10 Arrays and initialisation

- Arrays in C are declared in the following way:

  ```
  int a[100];  double b[200];

  a is declared as an array of 100 ints,
  and b as an array of 200 doubles.
  ```

- The elements of the array `a` are

  ```
  a[0], a[1], a[2], ..., a[99]
  ```

  A peculiarity of C. Array indexing always begins at 0, so the last element in the array `a` is `a[99]`.

- The notation can be confusing.

  ```
  int a[10];   // declares a to be an array of 10 ints
  printf("%d\n", a[5]);        // the sixth entry in the array a.
  ```

- But in general, an array of int/double is equivalent to a list of several int/double variables.

10.1 Example reading an array from the keyboard

```
#include <stdio.h>
main()
{
  double a[1000];
  int count; double x;
  count = 0;
  while ( scanf("%lf", &x) == 1 )
  {
    if ( count < 1000 ) // ignores excess numbers
    {
      a[count] = x;
      count = count+1;
    }
  }
  printf("%d numbers read\n", count);
  int i;
  for (i=0; i<count; i=i+1)
  { printf(" %f", a[i]); }
  printf("\n");
  printf("and in reverse order\n");
  for (i=count-1; i>=0; i=i-1)
```

```
  { printf(" %f", a[i]); }
  printf("\n");
}
% gcc read-array.c
% cat da3
3.14 15.926 5.81 2 3.4
5 numbers read
 3.140000 15.926000 5.810000 2.000000 3.400000
and in reverse order
 3.400000 2.000000 5.810000 15.926000 3.140000
%
```

10.2 Initialisation

A **declaration** can include an **initial value**. Otherwise the value is undefined (garbage). Or maybe not garbage with modern compilers. Here is an example from a rather old version of gcc:

```
% cat garbage.c
#include <stdio.h>
main()
{ int x;
  printf("%d\n", x);
}
% gcc garbage.c
% a.out
-1217028108
```

Declaring with initialisation:

```
  int x = -345;
this is equivalent to
  int x; x = -345;
```

Declaring with initialisation saves keystrokes, but it should be treated ***with caution.*** There are two reasons. One is that you might base some part of the program on the assumption that `x` held its initial value, forgetting that you had already changed it. The other is that people ignorant of C programming think that `int` is required in every assignment to `x`, as with

```
  int x = 4;    // As a declaration, correct
  int x = 5;    // as an assignment, utterly wrong.
```

A great strength of C is an efficient way to initialise arrays. This is where an array is used as a table of values; for example, the lengths of the months in a non-leap year.

```
  int month[12]={31,28,31,30,31,30,31,31,30,31,30,31};
```

10.3 Character strings

A variable taking single character (actually, single byte) values is declared this way:

```
char x;
char y = 'z';
char newline = '\n';
```

- An array of characters is often called a *character string.*
- Usually, a character string is used to store a piece of text.
- Initialisation is possible. For example,

  ```
  char hello[6] = "hello";
  ```

- ***Important.*** First, this is possible as *initialisation,* but for technical reasons it is *impossible* as assignment. For example, the following statements are incorrect.

  ```
  char hello[6];
  hello = "hello";  // This is not a valid C statement
  ```

- ***Important.*** Secondly, "hello", which is called a character string constant, fits into 6 characters, not 5. The following is valid C code, but it is fatally flawed:

  ```
  char hello[5] = {'h','e','l','l','o'};
  ```

  It is wrong because the end of the string is not clearly marked. The end of a character string is always marked by a *null character.* This is written in C as `'\0'`, a byte consisting of 8 zero-bits, or $(00)_{16}$.[1]

  The following are both correct, and both have the same effect.

  ```
  char hello_1[6] = "hello";
  char hello[6] = {'h','e','l','l','o','\0'};
  ```

- A useful initialisation is

  ```
  char hex_digit[17] = "0123456789abcdef";
  ```

10.4 Danger signals

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.

[1] It differs from the ASCII code of `'0'` which is 48 or $(30)_{16}$.

10.5 Some fancy initialisations

It is possible to have an array of character strings of different lengths, initialised. For example,

```
char * weekday[7]={"Su","Mo","Tu","We","Th","Fr","Sa"};
```

All right, these character strings are the same length, but they needn't be.

What does this mean?

```
char * weekday[7] ....
```

says that `weekday[]` is an array of 7 character strings. The asterisk indicates character string in a sense which can't be explained now, but will be explained later.

Also,

```
char * month_name[12] =
{ "January", "February", "March", "April", "May", "June",
  "July", "August", "September", "October", "November", "December"
};
```

They are character strings of different lengths.

10.6 Format item for character strings

In `printf()`, the correct format item for

- a character is `%c`, and for
- a character string it is `%s`.

Fior example, with `month_name[]` declared as above,

```
  int i;
  for (i=0; i<12; i = i+1)
  { printf("%s ", month_name[i]); // no newline!
    if ( i ==  5)
    { printf("\n"); }
  }
  printf("\n");          // partial code fragment
...
% a.out
January February March April May June
July August September October November December
%
```