Programming with C

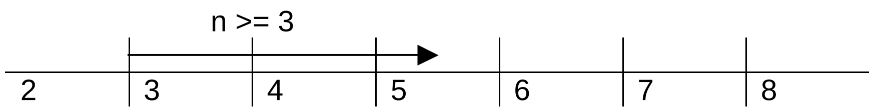
Terry Marris November 2010

7 The Logical Operators

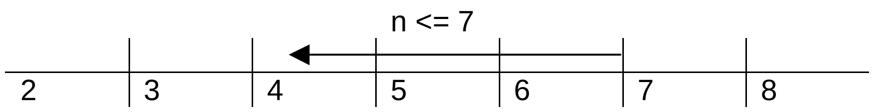
Previously we looked at simple selections. Now we look at more complex selections and the logical operators *and* and *or*.

7.1 Logical And

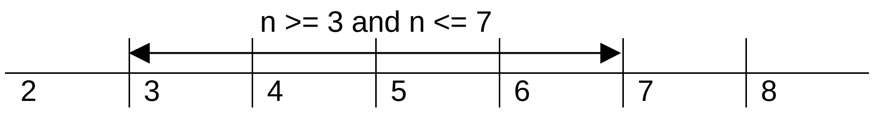
If *n* is an integer, n >= 3 specifies a range of numbers from 3 onwards,



n <= 7 looks like

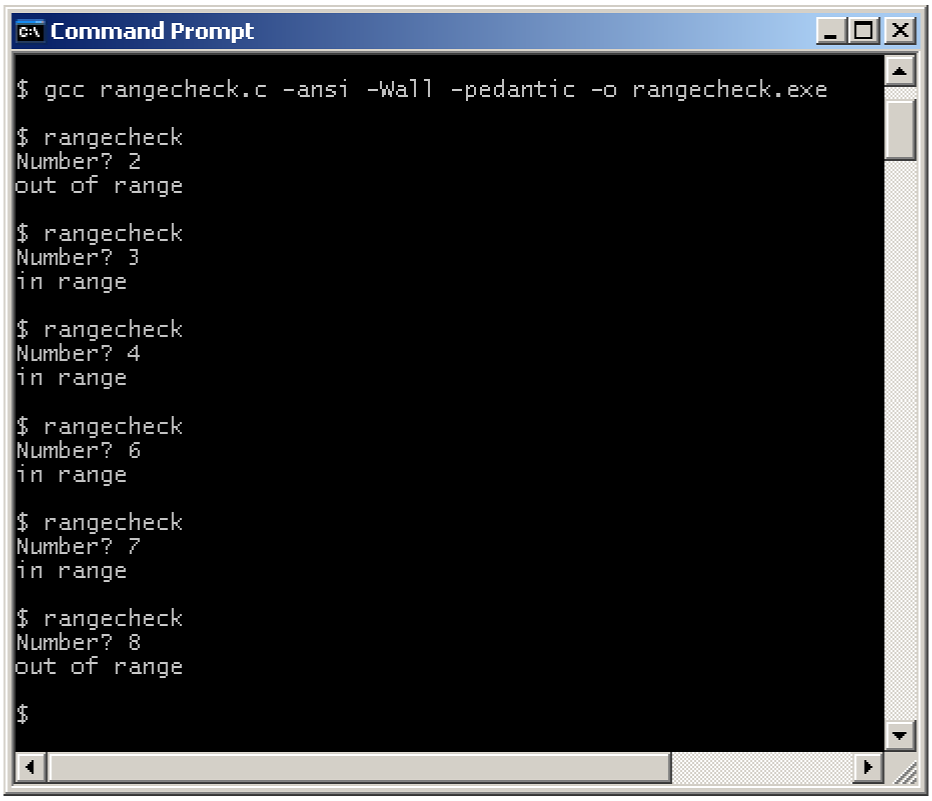


Putting them both together:



n >= 3 and n <= 7 specifies a *range*. In C we would write *n >= 3 && n <= 7*. *&&* is the logical *and* operator; it means and-at-the-same-time.

```
/* rangecheck.c: illustrates use of logical and */

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

int main()
{
  const int lower = 3;
  const int upper = 7;
  int n;
  char string[BUFSIZ];

  printf("Number? ");
  gets(string);
  n = atoi(string);
  if (n >= lower && n <= upper)
    printf("in range\n");
  else
    printf("out of range\n");
  return 0;
}
```



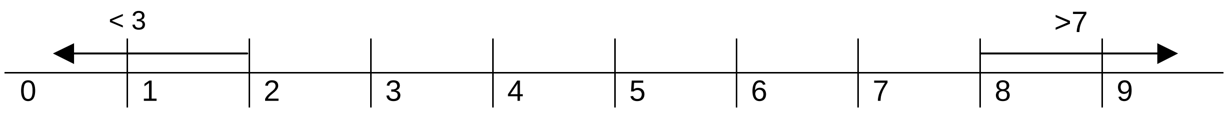
```
$ gcc rangecheck.c -ansi -Wall -pedantic -o rangecheck.exe

$ rangecheck
Number? 2
out of range

$ rangecheck
Number? 3
in range

$ rangecheck
Number? 4
in range

$ rangecheck
Number? 6
in range

$ rangecheck
Number? 7
in range

$ rangecheck
Number? 8
out of range

$
```

Now we have two boundaries defined by *n* >= ***3*** and *n* <= ***7***. So we test as close as possible to each boundary point, with values of *n* = 2, 3, and 4 for the lower boundary, and *n* = 6, 7, and 8 for the upper boundary. Why take such pains with testing? Well, history has shown that programming errors often occur around boundary points, and it is better for you to find your errors before your assessor or client does.

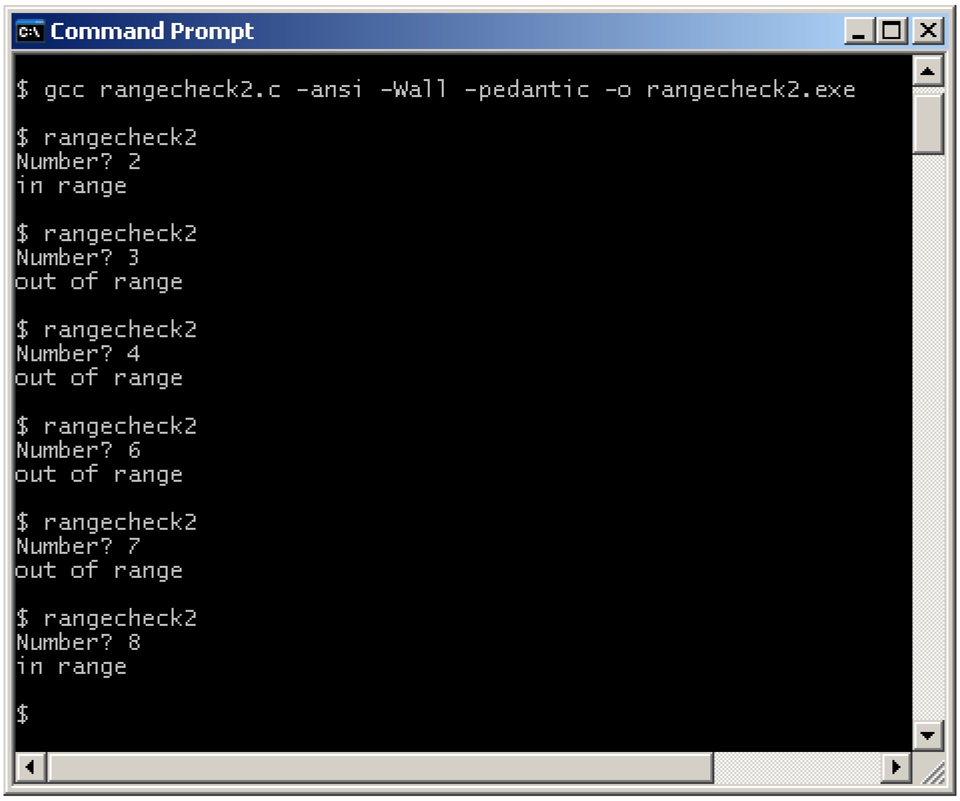
What you do with errors when you find them is up to you. You could fix them. You could acknowledge them, put them in a list of known bugs, and move on, intending to fix them at a later date. Or you could ignore them and lose marks or customers.

7.2 Logical Or

Continuing with the same problem, we can say that n < 3 *or* n > 7 defines two distinct ranges. In C we would write *n < 3* || *n > 7*. || is the logical *or* operator. The | key is found at the bottom left of the keyboard.



```
/* rangecheck2.c: illustrates use of logical or */

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

int main()
{
  const int lower = 3;
  const int upper = 7;
  int n;
  char string[BUFSIZ];

  printf("Number? ");
  gets(string);
  n = atoi(string);
  if (n < lower || n > upper)
    printf("in range\n");
  else
    printf("out of range\n");
  return 0;
}
```



```
$ gcc rangecheck2.c -ansi -Wall -pedantic -o rangecheck2.exe

$ rangecheck2
Number? 2
in range

$ rangecheck2
Number? 3
out of range

$ rangecheck2
Number? 4
out of range

$ rangecheck2
Number? 6
out of range

$ rangecheck2
Number? 7
out of range

$ rangecheck2
Number? 8
in range

$
```

We return to the body mass index problem. A person is regarded as clinically obese if their weight (in Kilograms) divided by their height (in metres) squared is 30.0 or more.

```
bmi = weight / pow(height, 2);
if (bmi >= 30.0)
  printf("clinically obese\n");
      ...
```

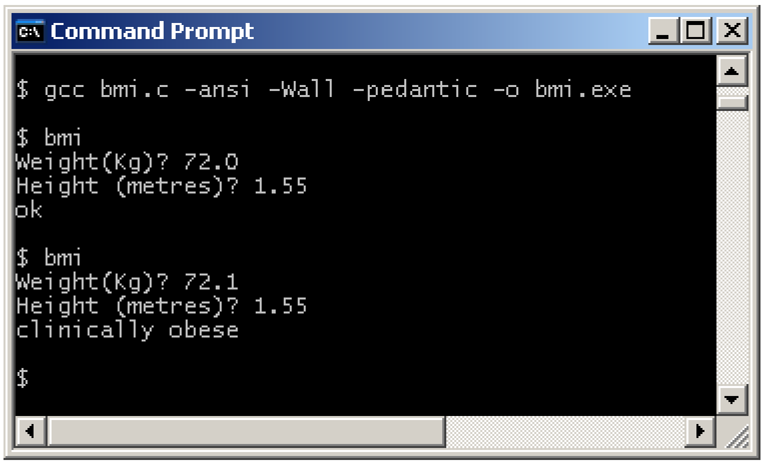
We have already dealt with the problem of determining whether two values of type *double* are identical. We have a go at dealing with the relational operator, >=.

>= means more than *or* equal to. We can split the operators with || into

```
if (bmi > 30.0 || bmi == 30.0)
  printf("clinically obese\n");
```

Here is the complete program.

```
/* bmi.c: investigates logical or with >= and doubles */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
  const double epsilon = 0.0000005;
  double weight, height, bmi, difference;
  char string[BUFSIZ];

  printf("Weight(Kg)? ");
  gets(string);
  weight = atof(string);
  printf("Height (metres)? ");
  gets(string);
  height = atof(string);

  bmi = weight / pow(height, 2);
  difference = fabs(30.0 - bmi);
  if (bmi > 30.0 || difference < epsilon)
    printf("clinically obese\n");
  else
    printf("ok\n");
  return 0;
}
```



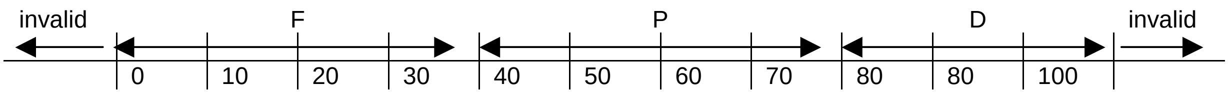
```
$ gcc bmi.c -ansi -Wall -pedantic -o bmi.exe

$ bmi
Weight(Kg)? 72.0
Height (metres)? 1.55
ok

$ bmi
Weight(Kg)? 72.1
Height (metres)? 1.55
clinically obese

$
```

The expression *bmi == 30.0* has been replaced with

```
const double epsilon = 0.0000005;
...
difference = fabs(30.0 - bmi);
if (... difference < epsilon)
  printf("clinically obese\n");
```

7.3 Else If Sequence

Grades F, P and D are awarded according to the score achieved.



Scores of less than zero or more than 100 are invalid. A score between zero and 39 inclusive is awarded an F grade. A score between 40 and 79 inclusive is awarded a P grade. A score between 80 and 100 inclusive is awarded a D grade. One way of expressing this scenario is to use a sequence of else if's.

```
if (score < 0)
  printf("invalid");
else if (score < 40)
  printf("F");
else if (score < 80)
  printf("P");
else if (score <= 100)
  printf("D");
else
  printf("cheated");
...
```

A sequence of *else if*'s is easy to understand. You look down the sequence of Boolean expressions, find the first one that is true, execute its statement block, and then skip to beyond the end of the sequence. For example, suppose the *grade* was 79.

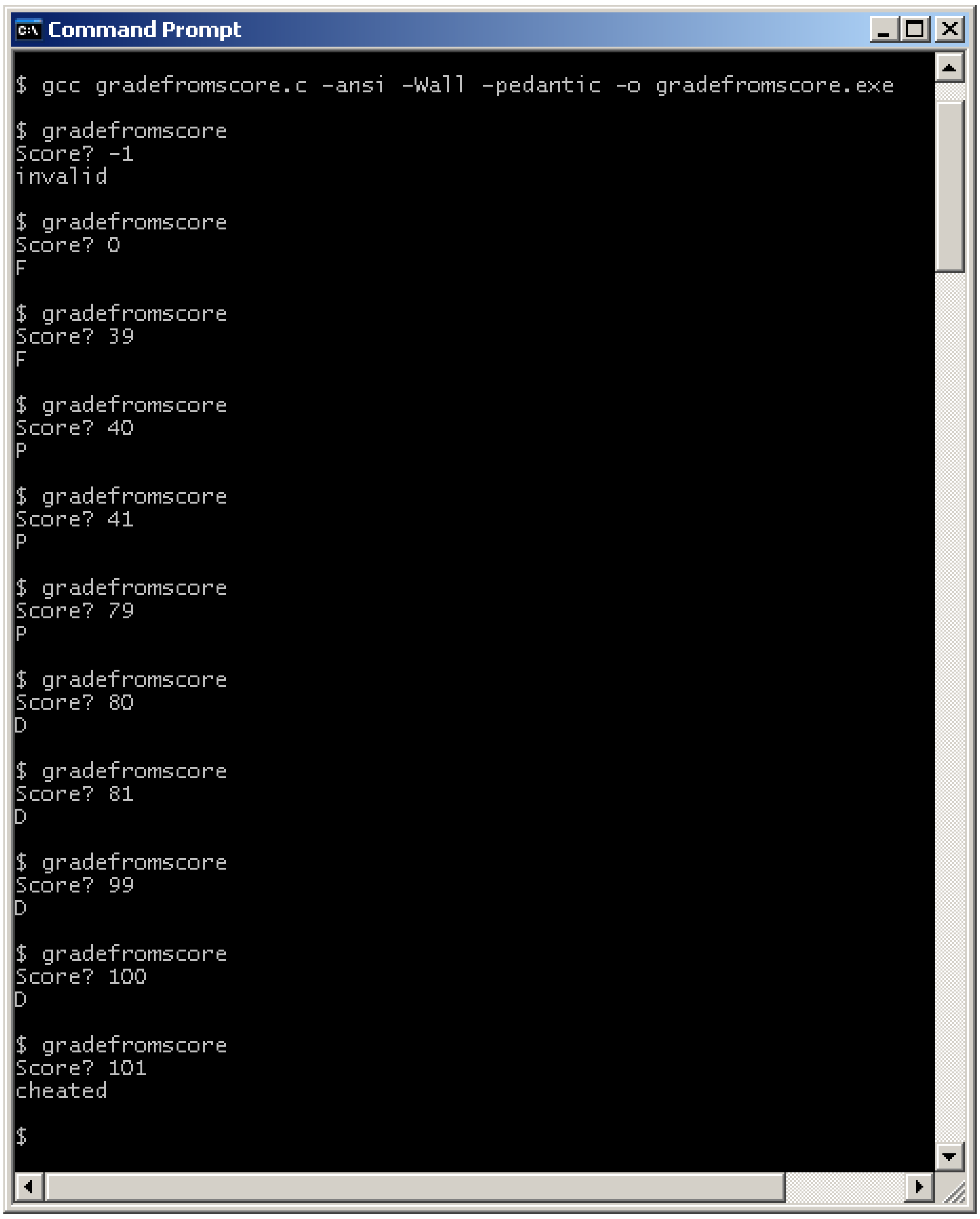
Is *score* < 0? No.
Is *score* < 40? No.
Is *score* < 80? Yes. So print *P* and skip to **...**

The important thing here is, when programming, to get the sequence of Boolean expressions in the right order.

The program, shown below, inputs a score and outputs the corresponding grade.

```
/* gradefromscore.c: converts a score into a grade */

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

int main()
{
  char string[BUFSIZ];
  int score;

  printf("Score? ");
  gets(string);
  score = atoi(string);
```

```
    if (score < 0)
      printf("invalid\n");
    else if (score < 40)
      printf("F\n");
    else if (score < 80)
      printf("P\n");
    else if (score <= 100)
      printf("D\n");
    else
      printf("cheated\n");
    return 0;
  }
```



```
$ gcc gradefromscore.c -ansi -Wall -pedantic -o gradefromscore.exe

$ gradefromscore
Score? -1
invalid

$ gradefromscore
Score? 0
F

$ gradefromscore
Score? 39
F

$ gradefromscore
Score? 40
P

$ gradefromscore
Score? 41
P

$ gradefromscore
Score? 79
P

$ gradefromscore
Score? 80
D

$ gradefromscore
Score? 81
D

$ gradefromscore
Score? 99
D

$ gradefromscore
Score? 100
D

$ gradefromscore
Score? 101
cheated

$
```

Notice that the closest values to each boundary have been used to test the program.

Exercise 7.1

1. Students rate the quality of the chips supplied by their refectory on a scale from 0 (appalling) up to 5 (brilliant). Write and test a program that inputs a student's rating, and outputs *ok* if the integer input is between 0 and 5 inclusive, and outputs *out of range* if otherwise.

2. Students pass their programming course if either they gain at least 85 marks in their coursework or gain at least 85 marks in their final exam. Write and test a program that inputs coursework and exam marks, and outputs whether they have passed, or not.

3. Write and test a program that inputs a pollen count, and prints *error* if the count is less than zero, *low* if the count is between 0 and 29 inclusive, *medium* if the count is between 30 and 69 inclusive, *high* if the count is between 70 and 100 inclusive, and *very high* if the count is more than 100.

4. Write and test a program that might help a marathon runner with their training. The program should input the number of steps taken in one minute, and print *speed up* if the steps taken is less than 175, *slow down* if the steps taken is more than 225, and *right on* if the number of steps per minute is between 175 and 225 inclusive. If the number of steps per minute is less than zero or more than 500, the program should write *error*.

5. Normal body temperature is 35.7 ± 2.5 Celsius. Write and test a program that inputs a body temperature, and writes *normal*, *hypothermic* (too cold) or *hyperthermic* (too hot) as appropriate.

6. Write and test a program that inputs a person's weight (in Kg) and the person's height (in metres), and prints their weight state according to the table shown below

weight state	***bmi***
starvation	< 15.0
underweight	15.0 to18.5
normal	18.6 to 24.9
overweight	25.0 to 29.9
obese	30.0 to 40.0
morbidly obese	> 40.0

Body mass index is: bmi = weight / height^2

We have seen how to use the logical operators, and how to write a sequence of else if's.

Next we continue our study of selections by investigating switch-with-case and the conditional operator ?:

Bibliography

Kernighan B and Ritchie D *The C Programming Language* Prentice Hall 1988
Mark Williams Company *ANSI C A Lexical Guide* Prentice Hall 1988