11 Character-string arrays

11.1 Danger signals

This warning is repeated from the previous section on arrays, but not included in the recorded lectures.

C allows an array of given size to be created, but then pays no attention to the size. This is the source of most 'segmentation fault' errors in C programming, and far worse. One must be very careful.

For example,

```
char hello[5] = "hello";
```

will compile, but it is an instance of *array overflow.* Six characters are initialised, and the last (null character) is beyond the range of the array.

11.2 Arrays of character strings

As already mentioned in the previous section, but not in the recorded version,

```
char * xxx[  ... ] =
{ "string", etcetera };
```

defines an initialised array of character strings.

Also, a character string can be printed using the `%s` format. The following example shows how.

```
% cat tengreen.c
#include <stdio.h>
int main()
{ char * one2ten [ 10 ] =
   { "one", "two", "three", "four", "five", "six",
     "seven", "eight", "nine", "ten" };

  char otherwords[36] = "green bottles, standing on the wall";

  int i;
  for ( i = 9; i >= 7; i = i-1 )
  {
    printf("%s %s\n", one2ten[i], otherwords);
  }
}
% a.out
ten green bottles, standing on the wall
nine green bottles, standing on the wall
eight green bottles, standing on the wall
%
```

11.3 Command-line arguments

There is a similar construction through which *command-line arguments* can be passed to the program.

This is an alternative way to pass small amounts of data to the program. It will be needed for the fourth programming assignment.

It requires a different `int main ( ... )` line:

```
int main ( int argc, char * argv[] )
  // argc is the number of command-line arguments, including
  // 'a.out'.  argv[] is an array of character strings in
  // which the command-line arguments are passed to the program.
```

Besides this, we need three new features.

```
  atoi : a function which converts character
         strings to integers.
  #include<stdlib.h> : This is needed when using the
         function atoi()
  %02d format item: for printing integers,
           padded with zeroes if necessary to make them
           at least 2 decimal digits long.
```

The following example takes three command-line arguments, besides the `a.out` which is always the string in `argv[0]`, assuming they represent a date in this century, and prints them out in a slightly different format.

```
% cat daymonthyear.c
#include <stdio.h>
#include <stdlib.h>  // Required for atoi

int main ( int argc, char * argv[] )
{
  int dd = atoi ( argv[1] );
  int mm = atoi ( argv[2] );
  int yy = atoi ( argv[3] );

  /*
   * The %02d format is for printing integer values,
   * but if necessary padding them with zeroes on
   * the left to produce an integer with at least
   * two digits.  This is useful for the next assignment.
   *
   * Furthermore, it is assumed that yy is between
   * 0 and 99, but it is printed as a 4-digit number.
   */
```

```
  printf("%02d/%02d/20%02d\n", dd, mm, yy );
}
% gcc daymonthyear.c
% a.out 2    11 20
02/11/2020
% a.out 1 1 0
01/01/2000
%
```

11.4 Difference between command-line arguments and keyboard input

- Data which is input using `scanf()` from the keyboard, not redirected, is not read off the command line but off the lines following it.
- Command-line arguments are on the same line as the command line.

In the above example, at least four command-line arguments are expected (including `a.out`). If there are fewer, `argv[1], argv[2],` or `argv[3]` will be uninitialised and this will cause a 'segmentation fault.'

```
% a.out
Segmentation fault
% a.out 1 2
Segmentation fault
```

11.5 Day in week calculation

One can convert a date, such as `2 11 20`, to a number from 0 to 6 representing Sunday to Saturday, the day on which that date falls.

It is an interesting use of arithmetic modulo 7.

We count the number of days since `1/1/2000`, which was a Saturday.

```
6 +       // Saturday
20 * 365 + // years
5 + // leap years
31+28+..31 = 304 + // months before November
2 // Day in month.

Since 365 mod 7 = 1, and 304 mod 7 is 3, we can
simplify this

6+20 + 5 + 3 + 2 = 36 = 1 mod 7.
Answer: day 1 in week, Monday.
```

This needs correction for a leap-year, whose extra day is at the end of February. In this case, if the month is January or February, the answer should have 1 subtracted (or 6 added, modulo 7)

11.6 Julian date

The Julian date is 5 decimal digits. The first two give the year, padded with a zero if necessary, and the last three give the day in year, from 1 to 366, padded with zeroes to 3 digits if necessary. Example.

```
Monday, 2 November 2020, Julian date 20307
Thursday, 31 December 2020, Julian date 20366
```