Annoated slides from Wednesday

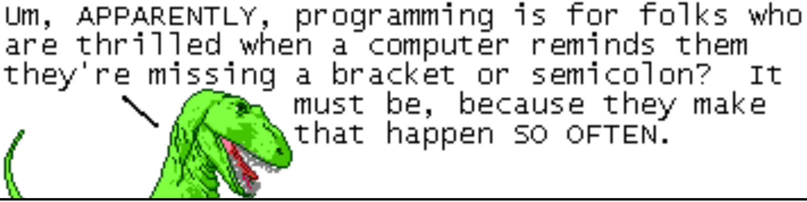
(Thursday's lecture was cancelled!).

Week 3: Loops, Input and Output

CS211: Programming and Operating Systems

Niall Madden

Wednesday and ~~Thursday~~, 29~~-30~~ Jan, 2020


Um, APPARENTLY, programming is for folks who are thrilled when a computer reminds them they're missing a bracket or semicolon? It must be, because they make that happen SO OFTEN.

Reminder

Lectures

Lecture	Wednesday	15:00–15:50	AC202
Lecture	Thursday	13:00–13:50	AC204
Lab	Friday	09:00–11:50	AdB-G021

attend for 2 hours.

Computer labs are a very important part of this course, and attendance is considered mandatory

First lab: this Friday, 1 Feb.

Total of 8 during Semester.

Reminder

1. Selection statements and loops
   - `if` statements
2. for() Loops
   - for-loop arguments
   - Recall... Algorithms
3. `while` - loops
   - `do ... while`
   - Exiting a loop
4. Why not to use `goto`
5. Output: print()
   - plain text
   - Escape Characters
   - Conversion characters
   - Other output functions
6. Input: scanf()

Selection statements and loops

To control the **flow** of a program, one uses

- **Selection Statements:** select a particular execution path. The most important is `if/if else/else` statements. See also, `switch` and, especially, `?:`
- **Iteration statements:** `for`, `while` and `do`
- **jump statements:** `break`, `continue` and `goto`

Selection statements and loops　　if statements

if statements are used to conditionally execute part of your code.

Structure:

```
if( exprn )
  {
    perform statements if exprn evaluates as
                non-zero
  }
  else
  {
    statements if exprn evaluates as 0
  }
```

"Exprn", is any expression that can evaluate as "true" or "false".

Selection statements and loops

Also, if blocks can take the form:

Structure:

```
if( A )
  {
    perform statements if expression A evaluates
                    non-zero
  }
  else if( B )
  {
    statements if A is false, but B evaluates as true
  }
  else
  {
    statements if both A and B evaluate as false
  }
```

Can be repeated many times.

A trivial example

```
#include <stdio.h>
int main(void )
{
  if (10)
  {
    printf("Non-zero is always true\n");
  }
  if (0)
  {       /* dummy line */    }
  else
    printf("But 0 is never true\n");
  return(0);
}
```

Selection statements and loops

Typically, however, the expressions that `if()` depends on are ***logical expressions***, based on **relational operators**, that must be evaluated.

- `a == 10` — means a is equal to 10 (not the same as a=10).
- `c == 'n'`
- `x != 10` not equals.
- `z < y`
- `y >= z`

set a to be 10.

Selection statements and loops

Examples

Logical operators, AND, and OR, allow more complex if-statements: && ||

```
if( ( (i%3) == 0) && ( (i%5)==0) )
  printf("%d divisible by 15\n", i);

if( ( (i%3) == 0) || ( (i%5)==0) )
   printf("%d divisible by 3 or by 5\n", i);
```

i%a is the remainder on dividing i by a.

Selection statements and loops

01EvenOdd.c ⟵ link!

```
// Check Even or Odd
int a=rand()%10; // a is a random number between 0 and 9.
printf("a=%d\n", a);
if ( (a % 2) == 0)
  printf("a is even\n");
else
  printf("a is odd\n");

// Check positive, negative or zero
a=rand()%7-3; // a is a random number between -3 and 3.
printf("a=%d\n", a);
if ( a>0 )
  printf("a is (strictly) positive\n");
else if ( a<0)
  printf("a is (strictly) negative\n");
else
  printf("a is zero\n");
```

rand() returns a (pseudo)random number between 0 and something big.

for() Loops

```
for( initial val; continuation cond; increment)
```

`for()` is an expression used to execute “loops”: groups of similar tasks to be repeated a certain number of times. It takes three arguments,

- an initial value for the increment variable.
- a condition for continuing the loop.
- instructions on how to modify the increment variable at each iteration.

The tasks to be completed within the loop are contained within curly brackets.

If `{ }` are omitted, then the loop consists only of the line immediately after the `for()` command.

for() Loops

Example (Print a line)

Sometimes we just want a simple operation repeated a fixed number of time. This example just prints a “line” across the screen

int i;

start with i=1

```
printf("\n");
for (i=1; i<=60; i++)
  printf("-");
printf("\n");
```

add 1 to i at each iteration,

Continue so long as i<=60

This is the same as

```
for (i=1; i<=61; i++)
{
  printf("-");
}
```

The indentation is only cosmetic.

for() Loops

More often, in the body of the loop we use the "***increment variable***" (== "***the loop index"***), as in the following example.

Recall that the ***Fibonacci*** sequence is defined as

$$f_0 = 1, f_1 = 1, \text{ and for } k = 2, 3, \ldots, f_k = f_{k-1} + f_{k-2}.$$

for() Loops

02Fibonacci.c

```c
#include <stdio.h>
int main(void )
{
  int i, Fib[10];
  Fib[0]=1;
  printf("Fib[0] = %d\n", Fib[0]);
  Fib[1]=1;
  printf("Fib[1] = %d\n", Fib[1]);

  for (i=2; i<=9; i++)
  {
     Fib[i] = Fib[i-1] + Fib[i-2];
     printf("Fib[%d] = %d\n", i, Fib[i]);
  }
  return(0);
}
```

There are 2 lines in the block, so { } are needed.

for() Loops

Example (Print the odd numbers from 1 to 19)

```
for(i=1; i<= 19; i+=2)
  printf("%d ",i);
```

at each step, set

$i = i + 2$.

for() Loops

Example (Count down from 10 to 0)

```
for(i=10; i >=0; i--)
  printf("%d ",i);
```

set i = i - 1.

```
for (int j=0; j<10; j++)
{

}
```

This is unofficially legal.

[j does not exist outside the loop]

for-loop arguments

The three arguments to `for` are optional, but the second one is the most important and it is bad practice to omit it.

Example (A bad example)

```
int i=2;
for (; i<10;)
{
  i++;
}
```

Recall... Algorithms

Definition

An **Algorithm** is a finite set of precise instructions for performing a computation or for solving a problem.

Here is an algorithm for finding the maximal element in a finite sequence $a_1, a_2, \ldots, a_n$

Linear Search

```
$m \longleftarrow a_1$
FOR $k = 2$ to $n$
   IF $m < a_k$
      THEN $m \longleftarrow a_k$
   END
END
RETURN $m$
```

"Pseudocode"

Example

Write a short C program that creates a list of 8 randomly chosen integers between 0 and 20, and then finds the largest one.

To solve the problem, we need to do several things:

- Create a random number. This is done using the `rand` function, which requires the `stdlib` header file.
- `rand` produces a number between 0 and 2147483647. Use modulus operator to get one between 0 and 20.
- Use a `for` loop to implement the **linear search algorithm**.
- Use a if statement to identifes a newest largest number.

for() Loops

03Largest.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int k, m, a[8];

  printf("\nThe list is: ");
  for (k=0; k<8; k++)   {
    a[k] = rand()%21;
    printf("\t%d", a[k]);
  }
  m = a[0];
  for (k=1; k<8; k++)
    if (m < a[k])
      m = a[k];

  printf("\nThe largest element is: %d\n", m);
  return(0);
```

array of random numbers (ints).

while - loops

The while loop is probably the simplest loop in C, though not quite as useful as the for loop.

while(*expression*) *statement*

Example

```
while(i < n)
  i*=2;
```

Example

```
i = rand()%100;
while(i < n)
{
  printf("i=%d. Guessing again...\n", i);
  i = rand()%100;
}
```

while - loops

These two are equivalent:

```
for (i=0; i<=10; i++)
  sum+=f[i];
```

```
i=0;
while ( i<=10 )
{
  sum+=f[i];
  i++;
}
```

while - loops

This is a trivial loop — it’s statements are never executed:

~~These two are equivalent:~~

```
while (0)
{
  // this stuff is ignored
}
```

Whereas the following as an infinite loop:

```
while(1)
{
  printf("We are going to be here a while...");
}
```

while - loops

A do loop is like a while loop, but with the condition for continuation/iteration coming at the end of the block:

```
do
{
    statements
}
while( expression );
```

This is used when we want the statements in the loop to be executed at least once.

while - loops

do ... while

04DoWhile.c

```c
#include <stdio.h>

int main(void)
{
  int a;

  do
  {
    printf("Enter an even number : ");
    scanf("%d", &a);
  } while ( a%2 != 0);

  printf("Number %d accepted.\n", a);

  return(0);
}
```

There are (rare) occasions where we might want to

- jump out of a `while`, `for` or `do` loop. This is achieved using `break`.
- skip to the next iteration of the loop, using `continue`.
- jump to another part of a program entirely, using `goto`.

`goto`

There is ***never*** a good reason to use `goto`. ***Never*** (well, hardly ever)

Exiting a loop

05BreakContinue.c

```c
#include <stdio.h>

int main(void)
{
  int a;

  for (a=0; a<=100; a++)
  {
    if (a%2 != 0)
      continue;
    printf("a=%d\n", a);

    if (a>=10)
      break;
  };

  return(0);
}
```

[Finished Here at 4pm, Wed]