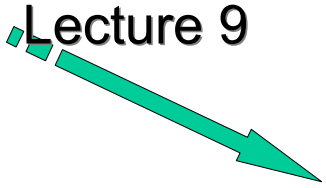
Lecture 9



1. Introduction
2. Binary Representation
3. Hardware and Software
4. High Level Languages
5. Standard input and output
6. Operators, expression and statements
7. Making Decisions
8. Looping
9. Arrays
10. Basics of pointers
11. Strings
12. Basics of functions
13. More about functions
14. Files
14. Data Structures
16. Case study: lottery number generator

Arrays

- For the last two lectures we have looked at algorithms. We shall now return to data structures.
- So far we have only looked at simple variables,
  - integers
  - characters
  - floating point numbers (real numbers)
- But often we get data in sets of 'like' objects, e.g.
  - images - 2D array of pixels
  - strings - 1D array of characters
- The order of the data is important

Declaring Arrays

- C provides a simple data structure, applicable to all data types: the array
  - `int teleph[100]; /* 100 telephone numbers*/`
  - `float marks[115]; /* 115 exam marks */`
  - `char text[20]; /* a string 20 characters long */`
- We can use any constant integer expression to declare the size of an array:

```
#define CLASS_SIZE 108 /* number of students*/
      - - - -
float marks[CLASS_SIZE];
int thing[2*3+5];
```

Declaring Arrays

- We can assign initial values to any array,

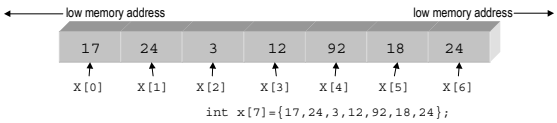
```
    int a[3]={10,11,12};
or  int a[]={10,11,12};
```

- C counts the number of items in the initialisation list
- The above declaration assigns values to the individual array elements as follows:

  `a[0]=10,  a[1]=11,  a[2]=12`
- Each a[…] is an integer variable. They are identified by subscripts in square brackets [ ]
- In C, subscripts always begin at Zero i.e. a[0]
  - Well see why in the next lecture

Array Storage

- Array elements are stored sequentially in memory, starting with the [0] element




```
int x[7]={17,24,3,12,92,18,24};
```

- Note that although we declare x[7] (7 elements), the highest subscript is 6. If we try to access x[7] the computer will let us but the result will be rubbish or may result in the programming crashing

Subscripts and loops

- It is very common to use a for loop to access each element of an array in turn.

array2.c

```
/* Example: arrays */

#include <stdio.h>

#define MAX 21

main()
{
  int u;
  long p2[MAX]; /* powers of two */

  /* initialise array: */

  p2[0] = 1;
  for (u = 1; u < MAX; u++)
    p2[u] = 2 * p2[u - 1];

  /* print: */

  for (u = 0; u < MAX; u++)
    printf("2 to the power %2i = %7li\n", u, p2[u]);
}
```

array1.c

```
/* Example: arrays */

#include <stdio.h>

main()
{
  int u;
  long p2[21] = { 1, 2, 4, 8, 16, 32, 64, 128,
                  256, 512, 1024, 2048, 4096,
                  8192, 16384, 32768, 65536,
                  131072, 262144, 524288,
                  1048576
                };

  for (u = 0; u < 21; u++)
    printf("2 to the power %2i = %7li\n", u, p2[u]);
}
```

Vectors as Arrays

- A vector in 3-space has three components (x,y,z for a position vector).
- It can be represented by a 3 element array.

array3.c

```
/* Example: vectors represented by arrays */

#include <stdio.h>

main()
{
  int i;
  int a[3] = {2, 4, 6};
  int b[3] = {-5, 1, -9};
  int c[3], d[3];

  puts("Vectors in 3-space:\n");

  printf("a     = [%2i  %2i  %2i]\n\n", a[0], a[1], a[2]);

  printf("b     = [%2i  %2i  %2i]\n\n", b[0], b[1], b[2]);

  /* form vector sum: */

  for (i = 0; i < 3; i++)
    c[i] =  a[i] + b[i];

  printf("a + b = [%2i  %2i  %2i]\n\n", c[0], c[1], c[2]);

  /* form vector difference: */

  for (i = 0; i < 3; i++)
    d[i] =  a[i] - b[i];

  printf("a - b = [%2i  %2i  %2i]\n", d[0], d[1], d[2]);
}
```

Strings as Arrays

- We'll look at this in more detail later, but note that strings are ordered sets of characters

analyse.c

```
* Example: analysis of text */

#include <stdio.h>

#define MAX 1000 /* The maximum number of characters */

main()
{
  char text[MAX], c;
  int i, lc, uc, dig, oth;

  puts("Type some text (then ENTER):");

  /* Save typed characters in text[]: */

  for (i = 0; i < MAX; i++)
  {
    text[i] = getchar();
    if (text[i] == '\n')
      break;
  }
```

```
/* Analyse contents of text[]: */

  for (i = lc = uc = dig = oth = 0; i < MAX; i++)
  {
    c = text[i];
    if (c >= 'a' && c <= 'z')
      lc++;
    else if (c >= 'A' && c <= 'Z')
      uc++;
    else if (c >= '0' && c <= '9')
      dig++;
    else
    {
      oth++;
      if (c == '\n')
      break;
    }
  }

  puts("\nYou typed:");
  printf("%i% lower case letters\n", lc);
  printf("%i upper case letters\n", uc);
  printf("%i digits\n", dig);
  printf("%i others\n", oth);
}
```

Sorting an Array

- Very often we would like to sort an array into order.
- There are many ways to do this, but the simplest is the *bubble sort*
- A *bubble sort* uses 2 nested loops
  - Values in the array are compared in pairs and swapped if necessary so the larger value is in the higher position.
  - This continues until no more swaps are needed.
  - The value 'bubbles up' to the 'top' of the array.

bubble.c

```
/* Example: bubble sort values in array */

/* The bubble sort works by comparing values in adjacent
   array positions. If the value in the "lower" position is
   greater than that in the higher position, the two are
   swapped. Thus the largest value "bubbles up" to the "top"
   of the array. When there are no more values to be swapped,
   sorting is complete. */

#include <stdio.h>

#define VALS 10 /* The number of values to be sorted */

main()
{
  int i, sorted, swaps = 0;
  float x[VALS], temp;

  /* Input values: */

  printf("Enter %i floating point values:\n", VALS);
  for (i = 0; i < VALS; i++)
  { printf("#%i: ", i + 1);
    scanf("%f", &x[i]);
  }

  /* Bubble sort: */

  do {  for (i = 1, sorted = 1; i < VALS; i++)
        {  if (x[i-1] > x[i])
           {  sorted = 0;
              temp = x[i-1];
              x[i-1] = x[i];
              x[i] = temp;
              swaps++;
           }
        }
  } while (!sorted);

  /* Output sorted values: */

  puts("\nIn ascending order, the values are:");
  for (i = 0; i < VALS; i++)
    printf("%G\n", x[i]);
  printf("\n%i swaps were performed.\n", swaps);
}
```

Two dimensional Arrays

- Sometimes the data has a 2D structure e.g. a table or image

```
int table [2][3]
unsigned char image[640][480]
```

- The array can be initialised by a array of arrays

	COL 0	COL 1	COL 2
ROW 0	X[0][0]	X[0][1]	X[0][2]
ROW 1	X[1][0]	X[1][1]	X[1][2]

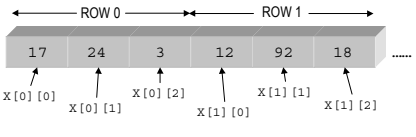
X[i][j]

Row index

Column index

Two dimensional Arrays

- Two dimensional arrays are actually stored in a linear fashion, row by row




```
int x[2][3]={{27,19,53},{14,41,59}};
```

Multidimensional arrays

- You can have as many subscripts as you like, though its rare to have more than three:

```
double large[10][20][30][40];
```

- Again the array is stored linearly, with the RH index varying most rapidly and the LH index most slowly
  - Think of it like a mileometer

```
/* BUG ZONE!!!
Example: arrays */

#include <stdio.h>

#define N 2
#define M 5
#define SIZE 17

main()
{
  int carrots[SIZE], parsnips[4][3];
  int i, j, n = 2, m = 5;
  float cabbages[N][M], potatoes[n][m];  /* BUG */
  float x = 4.17, y = 5.73;

  for (i = 1; i <= SIZE; i++)  /* BUG */
    carrots[i] = 99;

  for (i = 0; i < 3; i++)
    for (j = 0; j < 4; j++)  /* BUG */
      parsnips[i][j] = 0;

  carrots[n+m] = 37;
  carrots[n-m] = 38;  /* BUG */
  parsnips[x][y] = 13.654;  /* BUG */
}
```

array.bug