CS 101: Computer Programming and Utilization

08-C++ control flow statements

Instructor: Sridhar Iyer
IIT Bombay

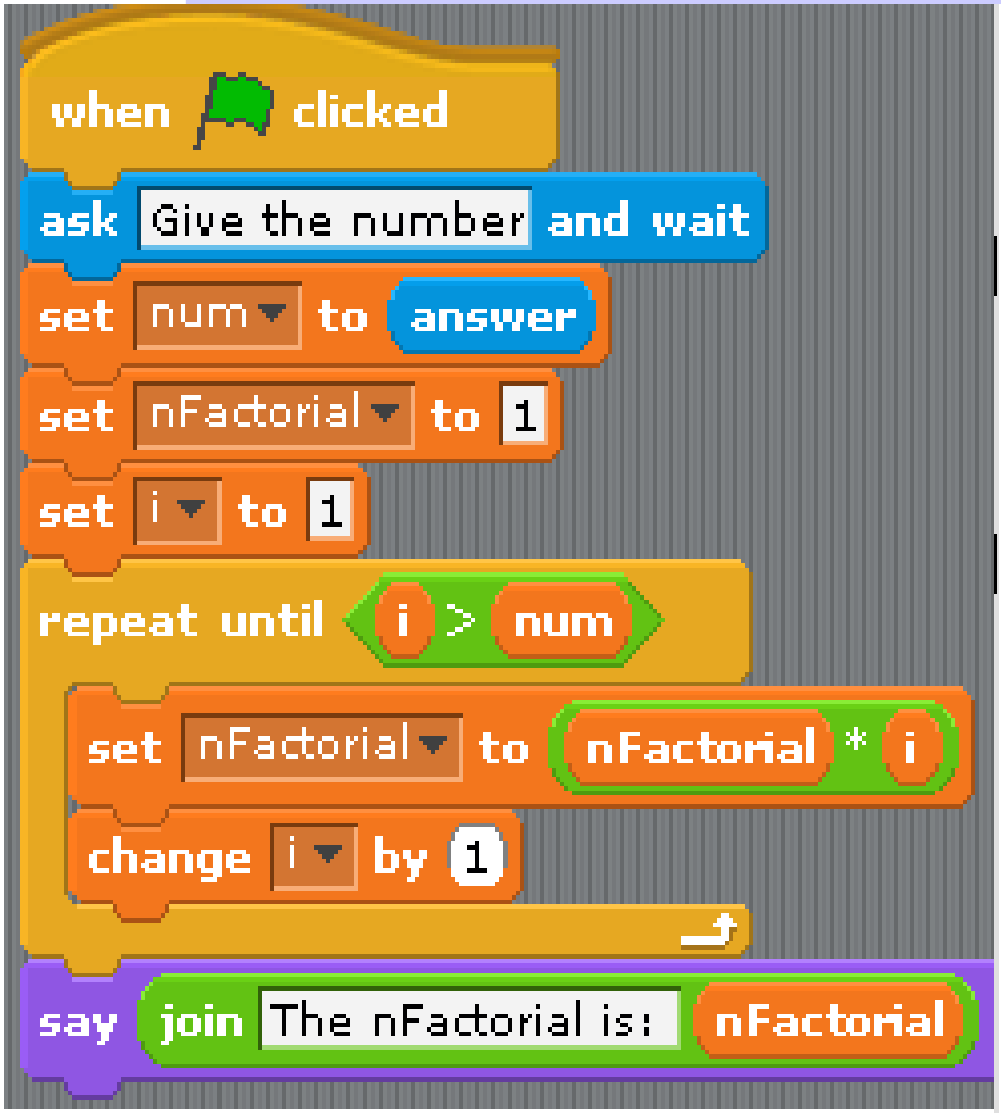
Activity: Write a program

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




```
#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, expresions: 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;`

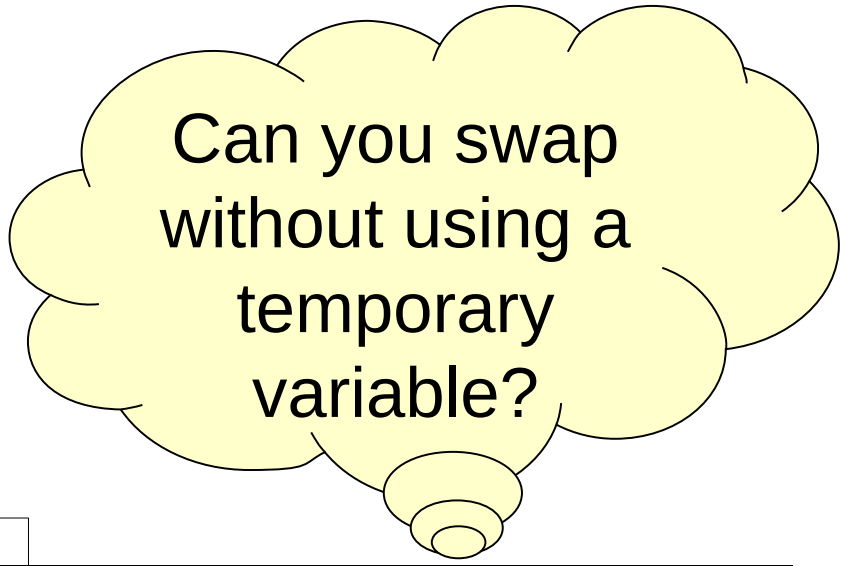
- Assignment statement has value = rhs
  - Lets us cascade `x = y = z+1;`

Logical expressions – Similar to Operator block in Scratch

- Compare numeric expressions
  - `5 < 13, 1e5 >= 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) || !(e <= f)`
- Operator precedence like with numbers
- Logical expressions used to control the execution of statements

Think-Pair-Share: swapping two numbers

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



Hand Execution

Or

Single Stepping

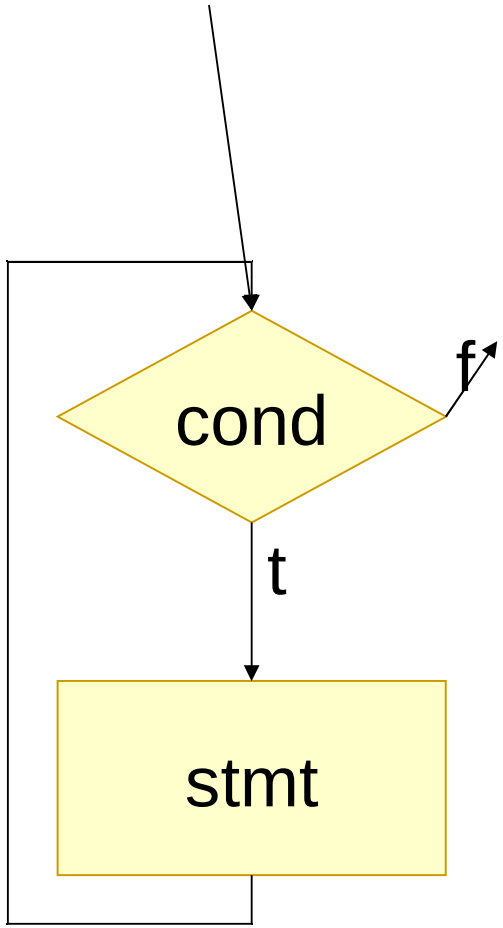
x	y	temporary
5	11	
5	11	5
11	11	5
11	5	5

More C++ constructs seen today

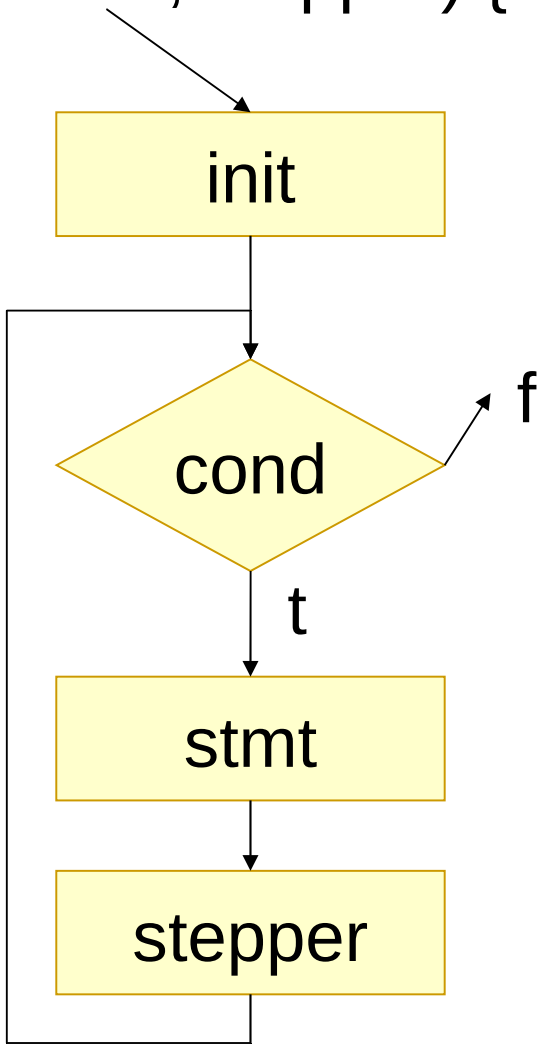
- Compound assignment: nFactorial *= i;
- Increment/Decrement: i++ (different from ++i)
- Condition blocks: if (flag != 'y') { … };
- Conditional expressions: c = (a < b)? a : b;
    - If (a < b) c = a; else c = b;
- Loops: for (i=0; i <= n; i++) { … };
- Nested loops: while () { ... for () {...}; ...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

Prolog

Loop body

Epilog

```
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_{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_{dec}), 00000010 (=2_{dec}), 00000011 (=3_{dec}), 00000100 (=4_{dec}), 00000101 (=5_{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 (97_{dec}) means 'a', 98_{dec}= 'b', 1000001 (65_{dec}) = 'A', 66_{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 0.314×10^{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

	Costs how many bits to store	Magnitude of maximum value	Magnitude of minimum value
`float`	32	3.4×10^{38}	1.4×10^{-45}
`double`	64	1.798×10^{308}	4.9×10^{-324}

- 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)

x	y	x && y
0	0	0
0	1	0
1	0	0
1	1	1

- 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 ...

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

Input x	Output !x
0	1
1	0

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

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

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

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);
```

“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

```
{ 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

```
int a, b;
cin >> a;
if (a >= 0) {
  b = a;
}
else {
  b = -a;
}
```

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

```
int a, b, c;
cin >> a >> b;
if (a < b) {
  c = a;
}
else {
  c = b;
}
```

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

Example: withdrawing money from bank

```
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

```
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