An Introduction to Programming

through C++

with

Manoj Prabhakaran

Lecture 3

Conditional Loops

Loops and conditions meet

Based on material developed by Prof. Abhiram G. Ranade

So far

- Control flow: sequential, `repeat` loops, `if-else` conditions
- Variables, types (`int`, `char`, `bool`), operators, expressions
- Assignment

Today

- Conditional Loops
  - `while` loops and `for` loops
- More operators and expressions

Chapter 7 in the textbook

Recall

Drive the Turtle Around

```
bool done = false;
cout << "Enter f for forward, r for right, l for left, q to quit: ";
repeat(100) {
    if (!done) {
            char input;
            cin >> input;
            if (input == 'f' || input == 'F')  forward(100);
            else if (input == 'r' || input == 'R') right(90);
            else if (input == 'l' || input == 'L') left(90);
            else if (input == 'q' || input == 'Q') done = true;
            else cout << "Invalid input. Ignoring." << endl;
        }
}
```

a priori limit on number of iterations

even after being done, condition checks are repeated

Drive the Turtle Around

```
bool done = false;
cout << "Enter f for forward, r for right, l for left, q to quit: ";
while (!done) {
    char input;
    cin >> input;
    if (input == 'f' || input == 'F')  forward(100);
    else if (input == 'r' || input == 'R') right(90);
    else if (input == 'l' || input == 'L') left(90);
    else if (input == 'q' || input == 'Q') done = true;
    else cout << "Invalid input. Ignoring." << endl;
}
```

`while` Statement

- Syntax:

  `while (`*condition*`) {` *body* `}`

- Semantics: Repeat the following until finished: Evaluate condition; if condition holds, execute body, else finish.
- Some examples:

  same as `(i = i + 2)`

- `int i = 0; while(i<10){ cout << (i+=2) << endl; }`
- `int i = 0; while(false){ cout << (i+=2) << endl; }`
  - What if we replace `false` with `true`? `// bad idea!`

`while` Example: Number of Digits

```
unsigned int num; cout >> "Enter a non-negative integer: "; cin >> num;
// find smallest d>0 s.t. num < 10^d
int d = 1; // note: even for num==0, we need one digit to write it
int tpd = 10;  // stands for "ten power d", tpd = 10^d
while (num >= tpd) {
        d += 1;
        tpd *= 10;
}
// when we exit the loop, num < 10^d, and this is the smallest such d
cout << num << " has " << d << " digits " << endl;
```

Can also be written as ++d;

same as `tpd = tpd * 10;`

`while` Example: Reading in a Number

```
cin >> noskipws; // don't skip whitespace (we will do it ourselves)
char c;
bool gotDigit = false;
while(!gotDigit){       // ignore non-digit characters
    cin >> c;
    if (c >= '0' && c <= '9') // chars can be compared. '0',...,'9' in that order.
        gotDigit = true;
} // c has a digit now
unsigned int n = 0;
bool done = false;
while(!done) {
    n = n*10 + (c-'0');
    cin >> c;
    if (c < '0' || c > '9')
        done = true;
} // c has a non-digit now
cout << "Read number " << n << endl;
```

Compiler automatically converts this to an integer-valued expression

`while` Example: Reading in a Number

```
cin >> noskipws; // don't skip whitespace (we will do it ourselves)
char c;
bool gotDigit = false;
do {
    cin >> c;
    if (c >= '0' && c <= '9') // chars can be compared. '0',...,'9' in that order.
        gotDigit = true;
} while(!gotDigit)
unsigned int n = 0;
bool done = false;
do {
    n = n*10 + (c-'0');
    cin >> c;
    if (c < '0' || c > '9')
        done = true;
}while(!done)
cout << "Read number " << n << endl;
```

`while` and `do-while` Statements

- Syntax:

  `while (`*condition*`) {` *body* `}`
- Semantics: Repeat the following until finished:

```
evaluate condition;
if condition holds,
    execute body,
else finish.
```

- Syntax:

  `do {` *body* `} while (`*condition*`)`
- Semantics: Equivalent to

  `{` *body* `}`

  `while (`*condition*`) {` *body* `}`
- Compared to while, can avoid one condition evaluation, if it anyway holds at the beginning
- Less commonly used structure

`break` to Exit a Loop

```
cin >> noskipws;
char c;
~~bool gotDigit = false;~~
while(~~!gotDigit~~) {
   cin >> c;
   if (c >= '0' && c <= '9')
      ~~gotDigit = true;~~
} // c has a digit now
unsigned int n = 0;
~~bool done = false;~~
while(~~!done~~) {
   n = n*10 + (c-'0');
   cin >> c;
   if (c < '0' || c > '9')
      ~~done = true;~~
} // c has a non-digit now
cout << "Read number " << n << endl;
```

equivalent

```
cin >> noskipws;
char c;

while(true) {
   cin >> c;
   if (c >= '0' && c <= '9')
      break;
} // c has a digit now
unsigned int n = 0;

while(true) {
   n = n*10 + (c-'0');
   cin >> c;
   if (c < '0' || c > '9')
      break;
} // c has a non-digit now
cout << "Read number " << n << endl;
```

`break` Statement

Demo

- Syntax: `break;` is a full statement by itself
- Semantics: Immediately exit the loop in which the statement appears.
- Example:

```
int i = 0;
while (true) {
   cout << ++i << endl;
   if (i>4) break;
}
```

```
int i = 0;
while (true) {
   cout << i++ << endl;
   if (i>4) break;
}
```

Expression has the value before incrementing

`break` Example: Smallest Prime Factor

```
int x, d;
cin >> x;
if (x == 1 || x == -1) {
  cout << x << " has no prime factors!" << endl;
} else {
    d = 2;
    while(true) {
      if (x%d == 0)
          break;
      ++d;
    }
    cout << "Smallest prime factor of " << x << " is " << d << endl;
}
```

`break` and `continue` Example: Factorisation

```
while(true) {
    cout << "Enter a non-negative number (0 to exit) : ";
    unsigned int x; cin >> x;
    if (x == 0) break;  // exiting the loop
    if (x == 1) {
        cout << "1 has no prime factors!" << endl;
        continue;  // done with this iteration! continue on to the next.
    }
    int d = 2;  // next factor to be checked. start with 2.
    while ( x > 1) { // as long as x has a prime factor left
        while(x%d == 0) { // find and remove all factors of d from x
            x /= d;
            cout << d << " ";
        } // no more factors of d
        ++d; // try next number
    }
}
```

`break` and `continue` Statements

Demo

- Syntax: `break;` is a full statement by itself
- Semantics: Immediately exit the loop in which the statement appears.
-
```
int i = 0;
while (i<10) {
  if (++i == 4) break;
  cout << i << endl;
}
```

- Syntax: `continue;` is a full statement by itself
- Semantics: Immediately finish the current iteration of the loop, and proceed to the next iteration.
-
```
int i = 0;
while (i<10) {
  if (++i == 4) continue;
  cout << i << endl;
}
```

for Loop

initialisation

condition

update

```
int d = 2;
while ( x > 1) {
    while(x%d == 0) {
        x /= d;
        cout << d << " ";
    }
    ++d;
}
```

```
for (int d = 2; x > 1; ++d) {
    while(x%d == 0) {
        x /= d;
        cout << d << " ";
    }
}
```

`for` Statement

- Syntax: `for (`*initialisation* `;` *condition* `;` *update*`) {` *body* `}`
- Here *initialisation*, *condition* and *update* are expressions (e.g., `i=0`, `i==0`, and `++i`). *initialisation* can also be a variable declaration (e.g., `int i = 0`). All allowed to be empty.
- Semantics:

```
               initialisation;
               while (condition) {
                 body
continue_from:   update;
               }
```

if empty, taken as `true`

A continue statement in the body will bring control here (rather than the end of the loop)

`for` Examples

`repeat(n) {...}` can now on be replaced with

```
for (int it = 0; it < n; ++it) { ... }
```

Beware of off-by-one errors

```
for (int it = n; it > 0; --it) { ... }
```

```
for ( ; ; ) { ... }
```

is a "forever" loop (just like while(true)).
Needs a break statement to come out of it

`for` Example: Prime Factorisation

```
const string prompt = "Enter a non-negative number (0 to exit) : ";
unsigned int x;
for(cout << prompt, cin >> x; x!=0; cout << prompt, cin >> x ) {
  cout << "Prime factors of " << x << ": ";
  if (x == 1)
    cout << "1 has no prime factors!" << endl;
  else { // for each d, find and remove all factors of d from x
    for (int d=2; x > 1; ++d)
      for( ; x%d == 0; x /= d)
        cout << d << " ";
    cout << endl;
  }
}
```

expr1 , *expr2* evaluates both (and takes on the value of the second one)

empty is OK

`for` Example: Prime Factorisation

```
const string prompt = "Enter a non-negative number (0 to exit) : ";
unsigned int x;
for(cout << prompt, cin >> x; x!=0; cout << prompt, cin >> x ) {
   cout << "Prime factors of " << x << ": ";
   if (x == 1)
      cout << "1 has no prime factors!" << endl;
   else { // for each d, find and remove all factors of d from x
      for (int d=2; x > 1; ++d)
         for( ; x%d == 0; x /= d, cout << d << " ") ;
      cout << endl;
}
```

expr1 , *expr2* evaluates both (and takes on the value of the second one)

empty is OK

This is a valid expression!

empty body!

Valid but obscure! Use update expression for "updates" only

`for` Example

```
cin >> noskipws;
char c; unsigned int n;

for(cin >> c; c<'0' || c>'9'; cin >> c);
for(n=0; c>='0' && c<='9'; n=n*10+(c-'0'),cin >> c);

cout << "Read number " << n << endl;
```

What does this do?

for Example: Inscribed Squares

Demo

```
int nsqr = 4; // number of squares to draw
float side = 400, step = 5;
for (int k = 0; k < nsqr; ++k, side /= sqrt(2)) {
   for (int it = 1, nsteps = side/step; it <= nsteps*4; ++it) {
      // count from 1: no turn at the beginning, turn at the end
      if (it % 2 == 0) penDown(); else penUp();
      forward(side/nsteps);  // approximately step long
      if (it % nsteps == 0) right(90);
   }
   // prepare for the next inner square
   penUp(); forward(side/2); right(45); penDown();
}
```

Summary

- `while`, `do-while` and `for` loops
  - `for` is the most expressive, but can be hard to understand when used in non-standard ways (especially if comments are missing)
- `break` and `continue` can help with simplifying the code
  - But try to avoid them: You may be able to include the `break` logic in the loop's condition and use if-else chains to avoid `continue`.
- `x += y`, `x *= y`, etc. Increment/decrement: `++x`, `--x` (i.e., `x+=1`, `x-=1`).
  - Also, `x++` and `x--`: Expression has the value prior to the increment/decrement
- Comma operator. Expressions involving `cout`, `cin`.
- More types: `unsigned int`, `string`, `float`. Arithmetic for `char`. `const` qualifier.

(Re)write using while and for loops

Exercises

Interpret collision as coming close to each other

- Drive the turtle around, but if it tries to go outside the 500x500 box, refuse the command and print an error message
  - Hint: Keep track of the x and y coordinates of the turtle

- Simulate two turtles in the same window, accepting commands for them alternately (turtle pointer shows the turtle about to move)
  - Keep track of two (simulated) turtles' positions, and which one is the actual turtle. When switching turtles, penUp and move from one to the other.

- While simulating two turtles, refuse moves resulting in a collision
  - Can you further allow moves of arbitrary step sizes and angles, via commands of the form "r 45", "f 30" etc.? Note that now collisions can occur in the middle of a move, and you should refuse such moves too.
  - use `float` type for real numbers