4 If-statements and while-loops; also, scanf

4.1 If statements

An if statement looks like

```
if ( condition )
{ ... statements ... }

or

if ( condition )
{ ... statements ... }
else
{ ... statements ... }
```

The curly braces are not absolutely necessary (no more are they with for-loops) but it is safest to include them.

If-statements will be used frequently in this module, but here is a simple example. Suppose that m and n are integers, where $n > 0$ (there is no good reason to consider the case $n < 0$). Remember that the C version of 'remainder mod n' differs from the 'official' version when $m < 0$. So, here is a piece of code (not, of course, a full program) to correct the difference.

```
int m,n,mmodn;
......

if ( m >= 0 )
{
  mmodn = m%n;
}
else
{
  mmodn = m%n + n;
}
```

And here is another, to get the maximum of m and n:

```
int m,n,max;

if ( m>= n)
{
  max = m;
}
else
{
  max = n;
}
```

4.2 While loops

A while loop is like this:

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

It operates in the obvious way:

```
If the condition holds, perform the statements, else the loop is finished.
If the condition holds, perform the statements, else the loop is finished.
If the condition holds, perform the statements, else the loop is finished.
If the condition holds, perform the statements, else the loop is finished.
and so on
```

Example. Here is a silly way to divide n into m, given that n is positive and m is nonnegative.

```
prompt% cat basic-divide.c
#include <stdio.h>
int main()
{
  int m,n, quotient, remainder; // quotient m/n
  m = 100; n = 13;
  quotient = 0; remainder = m;

  while ( remainder >= n )
  {
    remainder = remainder-n; quotient = quotient + 1;
  }

  printf("m %d n %d m/n %d m mod n %d\n", m,n,quotient,remainder);
}
prompt% gcc basic-divide.c
prompt% a.out
m 100 n 13 m/n 7 m mod n 9
prompt%
```

Let us follow this program step by step:

```
At the beginning, m is set to 100, n to 13,
                quotient to 0, and remainder to m.
while loop:
      remainder 100 >= 13; quotient becomes 1
      remainder 87 >= 13, quotient becomes 2
      remainder 74 >= 13, quotient becomes 3
```

```
      remainder 61 >= 13, quotient becomes 4
      remainder 48 >= 13, quotient becomes 5
      remainder 35 >= 13, quotient becomes 6
      remainder 22 >= 13, quotient becomes 7
      remainder  9 < 13, finishes.
Finishes with quotient 7 and remainder 9
```

4.3 Scanf

This part is included here because the second programming assignment will use input. In order to read a list of integers input from the keyboard (separated by spaces and newlines or a mixture of both)

```
  int x;
  while ( scanf("%d", &x ) == 1 )  // the ampersand & is vital
  { .... statements .... }
```

```
For example, to read a stream of numbers from the keyboard
and compute their sum

#include <stdio.h>
int main()
{
  int x, sum;
  sum = 0;
  while ( scanf("%d", &x) == 1 )
  { sum = sum+x; }
  printf("sum is %d\n", sum);
}

Here is a sample run

prompt% a.out
31 4
59

265
sum is 359

(What flagged the end of input?  A
CTRL-D at the start of the line.  This
single control character was not itself
displayed.  Numbers are separated by
spaces and newlines (and CTRL-D). It
```

```
doesn't matter how many blanks separate
the numbers or how many there are on
any line.)
prompt%
```

Explanations later.

4.4 Euclidean algorithm

Form a list of numbers beginning $720, 445$ and repeatedly take the remainder as shown.

```
This produces the list 720 445 275 170 105 65 40 25 15 10 5 0
Explanation:
720 mod 445 is 275
445 mod 275 is 170
275 mod 170 is 105
170 mod 105 is 65
105 mod 65 is 40
65 mod 40 is 25
40 mod 25 is 15
25 mod 15 is 10
15 mod 10 is 5
10 mod 5 is 0
```

(4.1) Lemma *This procedure yields the gcd (greatest common divisor, also called the highest common factor) of 720 and 445.*

Proof. One can show that every pair of consecutive numbers in the list has the same greatest common denominator. So the gcd of $720, 445$ is the gcd of $5, 0$ which is obviously 5. ∎

We shall use variables x, y, z so that x, y represent consecutive numbers in the list and z will be the next one further on:

720	445	275	170	105	65	40	25	15	10	5	0
x	y	z									
	x	y	z								
		x	y	z							
			x	y	z						
				x	y	z					
					x	y	z				
						x	y	z			
							x	y	z		
								x	y	z	
									x	y	z
										x	y

This leads to a C program ...

```
#include <stdio.h>
int main()
{
  int m,n,x,y,z;
  m = 720; n = 445;

  x = m; y = n;
  while ( y >  0 ) // Eventually, y will reach the
                   // value zero.  When y is zero,
                   // x is the gcd.
  {
    z = x%y; x = y; y = z;
  }
  printf("m %d n %d gcd %d\n", m,n,x);
}
```