C Programming under Linux

P2T Course, Martinmas 2003–4
C Lecture 3

Dr Ralf Kaiser

Room 514, Department of Physics and Astronomy

University of Glasgow

r.kaiser@physics.gla.ac.uk

Summary

- Arrays
- Strings
- Reading in from the Keyboard

`http://www.physics.gla.ac.uk/~kaiser/`

Arrays

- An array is a set of consecutive memory locations used to store data.
- Each item in the array is called an element. The number of elements in an array is called the dimension of the array.
- A typical array declaration is

```
/* List of data to be sorted and averaged */
int data_list[3];
```

  In this case the array `data_list` contains the 3 elements `data_list[0]`, `data_list[1]` and `data_list[2]`. The number in square brackets [] is the index.
- C starts counting at 0, not at 1.

Arrays - Example

Calculate sum and average of five numbers. (`array.c`)

```
#include <stdio.h>

float data[5];  /* data to average and total */
float total;    /* the total of the data items */
float average;  /* average of the items */

int main()
{
    data[0] = 34.0;
    data[1] = 27.0;
    data[2] = 45.0;
    data[3] = 82.0;
    data[4] = 22.0;

    total = data[0] + data[1] + data[2] + data[3] + data[4];
    average =  total / 5.0;
    printf("Total %f Average %f\n", total, average);
    return (0);
}
```

Output:

```
Total 210.000000 Average 42.000000
```

Multidimensional Arrays

- Arrays can have more than one dimension. The declaration for a two-dimensional array is
  `type variable[size1][size2]; /* comment */`
- Example:
  `int matrix[2][4]; /* a typical matrix */`
- C does not follow the notation used in other languages, e.g. `matrix[10,12]`.
- to access an element of the two-dimensional array `matrix` we use the notation
  `matrix[1][2] = 10;`
- C allows to use as many dimensions as needed, limited only by the amount of memory available. A four-dimensional array `four_dimensions[10][12][9][5]` is no problem.

Multidimensional Arrays - Example

(p_array.c)

```
#include <stdio.h>

/* Array of numbers */
int array[3][2];

int main()
{
    int x,y; /* Loop indices */

    array[0][0] = 0 * 10 + 0;
    array[0][1] = 0 * 10 + 1;
    array[1][0] = 1 * 10 + 0;
    array[1][1] = 1 * 10 + 1;
    array[2][0] = 2 * 10 + 0;
    array[2][1] = 2 * 10 + 1;

    printf("array[%d] ", 0);
    printf("%d ", array[0,0]);
    printf("%d ", array[0,1]);
    printf("\n");

    printf("array[%d] ", 1);
    printf("%d ", array[1,0]);
    printf("%d ", array[1,1]);
    printf("\n");

    printf("array[%d] ", 2);
    printf("%d ", array[2,0]);
    printf("%d ", array[2,1]);
    printf("\n");

    return (0);
}
```

What's the problem with this example ?

Multidimensional Arrays - Example cont.

The program on the previous slide produces the following output (again, on my laptop):

```
kaiser@npl03:~/oreilly/pracc/p_array> p_array
array[0] 134518256 134518264
array[1] 134518256 134518264
array[2] 134518256 134518264
```

The problem is the use of the expression `array[x,y]` in the `printf` statement, instead of using the correct expression `array[x][y]`, which would have resulted in

```
kaiser@npl03:~/oreilly/pracc/p_array> p_array
array[0] 0 1
array[1] 10 11
array[2] 20 21
```

More detail: `x,y` is equivalent to `y`, therefore `array[x,y]` is really `array[y]`, which is a pointer to row `y` of the array. But we will only later learn about pointers...

Initialising Arrays

- C allows variables to be initialised in the declaration statement
  `int counter = 0; /* number counted so far */`
- This is especially practical for arrays, where a list of element enclosed in curly braces {} can be assigned:
  `int product_codes[3] = {10, 972, 45};`
- If no dimension is given, C will determine the dimension from the number of elements in the initialisation list:
  `int product_codes[] = {10, 972, 45};`
- The same kind of initialisation at declaration can also be used for multidimensional arrays:

```
int matrix[2][4] =
    {
        {1, 2, 3 ,4},
        {10, 20, 30, 40}
    };
```

Strings

- Strings are sequences of characters. C does not have a built-in string type; instead, strings are created out of character arrays.
- Strings are character arrays with the additional special character \0 (NUL) at the end.
- String constants consist of text enclosed in double quotes, i.e. `"Linux"`. The first parameter to `printf` is a string constant.
- C does not allow to assign one array to another, instead, to fill an array with a string constant we have to copy it into the variable using the standard library function `strcpy`:

```
#include <string.h>
char system[6];
int main(){
    strcpy(system, "Linux");   /* Legal way to fill variable system */
    return(0);
}
```

Strings cont.

- The array can also be filled element by element:

```
system[0] = 'L';
system[1] = 'i';
system[2] = 'n';
system[3] = 'u';
system[4] = 'x';
system[5] = '\0';
```

- Because C allows variables to be initialised at declaration, this can be used to fill a string in a convenient way. In this case you don't even have to specify the length of the array. C will determine the dimension of the array itself.
  `char system[] = "Linux"`
- The above declaration is equivalent to the following initialisation:
  `char system[] = 'L','i','n','u','x','\0';`

Strings cont.

- String and character constants are different: `"X"` is a one-character string, taking up two bytes, one for X, the other one for `\0`. 'Y' is just a single character, taking up one byte.
- A string should never be copied into an array that is shorter than the string. Otherwise you are writing into memory that you shouldn't access and the program can behave unexpectedly.
- The most common string functions are

Function	**Description**
`strcpy(string1, string2)`	copy `string2` into `string1`
`strcat(string1, string2)`	concatenate `string2` onto the end of `string1`
`length = strlen(string)`	get the length of a `string`
`strcmp(string1, string2)`	0 if `string1` equals `string2`, otherwise nonzero

Strings - Example

Putting strings together using `strcat(full.c)`.

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

char first[100];        /* first name */
char last[100];         /* last name */
char full_name[200];    /* full version of first and last name */

int main()
{
    strcpy(first, "John");        /* Initialize first name */
    strcpy(last, "Lennon");       /* Initialize last name  */

    strcpy(full_name, first);     /* full = "John"         */
    /* Note: strcat not strcpy */
    strcat(full_name, " ");       /* full = "John "        */
    strcat(full_name, last);      /* full = "John Lennon"  */

    printf("The full name is %s\n", full_name);
    return (0);
}
```

Output:

```
The full name is John Lennon
```

Reading in Strings with `fgets`

The standard function `fgets` can be used to read a string from the keyboard. The general form of an `fgets` statement is:

```
fgets(name, size, stdin);
```

- `name`
  is the name of a character array, aka a string variable. The line, including the end-of-line character, is read into this array.
- `size`
  `fgets` reads until it gets a line complete with ending \n or it reads `size` - 1 characters. It is convenient to use the `sizeof` function: `fgets(name, sizeof(name), stdin);` because it provides a way of limiting the number of characters read to the maximum number that the variable can hold.
- stdin
  is the file to read. In this case the 'file' is the standard input or keyboard. Other files will be discussed later under file input/output.

`fgets` - Example 1

Read in a string and output it's length (`length.c`).

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

char line[100]; /* Line we are looking at */

int main()
{
    printf("Enter a line: ");
    fgets(line, sizeof(line), stdin);

    printf("The length of the line is: %d\n", strlen(line));
    return (0);
}
```

Output:

```
Enter a line: test
The length of the line is:  5
```

`test` has only 4 characters - but `fgets` also gets the `\n` character.

`fgets` - Example 2

Read in first and last name, print out full name (`full1.c`).

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

char first[100];        /* first name of person we are working with */
char last[100];         /* last name                                */
char full[200];         /* full name of the person (computed)       */

int main() {
    printf("Enter first name: ");
    fgets(first, sizeof(first), stdin);

    printf("Enter last name: ");
    fgets(last, sizeof(last), stdin);

    strcpy(full, first);
    strcat(full, " ");
    strcat(full, last);

    printf("The name is %s\n", full);
    return (0);
}
```

`fgets` - Example 2 cont.

Output of `fgets` - Example 2:

```
kaiser@npl03:~/oreilly/pracc/full1> full1
Enter first name: John
Enter last name: Lennon
The name is John
 Lennon
```

What happened ? Why is the last name in a new line ?

- The `fgets` command gets the entire line, including the end-of-line. For example, "John" gets stored as {'J','o','h','n','\n','\0'}.
- This can be fixed by using the statement
  `first[strlen(first)-1] = '\0';`
  which replaces the end-of-line with an end-of-string character and so end the string earlier.

The Function `scanf`

- The direct equivalent to the output function `printf` is the input function `scanf`. The syntax of a `scanf` statement is
  `scanf(format, &variable-1, &variable-2, ...)`
  where `format` specifies the types of variables and `&variable-1` is the address of variable-1.
- A typical `scanf` statement would be
  `scanf("%d%d%f", &a, &b, &x)`
  reading in integer values for the variables a and b and a floating point value for the variable x, entered from the keyboard.
- The `%s` conversion in `scanf` ignores leading blanks and reads until either the end-of-string '\0' or the first blank after non-blank characters. For example, if the input from the keyboard is " ABCD EFG ", %s will read "ABCD".

Reading Numbers with `fgets` and `sscanf`

- To quote my favorite C book: 'The function `scanf` provides a simple and easy way of reading numbers that almost never works'. `scanf` is not very good at end-of-line handling and you may find yourself having to hit 'return' a couple of times.
- One way around these problems is to use a combination of `fgets` and `sscanf` instead. `sscanf` stands for 'string `scanf`' and works like `scanf`, but acts on strings rather than on keyboard input.
- The code to read in and process a line from the keyboard then looks like this:

```
char line[100];
fgets(line, sizeof(line), stdin);
sscanf(line, format, &variable-1,...);
```

fgets/sscanf - Example 1

Read in a number from the keyboard and double it (double.c).

```
#include <stdio.h>
char  line[100];   /* input line from console */
int   value;       /* a value to double */

int main()
{
    printf("Enter a value: ");

    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d", &value);

    printf("Twice %d is %d\n", value, value * 2);
    return (0);
}
```

Output:

```
kaiser@npl03:~/oreilly/pracc/double> ./double
Enter a value: 21
Twice 21 is 42
```

Actually, `scanf("%d", &value);` worked as well.

`fgets/sscanf` - Example 2

Input width and height, output area of triangle (`tri.c`).

```
#include <stdio.h>
char line[100];/* line of input data */
int  height;   /* the height of the triangle
int  width;    /* the width of the triangle */
int  area;     /* area of the triangle (computed) */

int main()
{
    printf("Enter width height? ");

    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d %d", &width, &height);

    area = (width * height) / 2;
    printf("The area is %d\n", area)
    return (0);
}
```

`fgets/sscanf` - Example 2 cont

If we are trying to compile the program `tri.c` we get the following error messages:

```
kaiser@npl03:~/oreilly/pracc/tri> make
This program fails to compile
gcc -g -Wall -D__USE_FIXED_PROTOTYPES__ -ansi -o tri tri.c
tri.c:18:16: warning: "/*" within comment
tri.c: In function ‘main’:
tri.c:26: ‘width’ undeclared (first use in this function)
tri.c:26: (Each undeclared identifier is reported only once
tri.c:26: for each function it appears in.)
tri.c:30: parse error before "return"
make: *** [tri] Error 1
```

The source code of tri.c contains two of the most common errors: One missing `*/` at the end of a comment:

```
int height; /* the height of the triangle */
```

and one missing semi-colon at the end of the second `printf` statement:

```
printf("The area is %d\n", area);
```