CS 101: Computer Programming and Utilization

08-C++ control flow statements

Instructor: Sridhar Iyer
IIT Bombay
Activity: Write a program
to compute factorial of a given number

Write it in at least 2 of the following:

● psuedo-code
● Scratch
● C++
• Input N from keyboard; Initialize M to 1;
• Repeat M = M*i, where i goes from 1 to N+1
• Output M to display

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

int main() {
    int num, nFactorial = 1;
    cout << "give the value of num: "; cin >> num;
    for (int i = 1; i <= num; i++) {
        nFactorial *= i;
    }
    cout << " nFactorial is: " << nFactorial << endl;
    return 0;
}
```
Modify the program to

Calculate factorial for many numbers, taking each one from the input

Run: demo08-factorial.cpp and its modification. Also demo08-factorial.sb
C++ constructs seen so far

- Including libraries, namespaces:
  - `#include<iostream>; using namespace std`
- Functions: `main()`
- Data types and variable declarations:
  - `int n; float nFactorial; char flag = 'y';`
- Arithmetic operations, expressions: `5*(F-32)/9`
  - Boolean values and operations: `(x && y)`
  - Type conversions: Experiments in lab 04
- Assignment statement: `c = 5*(F-32)/9;`
- Compiling and executing a C++ program
Assigning values to variables - Similar to Variables block in Scratch

• Lhs = Rhs
• Lhs is a variable name
  • Later we will consider arrays, pointers etc.

• Rhs is an expression compatible with the type of the lhs
  • centigrade = 5*(fahrenheit - 32)/9;

• Assignment statement has value = rhs
  • Lets us cascade x = y = z+1;
Logical expressions – Similar to Operator block in Scratch

• Compare numeric expressions
  • $5 < 13$, $1e5 \geq 2e6$
  • $a == b+1$ (note the double equals), $c != d$

• Each expression is true or false
  • Internally represented as integers 1 and 0

• Combine using $and$, $or$, $not$
  • $(a == b+1) \&\& (c != d) \mid\mid !(e <= f)$

• Operator precedence like with numbers

• Logical expressions used to control the execution of statements
Think-Pair-Share: swapping two numbers

```java
float x = 5, y = 11;
float temporary = x;
x = y;
y = temporary;
```

Can you swap without using a temporary variable?

<table>
<thead>
<tr>
<th>Hand Execution</th>
<th>x</th>
<th>y</th>
<th>temporary</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>11</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Or

Single Stepping
More C++ constructs seen today

- Compound assignment: nFactorial *= i;
- Increment/Decrement: i++ (different from ++i)
- Condition blocks: if (flag != 'y') { ... };
- Conditional expressions: c = (a < b)? a : b;
  - If (a < b) c = a; else c = b;
- Loops: for (i=0; i <= n; i++) { ... };
- Nested loops: while () { ... for () {...}; ... };
- Infinite loops and break: while (1) {... break};
While and for

while (cond) { stmt }

for (init; cond; stepper) { stmt }
Another Activity: Try this solo

- How many times must we divide a number by 10 until the result goes below 1?

- Given input x

- Output should be numDivs –
  - the number of times you had to divide x by 10 before you got the result to go below 1

- Do this on your own:
  - Assume that this is a quiz question and write
  - First get the logic in pseudo-code, then C++
Iteration: Approximating the logarithm

• How many times must we divide a number \( x \) by 10 until the result goes below 1?

• \( \text{while (condition) statementOrBlock} \)

```cpp
float x;
cin >> x;
int numDivs = 0;
while (x > 1) {
    x = x / 10;
    numDivs = numDivs + 1;
}
cout << numDivs;
```

Another shorthand:
\[ x /= 10; \]

More shorthands:
\[ numDivs += 1; \]
or
\[ ++numDivs; \]
How much have you learnt?

• Rather than my describing constructs in C++, you have learnt them by directly using them
  • that too, in a short span of time!
  • For descriptions of these constructs, read the notes

• Two reasons for this achievement of yours:
  • We wrote pseudo-code to first get the logic of the program right, without worrying about syntax
  • We wrote Scratch programs to develop familiarity with the logic of many commonly used constructs, so transitioning to C++ syntax is easier
Notes
Memory, values, variables

- Unit of storage: bit (0/1)
- Because such computers are easier to implement by switching transistors off and on

- A byte is 8 bits wide
  - Values range from 00000000 to 11111111
  - $2^8 = 256$ possible bit configurations
  - Can be interpreted as integers from 0 to 255 ("unsigned char")
  - Electronic and magnetic memory is allocated in units of bytes
Binary arithmetic

- Byte value in binary: 00000000 (8 bits)
- Corresponding decimal value = 0
- Written as $0_{\text{dec}}$ to avoid confusion

- In decimal, to increment a number, increment the unit position, carry over... etc; Same in binary

- Next few values are:
  - 00000001 ($=1_{\text{dec}}$), 00000010 ($=2_{\text{dec}}$), 00000011 ($=3_{\text{dec}}$),
  - 00000100 ($=4_{\text{dec}}$), 00000101 ($=5_{\text{dec}}$) etc.
Character (char)

• Typically, a character is the same as a 8-bit byte
  • (More recently, multi-byte characters have been designed to support all the world’s languages)

• The key difference is in how the byte is interpreted and processed (e.g., printed)

• E.g., $1100001 \text{ (97}_{\text{dec}})$ means ‘a’, $98_{\text{dec}}$ = ‘b’,
  $1000001 \text{ (65}_{\text{dec}})$ = ‘A’, $66_{\text{dec}}$ =‘B’ etc

• How to distinguish between char and integer?
Fixed size integer types

- "Short integers" (short) are 16 bits wide
  - 65536 possible values

- Standard integers (int) are 32 bits wide
  - 4,294,967,296 possible values

- A long long int is 64 bits wide
  - Will sometimes call long for brevity (as in Java)

- Real numbers are represented using float and double ("double precision") ... later
Real number representations

- “Floating (decimal) point”
- In decimal we write $0.314 \times 10^{11}$
- $0.314$ is the mantissa, $11$ is the exponent
- Mantissa has decimal point at beginning
- Same approach in computers, with radix 2 instead of 10

- In a float
  - 1 sign, 8 exponent, 23 mantissa bits

- In a double
  - 1 sign, 11 exponent, 52 mantissa bits
## Floating point numbers

<table>
<thead>
<tr>
<th></th>
<th>Costs how many bits to store</th>
<th>Magnitude of maximum value</th>
<th>Magnitude of minimum value</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>32</td>
<td>$3.4 \times 10^{38}$</td>
<td>$1.4 \times 10^{-45}$</td>
</tr>
<tr>
<td>double</td>
<td>64</td>
<td>$1.798 \times 10^{308}$</td>
<td>$4.9 \times 10^{-324}$</td>
</tr>
</tbody>
</table>

- Finite bits cannot represent all real values

- Need care in writing expressions that combine values to avoid errors, minimize loss of precision
Operations on numeric types

• All integers support +, −, *, /, % (remainder)
• Float and double support +, −, *, /

• More complicated operations like log, exp, sine, etc. are implemented as functions

• You can compare numbers using comparison operators <, <=, ==, >=, !=
  • The result is a Boolean (0/1) value
  • cout << (5 > 7);
  • cout << (4 != 3);
Boolean values and operations

- In C++, int can be reused as Boolean (0 = false, anything else is true)
- Binary operator && (and)
- Binary operator || (or)

- Such a table is called a Truth Table in logic
- x = “A is a cs101 student”
- y = “A stays in H8”
- When is (x&&y) TRUE ?
- More examples ...

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x &amp;&amp; y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

| x | y | x || y |
|---|---|------|
| 0 | 0 |   0  |
| 0 | 1 |   1  |
| 1 | 0 |   1  |
| 1 | 1 |   1  |
Not and ex(clusive) or

- Unary operator ! (not)
- Binary operator exor

- exor is not available on single Booleans but instead on bit vectors (later class)

- Old C++ used int to store Boolean values
- ANSI standard C++ has a type called bool

<table>
<thead>
<tr>
<th>Input x</th>
<th>Output !x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x ^ y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Compound assignment

/* read two numbers from cin, print sum of their squares */

int sum = 0, num;
cin >> num;
sum += (num * num);
cin >> num;
sum += (num * num);
cout << sum << endl;

How about

int va = 5;
int vb = (va += 2);

cout << sum << endl;

“expression with side effect” - va is modified
Increment/decrement

- Special case of compound assignment
- Syntax: `++va` and `vb++`
  - `++va` means increase `va` by one and then access the incremented value
  - `vb++` means access the current value of `vb` and then increment it before any further access
    - May be slightly inefficient because the old value must be remembered
      - `vb = 2 * (va++);`

- Similarly `va--` and `--vb`
Statement block

- A simple statement assigns a variable the value of an expression
- A block looks like
  `{statement; statement; ... statement;}`
- 0 or 1 statement allowed for uniformity
- Walk down the list executing one statement after another
- Effect of each statement on memory completes before next executed

```c
float x = 5, y = 11;
float temporary = x;
x = y;
y = temporary;
```

- Note on scope: Outside {...} cannot use variables declared inside
If-then-else

- Store in variable \( b \) the absolute value of variable \( a \)

- Store in variable \( c \) the smaller of the values of variables \( a \) and \( b \)

- Else part is optional
- Cascades/nests allowed
- Statement blocks also optional but best used

```cpp
int a, b;
cin >> a;
if (a >= 0) {
b = a;
}
else {
b = -a;
}
```
Example: withdrawing money from bank

```cpp
cin >> deduct;
if (deduct > balance) {
    cout << "Account overdrawn\n";
}
else {
    cout << "Successfully withdrawing " << deduct << endl;
    balance -= deduct;
    // emit paper money
}
```
Curly brackets

- You can also write then or else parts without curly brackets, but this could be dangerous
- Best to always use curly brackets even if not needed

```cpp
int m = 5;
int n = 10;
if (m >= n)
    ++m;
    ++n;
cout << "m=" << m << "n=" << n << endl;
```

Increment of n happens outside the if-then action, which only increments m
Conditional expression

• Format: cond ? ifExpr : elseExpr

• Earlier examples rewritten
  • \( b = (a > 0)? a : -a; \)
  • \( c = (a < b)? a : b; \)

• If in doubt, use (parens) to make sure expression tree is correct

• Use sparingly, to avoid errors

• Nesting quickly gives unreadable code:
  \( (a > 0)? a : ((-a < b)? 100+c : c-100) \)
More notes

- See the cpp-tutorial posted on Moodle
  - Cpp resources folder

- See any textbook on C++
  - Ranade's book, softcopy posted on Moodle
  - Cohoon

- Many websites have good C++ tutorials
  - www.cplusplus.com