ESC101N
Fundamentals of Computing

Arnab Bhattacharya
arnabb@iitk.ac.in

Indian Institute of Technology, Kanpur
http://www.iitk.ac.in/esc101/

1st semester, 2010-11

Tue, Wed, Fri 0800-0900 at L7

Loops

- Print all numbers between 1 and 100 that are divisible by 7
- Algorithm
  1. Initialize `x = 1`
  2. Test if `x` is divisible by 7
  3. If yes, output
  4. Increment `x`
  5. If `x <= 100`, go back to step 2

Loops

- Print all numbers between 1 and 100 that are divisible by 7
- Algorithm
  1. Initialize `x = 1`
  2. Test if `x` is divisible by 7
  3. If yes, output
  4. Increment `x`
  5. If `x <= 100`, go back to step 2
- Requires *loops* – instructions that are repeated a number of times
- Each time (called an iteration), some variable may change
- For a loop to stop, either of these must be specified
  - Number of times the loop runs
  - Stopping condition

while statement

```
while (condition)
{
  statements
}
```

- condition evaluates to a boolean
- The statements in the loop are executed as long as condition is *true*
- Any expression fits as condition
- Value of condition, if initially *true*, must change at some appropriate later point to *false*
  - Otherwise, infinite loop is created

while statement

```
while (condition)
{
  statements
}
```

- `condition` evaluates to a boolean
- The statements in the loop are executed as long as `condition` is *true*
- Any expression fits as `condition`
- Value of `condition`, if initially *true*, must change at some appropriate later point to *false*
  - Otherwise, infinite loop is created
- Print all numbers between 1 and 100 that are divisible by 7

```
x = 1;
while (x <= 100)
{
  if ((x % 7) == 0)
    printf(‘‘%d ’’, x);
  x++;
}
```

for statement

```
for (initialization; condition; update)
{
  statements
}
```

- condition evaluates to a boolean
- The statements in the loop are executed as long as condition is *true*
- initialization initializes variables
- update updates the condition
- Value of condition, if initially *true*, must change at some appropriate later point to *false*
  - Otherwise, infinite loop is created

for statement

```
for (initialization; condition; update)
{
  statements
}
```

- `condition` evaluates to a boolean
- The statements in the loop are executed as long as `condition` is *true*
- `initialization` initializes variables
- `update` updates the condition
- Value of `condition`, if initially *true*, must change at some appropriate later point to *false*
  - Otherwise, infinite loop is created
- Print all numbers between 1 and 100 that are divisible by 7

```
for (x = 1; x <= 100; x++)
{
  if ((x % 7) == 0)
    printf(‘‘%d ’’, x);
}
```

Equivalence of while and for statements

- while and for statements are equivalent

```
for (initialization; condition; update)
{
  statements;
}
```

translates to

```
initialization;
while (condition)
{
  statements;
  update;
}
```

and vice versa

- It is a matter of convenience and ease

Example

- Given a geometric progression with first term a and common ratio r, print the first n terms
- Inputs: a and r are real numbers while n is an integer

```
for (i = 1; i <= n; i++)
{
  x = a * pow(r, i - 1);
  printf(‘‘%f\n’’, x);
}
```

- Comment: pow(x,y) function computes x^y
  - Requires #include <math.h> and gcc -lm

Example

- Given a geometric progression with first term a and common ratio r, print the first n terms
- Inputs: a and r are real numbers while n is an integer

```
for (i = 1; i <= n; i++)
{
  x = a * pow(r, i - 1);
  printf(‘‘%f\n’’, x);
}
```

- Comment: pow(x,y) function computes x^y
  - Requires #include <math.h> and gcc -lm
- Could also have been written as

```
x = a;
for (i = 1; i <= n; i++)
{
  printf(‘‘%f\n’’, x);
  x = x * r;
}
```

Breaking out of a loop

- A loop can be exited straightaway by using a `break` statement
- Find the *first* number between 103 and 145 that is divisible by 23

```
for (x = 103; x <= 145; x++)
{
  if ((x % 23) == 0)
  {
    printf(‘‘%d ’’, x);
    break;
  }
}
```

- After the number is found, it does not make sense to continue
- `break` immediately exits the loop
- If there are multiple nested loops, `break` exits *only* the one where it resides

Breaking out of an iteration of a loop

- A particular iteration of a loop can be skipped by using a `continue` statement
- Add all numbers between 103 and 145 that are not divisible by 7

```
sum = 0;
for (x = 103; x <= 145; x++)
{
  if ((x % 7) == 0)
  {
    continue;
  }
  sum = sum + x;
}
```

- When a number is found to be divisible by 7, the *rest* of the loop should not be executed
- `continue` immediately stops the current iteration and starts the next
- If there are multiple nested loops, `continue` exits the current iteration of *only* the one where it resides