Design and Computing 1

Introduction to Scientific Computing

Daniel Poole
Department of Aerospace Engineering
University of Bristol
d.j.poole@bristol.ac.uk

2017

LECTURE 7

Intro to Scientific Computing

- Have used programs to perform simple calculations
- Conditionals - branching; loops - perform calculations multiple times
- Can split a problem down and program using divide-and conquer using functions
- Can read and write with files and use scientific plotting software
- Can store data in structured arrays, and manipulate that data accordingly

TODAY

- Strings - arrays of characters
- Pointers
- Compiling with multiple files

Lecture Content

In this lecture, we will tie up a few loose ends of the introduction to C content that we will be learning. In general, for engineering/mathematical computing, strings do not tend to be used a huge deal. However, they are still a fundamental aspect of any programming language, hence it is important to introduce the concept and understand what they are, how they work and why they are useful. The concept of pointers, however, is a much more important one. The reason why we have not gone into the detail of pointers is primarily due to the complications that arise in the understanding of the subtlety of using them. Furthermore, it is reasonably simple to avoid the use of pointers entirely in C (at least for the problems you will tackle). Like strings, however, it is important that you understand these subtleties.

There is no associated lab with the contents of this lecture, however, you should still understand the content of this lecture as it may be useful for some of the future assignments. In particular, you should ensure you understand what a pointer is, how its used and why its useful in C.

We will also briefly touch on compiling codes split across multiple files.

Strings

ASCII Character Set

First, let us visit the ASCII library. The standard ASCII data set encodes 128 characters. Each character is assigned in binary, but also has a decimal value associated with it. These are the character values that can be assigned to a character variable:

`http://www.ascii-code.com/`

When we assign a character to the `char` variable, we assign an ASCII encoded binary number. That is we actually encode a number, but the computer sees we have a `char` variable and outputs us the corresponding ASCII character.

Hence, say we wanted a single character variable to be assigned `G`, this is ASCII number 71, so:

```
char letter;
int number;
letter = 'G';
printf("%i \n",letter);
number = 71;
letter = number;
printf("%c \n",letter);
```

```
71
G
```

Strings

Strings are a relatively simple concept in C, especially since we have already covered arrays in the last lecture.

A string is simply an **array of characters**. Each element of the array is therefore of type `char`, and like arrays, C can treat strings as a single entity, and can be manipulated in a very similar fashion to numerical arrays.

An important aspect of strings is the vast library that can be called upon to perform certain string manipulations:

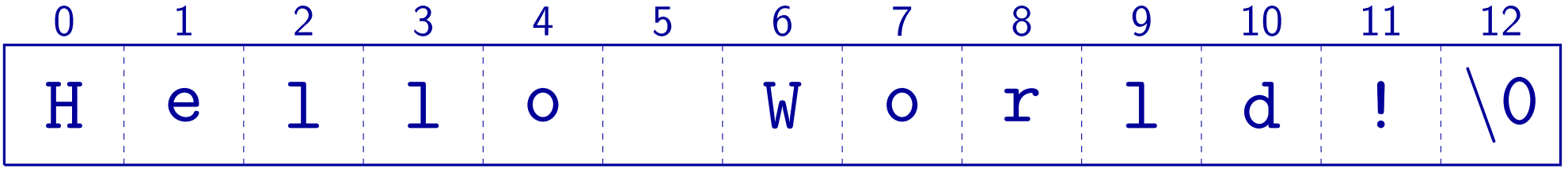
```
#include<string.h>
```

`http://www.cplusplus.com/reference/cstring/`

String Syntax

```
#include<stdio.h>
int main () {
  char message[13]="Hello World!";
  printf("%s \n",message);
```

```
Hello World!
```



String Syntax

`%s`

Tells the `printf` function to print a string

Number of characters

Note that the number of entries in the string in one more than the total number of characters we are printing. This extra entry is for a terminating character $\backslash\mathbf{0}$. This is not something you have to add, but you must always provide a string with one extra character than you will need.

String Manipulation

We are dealing with arrays so C can only modify one element at a time. Lets take the example of adding two strings together (called **string concatenation**), we can't just do:

```
message1="Hello ";
message2="World!";
message=message1+message2;
```

We must use one of the `string.h` functions - `strcat`

String Manipulation

So to add two strings together we can use:

`strcat(string1,string2)`: take `string2` and add it to the end of `string1`

`strncat(string1,string2,n)`: take the first `n` characters of `string2` and add them to the end of `string1`

NOTE: `string1` must contain enough blank characters to accommodate `string2`.

```
#include<stdio.h>
#include<string.h>
int main () {
  char message1[13]="Hello ";
  char message2[7]="World!";
  strcat(message1,message2);
  printf("%s\n",message1);
  return(0);
}
```

String Manipulation

Other string manipulation functions within `string.h` include:

- `strcpy(string1,string2)`: copy `string2` into `string1`
- `strncpy(string1,string2,n)`: copy the first `n` characters of `string2` into the first `n` characters of `string1`
- `strlen(string1)`: returns the length of `string1`
- `strcmp(string1,string2)`: returns:
  - <0 if `string1` lexically less than `string2`
  - 0 if `string1` lexically equal `string2`
  - >0 if `string1` lexically greater than `string2`

Reading files into strings

When reading from files, often we may need to assign lines of files to strings. We can then interrogate the string as appropriate.

We can read one character at a time from a file:

```
FILE *fid;
...
for(ii=0; ii<=13; ii++){
  string[ii]=fgetc(fid);
}
```

Or read until white space occurs:

```
fscanf(fid,"%s",&string);
```

Or one line at a time:

```
fgets(string,80,fid);
```

Reading from Strings

We can also read shorter strings from longer strings (useful if reading rows of data from a file):

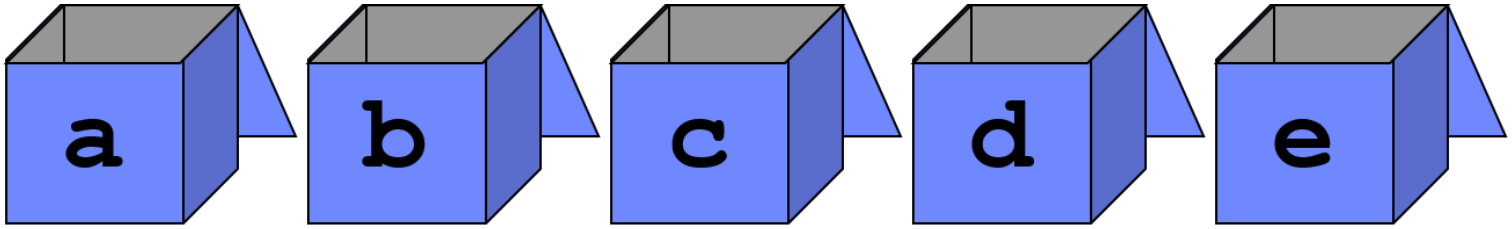
```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int main () {
  char string[80], word1[10], word2[10];
  int i1; double d1;
  ...
  fgets(string, 80, fid);
  sscanf(string, "%s %s",word1,word2);
  i1=atoi(word1);
  d1=atof(word2);
}
```

NOTE: we can do type conversion using `atoi` and `atof` to covert from a string to either an integer or a double.

Pointers

Variable Storage

Remember in lecture 2 we use the analogy of a variable being a number within a box. The box has a name (say a), which is the name of our variable, and the number within the box is the number within the variable. Now suppose we have lots of these boxes arranged in a library. Each box has a particular place within the library, which itself has some sort of location/address in memory.



In C, instead of changing the value of the variable directly, we can change it indirectly by storing the value of the location and telling C to manipulate what is in that location. The location can be stored in what is called a **POINTER VARIABLE**.

Pointer Syntax

A pointer to a given type is declared as:

```
int *p;
double *q;
```

This means p is a pointer to a variable that is an integer, and q is a pointer to a variable that is an double. Remember that the values stored in the variables p and q are the locations of different variables, not the values of the variables themselves.

To get the location of a variable x:

```
int *p, x;
p=&x;
```

This means get the location of the variable x and store it within the pointer variable p.

Using Pointers

Once we have stored the location of a variable within a pointer, the pointer variable and the variable itself can be used to change the value of the variable:

```
int *p, x;
x=5;
printf("x=%i \n",x);
p=&x;
*p=2;
printf("x=%i \n",x);
```

This means get the location of `x`, store it in `p` and change the value of the location of the variable store in `p`.

Pointers and Functions

When we considered functions, we considered functions that can only return one output (multiple input, single output). However, this is often not useful in programming. If we want a function to return multiple pieces of data (or at least manipulate and store multiple pieces of data), then we can send a pointer to the variable we want to change.

```
int power (double z, double *square, double *cube) {
  *square=z*z;
  *cube=(*square)*z;
  return(0);
}
int main () {
  double x,xsqu,xcub;
  x=2.0;
  power(x, &xsqu, &xcub);
  return(0);
}
```

Compiling with multiple files

Multiple Files

In all of the exercises that we have done so far, we have written all of the source code in a single file. For simple exercises this is obviously fine, but clearly, when developing a large code, this file would become unwieldy. Furthermore, in a multi-developer environment, it is impractical to all be working on a single file.

It is normally common to write all functions in separate files and link these together at compile time.

Multiple Files

If we separate our functions into separate files then we need to tell the file from which our function is called to look for that function. Functions are split into the header the body.

```
double myfunc(int xin)
{

   ...


}
```

Header

Body

When compiling the function that calls this function (which is often `main`), the header (also called the interface) needs to be known. We therefore separate the function headers into an overall header file, which has a `.h` extension.

Multiple Files

myfuncs.h

```
int printvar(int a);
```

printvar.c

```
#include <stdio.h>
int printvar(int a){
  printf("var=%i\n",a);
  return(0);
}
```

main.c

```
#include "myfuncs.h"
int main(){
  int x=1;
  printvar(x);
  return(0);
}
```

Multiple Files

The header file therefore tells the main program what function interfaces to expect.

Compiling is then done using all of the C files. Obviously the exact compilation nomenclature is compiler dependent, however, in gcc (this must be done from command line), the executable program called `prog.exe` is created thus:

```
>> gcc -o prog.exe main.c printvar.c
```

Note that the header file is not compiled but is just used as a reference for the main program.

Summary

- Have introduced strings and how to read to/from and manipulate
- Have introduced the idea of pointers
- Can now manipulate and store multiple pieces of data in a function
- Can compile multiple files containing different pieces of code

Summary

LAB

- No new exercise
- Ensure first five exercises are **MARKED BY END OF SESSION**

STUDY WEEK

- Complete strings exercise on blackboard
- Not marked but may be useful for TB2