Programming seminar 2015
Session 3: "Loopy" C/C++

Massimo Cavallaro

School of Mathematical Sciences
Queen Mary, University of London

4 March 2015

Outline

Outline

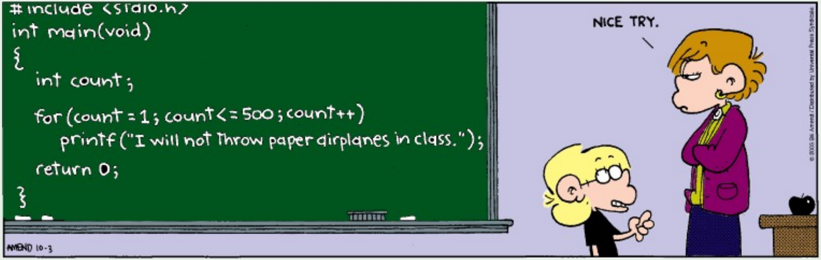
Motivation

Problem

Computers are very good at performing repetitive tasks. In this section we learn how to exploit them in real-life situations.

Example

Write a sentence on the blackboard 500 times



"For" Loops

Typical use

`for` loops run while a certain condition is true, but allow separate statement(s) to be run (1) before the first iteration and (2) after each iteration. The archetypal use for the `for` loop is to loop some pre-determined number of times indexed by a counter variable:

```
int count;
for(count=1; count<=500; count++){
    cout<<"I will not throw paper ↩
        airplanes in class"<<endl;
}
```

Variable counter is referred to as an "index". See **forloop0.cpp** for a complete program to play with.

"For" Loops

More generally

```
for(Statement1; Condition; ↩
    Statement2)
{/* Code block here is repeated ↩
    while Condition is true.*/}
```

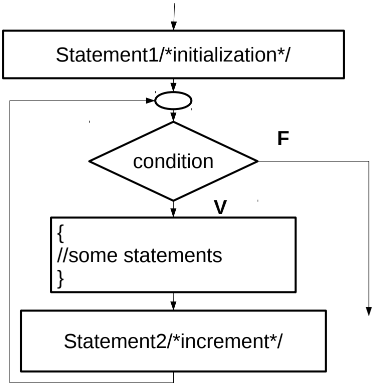
- Execute Statement1 (only the first time)
- Evaluate Condition
  - If Condition is true:
    - Execute the block
    - Execute Statement2
    - Back to the Condition
  - If Condition is false, jump directly at the end of the block

"Condition" may be any boolean expression. "Condition", "Statement1", "Statement2" can be empty (but the separators ';' are necessary).

"For" Loops

Flow chart

```
for(Statement1; Condition; ↩
    Statement2){ //Statements
```



Good programming practices

though not necessary

- Use indentation.
- Curly brackets $\{\ldots\}$ are necessary only if the code block has more than one statement...
- ...but I suggest to use them in any case.
- Don't declare more than one variable in Statement1.
- Use short name for indexes, e.g., *i*.

Examples

Print the first n numbers of the Fibonacci sequence.

```
int fib1=0;
int fib2=1;
int fib=fib1+fib2;
for (int i=0; i<=n; i++){
    cout<<i<<" "<<fib<<endl;
    fib=fib1+fib2;
    fib1=fib2;
    fib2=fib;
}
```

Exercise

write your own program!

Examples

Compute and print $n!$.

```
int n, factorial;
cout<<"Enter an integer: ";
cin>>n;
factorial=n;//init
for(int i=n-1;i>1;i--)
  factorial *= i;
cout<<"The factorial of "<<n<<" is↩
    "<<factorial<<"."<<endl;
```

Exercise

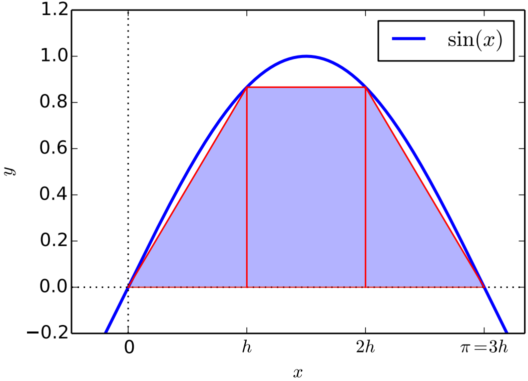
write your own program!

Example

Numerical solution of a definite integral: theory

The trapezium rule is a method for approximating an integral

$$\int_{a=0}^{b=\pi} sin(x)dx \approx \pi/3[sin(\pi/3) - sin(0)]/2$$
$$+ \pi/3[sin(2\pi/3) - sin(\pi/3)]/2 + \pi/3[sin(\pi) - sin(2\pi/3)]/2$$



Example

Numerical solution of a definite integral: snippet

For domain discretised into n equally spaced intervals with extremes at points $a, a+h, a+2h, \ldots, b$, after some algebraic manipulation, the approximation becomes:

$$\int_a^b f(x)dx \approx \frac{h}{2}[f(a) + 2\sum_i f(a+ih) + f(b)]$$

```
sum = sin(from) + sin(to);
for(int i = 1;i < n;i++) {
    sum += 2.0*sin(from + i * h);
}
```

Example

Numerical solution of a definite integral: snippet

Alternative: we can approximate the integral with the sum of rectangles:

$$\int_a^b f(x)dx \approx h \sum_{i=0}^{(b-a)/h} f(a+h*i)$$

```
sum = sin(from) + sin(to);
for(int i = 1;i < n;i++) {
    sum += 2.0*sin(from + i * h);
}
```

Play with **forloopintegral.cpp**.

Outline

"While" loops

Purely conditioned loops

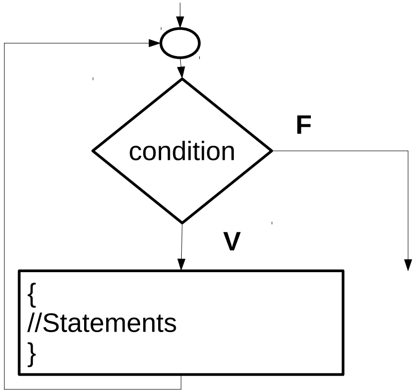
A `while` statement runs a block of code as long as a certain condition is true:

```
while(condition){
  /* Code here is repeated while
     condition is true (and will
     never run at all if condition
     is false to start with).*/
}
```

"While" Loops

Flow chart

```
while(condition){ /*code block*/}
```



Example

Series expansion of transcendental function

- You are familiar with $exp(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$
- To approximate $exp(x)$, iterate over the terms of this series, up to a certain order.
- You may not know how many terms are needed. Therefore a `while` is preferred over the `for` loop.
- The loop can be interrupted when the term $\frac{x^n}{n!}$ is smaller than a pre-setted threshold.
- Play with **whileloopexponential.cpp**

Exercise

compute $sin(\pi)$ using its series expansion.

"Do-while" loops

Post-conditioned loops

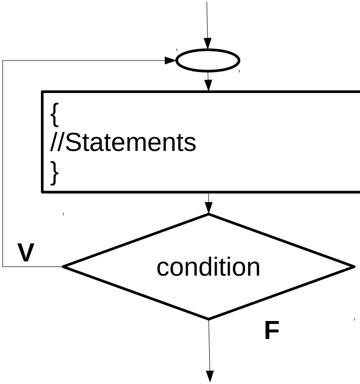
A `while` loop checks its condition even before its first iteration. Sometimes we don't want this: i.e., we want the loop always to run the first time, and only to test the condition before subsequent iterations. To do this, we use the `do–while` construct:

```
do{
  /* Code here is repeated while ↩
      condition is  true (but will ↩
      always run at least once) */
} while(condition);
```

"Do-while" Loops

Flow chart

```
do{
  //Statements
} while(condition);
```

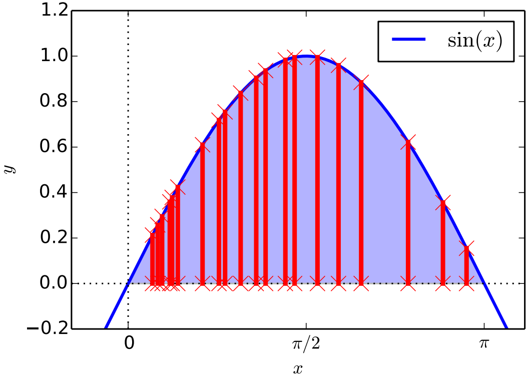


Example

Monte Carlo solution of a definite integral: theory

Strategy 1

Same idea of rectangle rules, but with random uniform sampling of n points in $[a, b]$.
$\int_0^\pi sin(x) \approx \pi/n \sum_{i=1}^n \sin(x_i)$, where $x_i \in [0, \pi]$



Example

Monte Carlo solution of a definite integral: theory

Strategy 1

Same idea of rectangle rules, but with random uniform sampling of n points in $[a, b]$.
$\int_0^\pi sin(x) \approx Q_n \equiv \pi/n \sum_{i=1}^{n} \sin(x_i)$, where $x_i \in [0, \pi]$

- i is the counter, n is number of iterations.
- We don't know a priory the best value for n.
- Iterate as long as the *relative error* $\pi Var(Q_n)/\sqrt{n}$ is above a certain threshold.

`for` or `while` loop?

Example

Monte Carlo solution of a definite integral: snippet

Strategy 1

```
confidence_level = 0.1;
do{
    x=(double)rand()/RAND_MAX*(xmax-↩
        xmin)+xmin;
    y=sin(x);
    n++;
    sum=sum+y;
    Q_n=sum/n*(xmax-xmin);
    tmp=tmp+(y-mean)*(y-mean);
    rel_error=sqrt(tmp)/n*(xmax-xmin↩
        );
}while(rel_error>confidence_level);
```

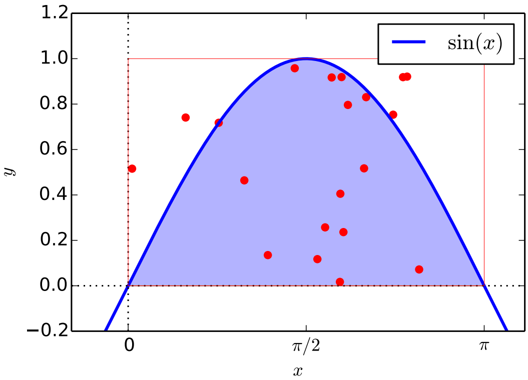
See **dowhileloopmontecarlo1.cpp**.

Example

Monte Carlo solution of a definite integral: theory

Strategy 2

Count the points that fall inside the shaded area (you need two random numbers for each sample).



Example

Monte Carlo solution of a definite integral: snippet

Strategy 2

```
confidence_level=0.01;
do{
    x=(double)rand()/RAND_MAX*(xmax-↩
        xmin)+xmin;
    y=(double)rand()/RAND_MAX;
    if (y<sin(x)) k++;
    n++;
    I=(double)k/n*(xmax-xmin);
    tmp=tmp+(1-mean)*(1-mean);
    rel_error=sqrt(tmp)/n*(xmax - ↩
        xmin);
}while(rel_error>confidence_level);
```

See **dowhileloopmontecarlo2.cpp**.

"For" or "While"?

- It depends on you own style, but...
- ..when the number of iteration is known a priori, the `for` loop is more clear.
- Exercise: implement a Monte-Carlo integration method using the `for` loop.
- Exercise: write a program for the Fibonacci series using a `while` loop.

Outline

Break the loop

The normal flow of a loop can be modified by using one of the following statements inside the loop's body:

- `break` ends a loop immediately. Control passes to the code immediately following the loop.
- `continue` ends the current iteration of a loop. The loop's condition is then evaluated again, and the loop either proceeds to its next iteration, or ends, as appropriate.
- `goto id;` unconditionally transfer the flow to the statement labeled by `id:`.

Prefer to control loops by their conditions where possible, rather than by using the `break` and `continue` statements. They will be easier to read.

Nested Loops

- Any loop can be completely nested inside another loop.
- In case of two nested `for` loops, each loop should have a different *index*.
- `continue` and `break` work only on a single "level".
- In order to exit from all the nested loops you could use `goto` ...
- ... or play with the condition, as in structured programming convention.

Exit from a nested Loops

Example 1: non-structured code

```
int i,j,v;
for(i=1,v=1; i<=8; i+=3){
    for(j=3; j<6; j++){
        cout<<i<<" "<<j<<endl;
        cin>>v;
        if(v==0){
            goto point;
        }
    }
}
point: //continue from here;
```

See **forloopnested.cpp**.

Exit from a nested Loops

Example 2: structured code

```
for(i=1, stop=false; i<=8&&stop==↩
   false; i+=3){
    for(j=3; j<6&&stop==false; j++) ↩
        {
          cout<<i<<" "<<j<<endl;
          cin>>v;
          if(v == 0){
              stop=true;
          }
    }
}
//continue from here
```

See **forloopnested.cpp**.

Outline

Exercises

- Find the area of unitary disk using Monte-Carlo strategy 2.
- Prompt the user for an angle and then display a message indicating whether its sine is positive, negative or zero.
- Play guess-the-number with the user: the computer chooses a number, and the user keeps guessing it until they get it right. The computer should report whether each guess is too large or too small.