You can use **`cin.fail()`** to test if the most recent input failed.

Using Failed Input for Processing

```
int number;
cout << "Enter a number";
cin >> number;

while(cin.fail()){
   …// Reenter number;
}
```

- If input fails (`cin.fail()` is true) you have to first clear it with `cin.clear();`

Using Failed Input for Processing

```
int number;
cout << "Enter a number";
cin >> number;

while(cin.fail()){
   cin.clear();
   …// Reenter number;
}
```

Declare a string variable, and read into it. String can read any input.

Remember, the program failed to read 'x' or "five"

- The offending input however is still in the input stream.
- To deal with this you have to read (and remove the offending input).

Using Failed Input for Processing

```
int number;
cout << "Enter a number";
cin >> number;

while(cin.fail()){
   cin.clear();
   string not_an_int;
   cin >> not_an_int;
   cout << "Try again:";
   cin >> number;
}
```

Declare a string variable, and read into it. String can read any input.

Now the offending input is removed.

*Another attempt to read **number**.*

- The offending input however is still in the input stream.
- To deal with this you have to read (and remove the offending input).

Using Failed Input for Processing

- You can combine **`cin.fail()`** with other requirements

```
int number;
cout << "Enter a positive number";
cin >> number;

while(cin.fail() || number<0){
    if(cin.fail()){
            cin.clear();
            string not_an_int;
            cin >> not_an_int;
   }
   cout << "Try again:";
   cin >> number;
}
```

Using Failed Input for Processing

- How to use `cin.fail()` as sentinel.

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

Using Failed Input for Processing

- How to use `cin >> value` itself as sentinel.

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

`cin >> value` *returns true if it succeeds to read, and false otherwise.*

Nested Loops

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

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.”

Nested Loops

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

Nested Loops

- The first step is to solve the “nested” 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?

Nested Loops

- As a for loop it would look like this:

```
for (int n = 1; n <= NMAX; n++)
      {
         cout << setw(10) << pow(x, n);
      }
```

setw(10) is for layout. We cover this next week.

- Similarly you can print the first row, the header

```
for (int n = 1; n <= NMAX; n++)
   {
      cout << setw(10) << n;
   }
```

Nested Loops

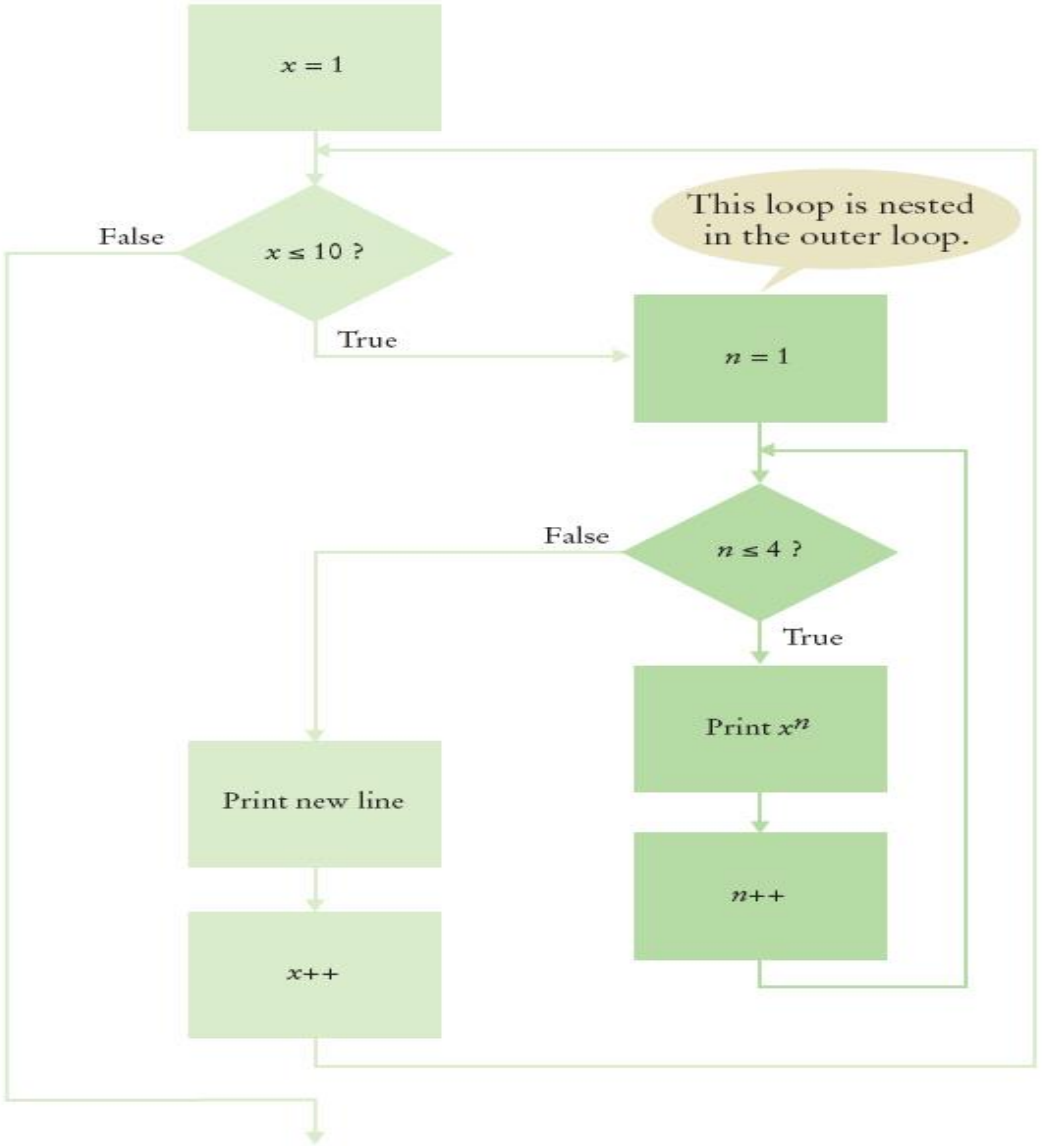
Now, putting the inner loop
into the whole process we have:

(don't forget to indent, nestedly)

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

}
```

Nested Loops



The Complete Program for Table of Powers

```
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;
```

…

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;
}
```

```
         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
```

Common Solutions

- Program to find the sum of Positive numbers
  - Let us assume that all numbers we are interested in are >=0 but we don’t know how many numbers there are and whether in any particular sequence.
  - We can type in the numbers one after the other and when we get to the end we can type in a negative number which will stop the loop.

Common Solutions

- Program to find the sum of Positive numbers
  - What are the inputs?
    - The input are numbers (int)
  - Are there any constants?
    - No
  - What other variables do we need?
    - The sum (int)
  - What is the output?
    - The sum

Common Solutions

- Pseudo-code
  - Ask for a number
  - While number positive
    - add number to the total sum
    - ask for a number
  - Print the sum

Common Solutions

```
#include <iostream>
#include <stdlib.h>
int main()
{
int sum  = 0, num;
   cout << "Enter Number: ";
   cin >> num;
   while (num >= 0){
      sum = sum + num;
      cout << "\nEnter Number: ";
      cin >> num;
   }
   cout << "\nSum is: " << sum << endl;
   system("PAUSE");
   return 0;
}
```

Common Solutions

- Program to find the average of Positive numbers
  - Let us assume that all numbers we are interested in are >=0 but we don’t know how many numbers there are and whether in any particular sequence.
  - We can type in the numbers one after the other and when we get to the end we can type in a negative number which will stop the loop.

Common Solutions

- Program to find the sum of Positive numbers
  - What are the inputs?
    - The input are numbers (double)
  - Are there any constants?
    - No
  - What other variables do we need?
    - The sum (double)
    - A counter (double)
  - What is the output?
    - The sum/counter

Common Solutions

- Pseudo-code
  - Ask for a number
  - While number positive
    - add number to the total sum
    - increase the counter
    - ask for a number
  - Print the sum/counter

Common Solutions

```
#include <iostream>
#include <stdlib.h>
int main()
{
   double sum  = 0, num, counter = 0;
   cout << "Enter Number: ";
   cin >> num;
   while (num >= 0){
      sum = sum + num;
      counter++;
      cout << "\nEnter Number: ";
      cin >> num;
   }
   cout << "\nAverage is: " << sum/counter << endl;
   system("PAUSE");
   return 0;
}
```

Common Solutions

- Program to find the maximum of Positive numbers
  - Let us assume that all numbers we are interested in are >=0 but we don’t know how many numbers there are and whether in any particular sequence.
  - We can type in the numbers one after the other and when we get to the end we can type in a negative number which will stop the loop.

Common Solutions

- Program to find the sum of Positive numbers
  - What are the inputs?
    - The input are numbers
  - Are there any constants?
    - No
  - What other variables do we need?
    - The cuurent maximum
  - What is the output?
    - The maximum

Common Solutions

- Pseudo-code
  - Ask for a number
  - While number positive
    - if the number if larger than current maximum
      - the number becomes the new maximum
    - ask for a number
  - Print the maximum

Common Solutions

```
#include <iostream>
#include <stdlib.h>
int main()
{
   double maximum  = 0, num;
   cout << "Enter Number: ";
   cin >> num;
   while (num >= 0){
      if(num>maximum){
            maximum=num;
      }
      cout << "\nEnter Number: ";
      cin >> num;
   }
   cout << "\nMaximum is: " << maximum << endl;
   system("PAUSE");
   return 0;
}
```

Exercise: Sum of Digits

- The sum of digits of a natural number is just the sum of the digits.
- The sum of digits of 274 is 2+7+4=13

- How to compute the sum of digits?
  - How to compute the digits one by one?

Exercise: Sum of Digits

- How to compute the digits one by one?
  - The last digit of a number *x* is the number *x mod 10*
  - In C++ this would be **`x % 10`**.
  - Example: **`274 % 10 = 4`**
  - The second last digit becomes the last digit by integer division with 10.
  - In C++ this would be **`x / 10`**. (`x` and 10 are **`ints`**)
  - Example 274 / 10 = 27
  - The digit sum of an inter between 0 and 9 is the number itself.

Exercise: Sum of Digits

- Program to find the sum of digits
  - What are the inputs?
    - The input is a positive integer (natural number).
  - Are there any constants?
    - No
  - What other variables do we need?
    - The current sum of digits
  - What is the output?
    - The sum of digits

Exercise: Sum of Digits

- Pseudo-code
  - Ask for a number
  - While number larger than 10
    - add the last digit to the sum
    - remove the last digit from the number by division.
  - Print the sum

Exercise: Sum of Digits

- Pseudo-code
  - Ask for a number
  - While number larger than 10
    - add the last digit to the sum
    - remove the last digit from the number by division.
  - Print the sum