C++ Programming for Engineers

Petar Jervic
Francisco Luro
Andrew Moss (course-resp)

May 2, 2012

Loops

Common sight in 1980s computer stores

```
10: print "Andrew was here"
20: goto 10
```

We can understand this by comparing it to assembly language

- The `print` would be some complex sequence of instructions
- The `goto` is an unconditional jump

Loops in C++

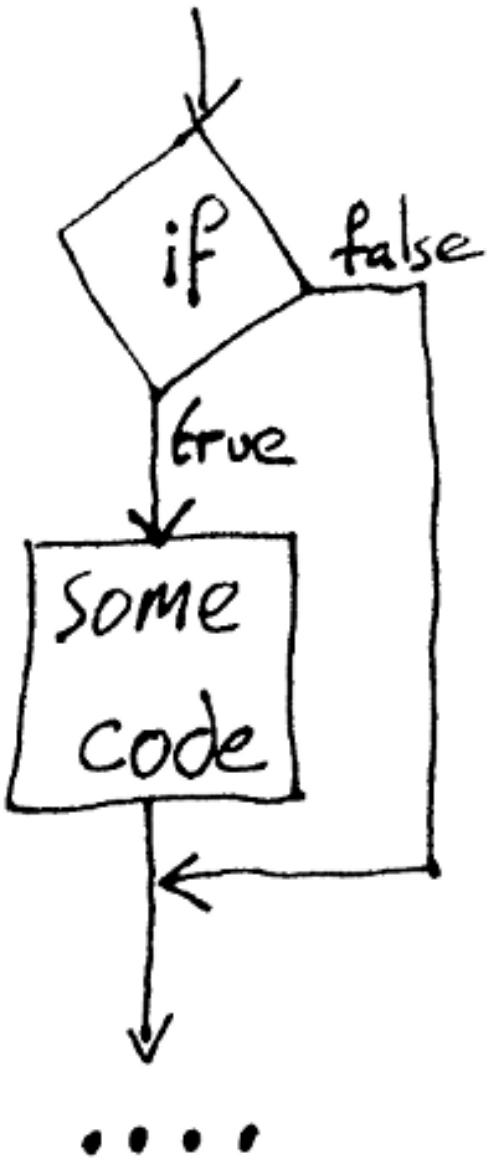
Let's convert the program into something that we recognise

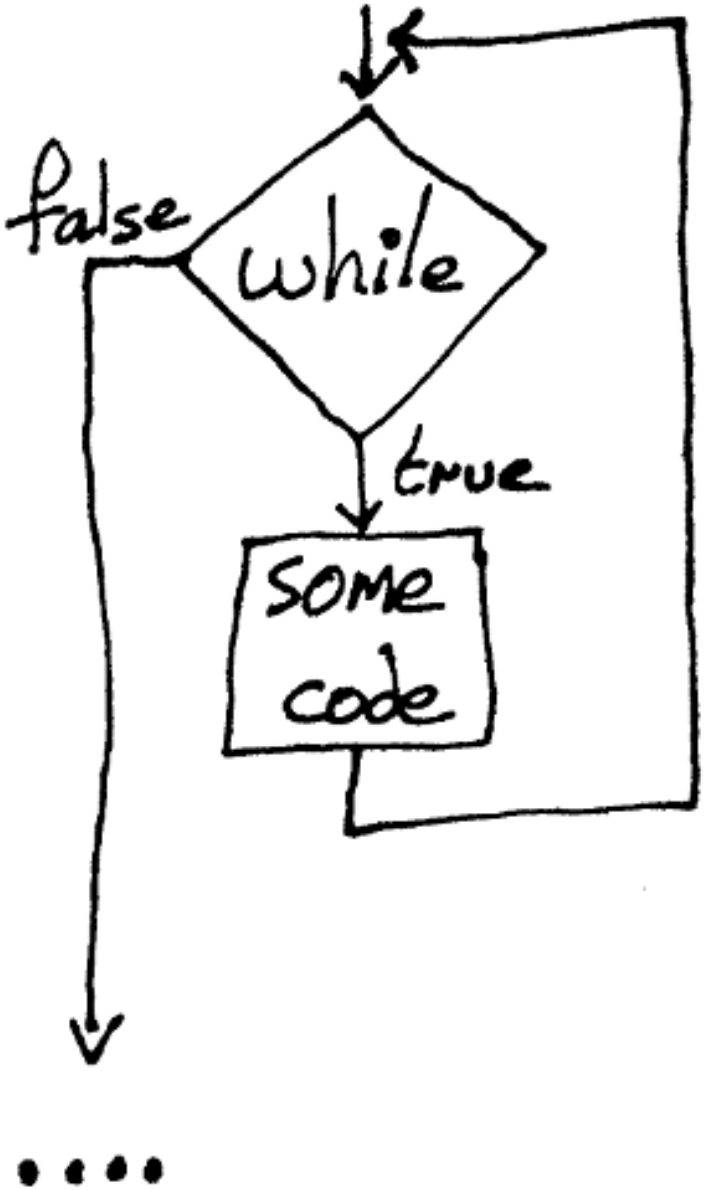
```
while(true)
  cout << "Andrew was here" << endl;
```

The `while` statement is the first loop we see in C++

- It takes a condition in brackets
  - Similiar to `if` statments
- The difference is what happens after the statement inside is executed
  - The `if` statement finished
  - The `while` statement repeats

Comparison of control-flow





Infinite loops

```
while(true)
  ...
```

The condition is constant

- So every evaluation is the same (`true`)
- After evaluating the condition it follows the true branch
- After executing the statement it starts again

Why would we want to loop forever?

- An ATM shouldn't handle a fixed number of transactions
- A word processor shouldn't fix the number of edits
- A telephone exchange should keep processing calls
- In general; the outermost loop in a program may be infinite

Non-constant conditions

If a condition is not constant

- it will change depending on the state
- it can make a decision about finishing or continuing

```
char key = ' ';
while(key != 'q')
{
  cout << "Enter command (q to quit)" << endl;
  cin >> key;
  ...
}
```

Counters

Commonly we need to repeat something a fixed number of times

- Think about the observable effect of a program
- An output statement makes something happen that we can see
- If we want to see ten outputs the same, either
  - we write ten identical output statements
  - or we write one statement inside a loop that run ten times

```
int counter = 0;
while(counter < 10)
{
  cout << "Hello world" << endl;
  counter++;
}
```

Non-uniform loop bodies

The statement inside the loop is called the *body*

- If we use braces { } then the body can be several statements
- The effect of a statement depends on the state
- Different iterations of the same body can produce different results

```
int counter = 0;
while(counter < 10)
{
  cout << "Hello world" << endl;
  counter++;
}
```

Changing the effect of the body

```
int counter = 0;
while(counter < 10)
{
  cout << "Hello number " << counter << endl;
  counter++;
}
```

Produces output that looks like

```
Hello number 0
Hello number 1
Hello number 2
...
```

Nesting loops

The statement(s) inside a loop body can include a loop

- This is called nesting
- Two separate counters (`i`, `j`, ... are common names)

```
int i = 0;
while(i < 10)
{
  int j=0;
  while( j<i )
  {
    cout << "Hello ";
    j++;
  }
  cout << endl;
  i++;
}
```

Break

Consider what output the final example produces

Output

```
Hello
Hello Hello
Hello Hello Hello
Hello Hello Hello Hello
Hello Hello Hello Hello Hello
Hello Hello Hello Hello Hello Hello
...
```

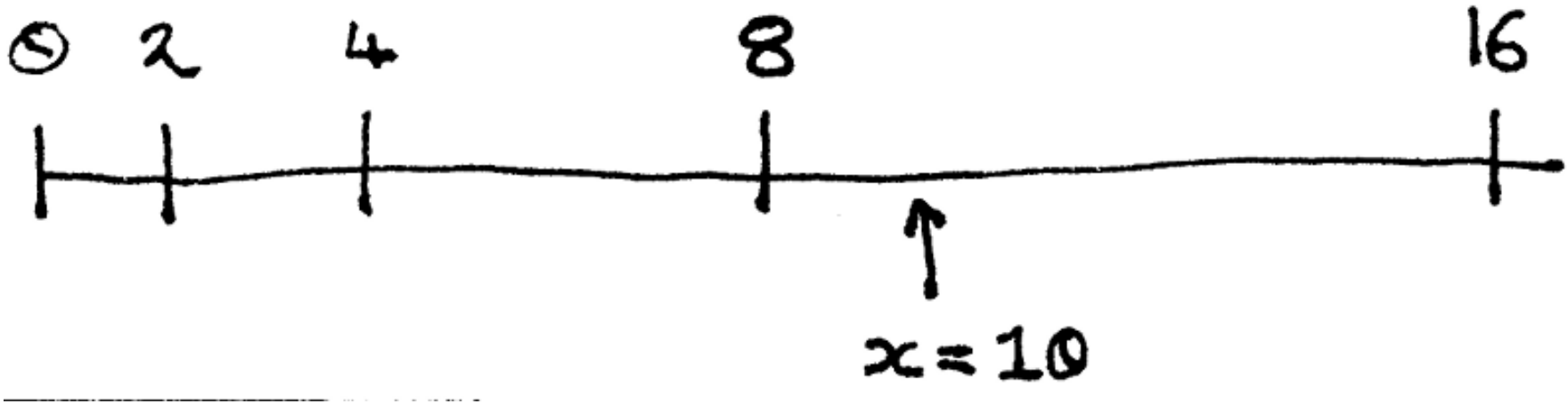
Hard version of rounding

We've looked at several kind of rounding problem

- Rounding down, Rounding up, Rounding to nearest...
- In each case the interval has been regular

What about rounding up to a power of two?

- Each interval is a different size, makes it tricky
- Either we can transform the problem to make the intervals regular...
- Or we search until we find the answer



First approach

Increase `x` until it is a power of two

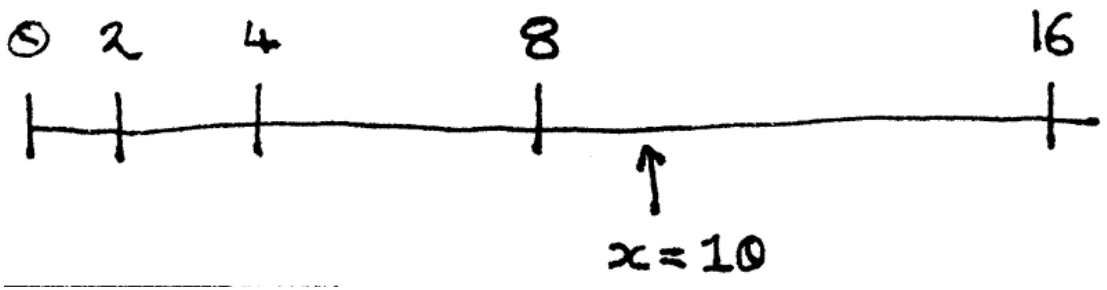
- Condition is quite complex
- You've seen how to calculate Hamming Weights in the exercises
- Later in the course we will define our own functions allowing

```
...
int x;
while( hw(x)!=1 )
  x++;
cout << x << endl;
```

Which is a very neat solution...

- But we are not there yet, we need another approach

Second approach



Let's consider the binary expansion when $x = 10$

- `00001010`
- The highest `1` in the expansion is in position 3
- So we know that $2^3 \leq x \leq 2^4 \ldots$
- Gives a solution for rounding down `00001000` (8)

For all the numbers in that range

- 8 rounds up to 8, but 9, 10, 11 . . . 16 all round up to 16
- So we have a binary choice

Second approach II

Lets write some psuedo code for what we have so far

```
t = the highest one in the binary expansion of x
if( x==2^t )
  result is x
else
  result is 2^(t+1)
```

Reduced the first problem to a part we need to discover

- How to find the highest one

And a part that we know how to do

- Split the cases into two using an if-statement

Finding the highest one

How many times can we (integer) divide `x` by two?

- After the final time it will be zero
- Destructive operation (it changes the value)
- So we do it on a copy of the data
- Using top as a counter, it records the number of divisions

```
int copy = x, top = -1;
while(copy>0)
{
  top++;
  copy /= 2;
}
```

Solution

Some details left, like why `1<<top` = 2^t

- What is the result if `x` is zero to begin?

```
int result, copy = x, top = -1;
while(copy>0)
{
  top++;
  copy /= 2;
}
if( x==(1<<top) )
  result = x;
else
  result = 1 << top+1;
```

Finish on a puzzle

```
#include<iostream>
#include<math.h>
using namespace std;
float pi=3.1415;
int main(int argc, char **argv) {
int y = -10;
  while(y<10) {
    int x=-10;
    while(x<10) {
      if( sin(pi*x/10.0) >= pi*y/20.0 )
        cout << '*';
      else
        cout << '.';
      x++;
    }
    cout << endl;
    y++; }}
```