

Objectives

This section brings together the use of two of C's fundamental data types, pointers and arrays, in the use of handling strings.

Having read this section you should be able to:

1. handle any string constant by storing it in an array.

Stringing Along:

Now that we have mastered pointers and the relationship between arrays and pointers we can take a second look at strings. A string is just a character array with the convention that the end of the valid data is marked by a null '\0'. Now you should be able to see why you can read in a character string using `scanf("%s", name)` rather than `scanf("%s",&name)` - name is already a pointer variable. Manipulating strings is very much a matter of pointers and special string functions. For example, the `strlen(str)` function returns the number of characters in the string `str`. It does this simply by counting the number of characters up to the first null in the character array - so it is important that you are using a valid null-terminated string. Indeed this is important with all of the C string functions.

You might not think that you need a function to copy strings, but simple assignment between string variables doesn't work. For example:

```
char a[10],b[10];  
b = a;
```

does not appear to make a copy of the characters in `a`, but this is an illusion. What actually happens is that the pointer `b` is set to point to the same set of characters that `a` points to, i.e. a second copy of the string isn't created.

To do this you need `strcpy(a,b)` which really does make a copy of every character in `a` in the array `b` up to the first null character. In a similar fashion `strcat(a,b)` adds the characters in `b` to the end of the string stored in `a`. Finally there is the all-important `strcmp(a,b)` which compares the two strings character by character and returns true - that is 0 - if the results are equal.

Again notice that you can't compare strings using `a==b` because this just tests to see if the two pointers `a` and `b` are pointing to the same memory location. Of course if they are then the two strings are the same, but it is still possible for two strings to be the same even if they are stored at different locations.

You can see that you need to understand pointers to avoid making simple mistakes using strings. One last problem is how to initialise a character array to a string. You can't use:

```
a = "hello";
```

because `a` is a pointer and `"hello"` is a string constant. However, you can use:

```
strcpy(a,"hello")
```

because a string constant is passed in exactly the same way as a string variable, i.e. as a pointer. If you are worried where the string constant is stored, the answer is in a special area of memory along with all of the constants that the program uses. The main disadvantage of this method is that many compilers use an optimisation trick that results in only a single version of identical constants being stored. For example:

```
strcpy(b,"hello");
```

usually ends up with `b` pointing to the same string as `a`. In other words, this method isn't particularly safe!

A much better method is to use array initialisation. You can specify constants to be used to initialise any variable when it is declared. For example:

```
int a=10;
```

declares `a` to be an integer and initialises it to 10. You can initialise an array using a similar notation. For example:

```
int a[5] = {1,2,3,4,5};
```

declares an integer array and initialises it so that a[0]= 1, a[1] = 2 and so on. A character array can be initialised in the same way. For example:

```
char a[5]='h','e','l','l','o';
```

but a much better way is to write:

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

which also automatically stores a null character at the end of the string - hence a[6] and not a[5]. If you really want to be lazy you can use:

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

and let the compiler work out how many array elements are needed. Some compilers cannot cope with the idea of initialising a variable that doesn't exist for the entire life of the program. For those compilers to make initialisation work you need to add the keyword static to the front of the string declaration, therefore:

```
static char a[] = "hello";
```

As easy as... B or C?:

A few words of warning. If you are familiar with BASIC then you will have to treat C strings, and even C arrays, with some caution. They are not as easy or as obvious to use and writing a program that manipulates text is harder in C than in BASIC. If you try to use C strings as if it were BASIC strings you are sure to create some very weird and wonderful bugs!

A Sort Of Bubble Program:

This sections program implements a simple bubble sort - which is notorious for being one of the worst sorting methods known to programmer-kind, but it does have the advantage of being easy and instructive. Some of the routines have already been described in the main text and a range of different methods of passing data in functions have also been used.

The main routine is sort which repeats the scan function on the array until the variable done is set to 0. The scan function simply scans down the array comparing elements that are next door to each other. If they are in the wrong order then function swap is called to swap them over.

Study this program carefully with particular attention to the way arrays, array elements and variables are passed. It is worth saying that in some cases there are better ways of achieving the same results. In particular, it would have been easier not to use the variable done, but to have returned the state as the result of the scan function.

```
#include <stdio.h>
```

```
void randdat(int a[] , int n);  
void sort(int a[] , int n);  
void scan(int a[] , int n , int *done);  
void swap(int *a ,int *b);
```

```
main()  
{  
    int i;  
    int a[20];  
  
    randdat(a , 20);  
    sort(a , 20);  
  
    for(i=0;i<20;++i) printf("%d\n" ,a[i]);  
}
```

```
void randdat(int a[1] , int n)  
{
```

```
int i;
for (i=0 ; i<n ; ++i)
    a[i] = rand()%n+1;
}

void sort(int a[1] , int n)
{
    int done;
    done = 1;
    while(done == 1) scan(a , n , &done);
}

void scan(int a[1] , int n , int *done)
{
    int i;
    *done=0;
    for(i=0 ; i<n-1 ; ++i)
    {
        if(a[i]<a[i+1])
        {
            swap(&a[i],&a[i+1]);
            *done=1;
        }
    }
}

void swap(int *a ,int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```