Computer Programming using C  
Lecture 9: Pointers and functions

Dr Julian Miller  
Room P/M/101  
Ext 2383; E-mail: jfm@ohm  
http://www.elec.york.ac.uk/intsys/users/jfm7

Contents

• Revision of last lecture (via solution to ex11.4)  
• Passing things to functions  
  – Passing by value  
  – Passing by reference  
• Passing structures to functions
Structures: Defining and accessing

- Structures in C are **variables** that can store and manipulate information of **different types** in a unified way.
- To illustrate this let’s look at a solution to lab exercise 11.4.
- This asked for a program that could:
  - Take the details of up to 10 students using an array of structures.
  - Display these details.
  - The structure required is below:

```c
typedef struct long_student_type
{
    char family_name[40];
    char given_name[40];
    int year_of_birth;
    int course_code;
} student_type;
```

Displaying a student’s data

- You were asked to write a function that could display the details of a single student.

```c
void display_student(student_type student) {
    printf("*** student data ***\n\n");
    printf(" Last Name: %s\n", student.family_name);
    printf(" First Name: %s\n", student.given_name);
    printf(" Year of Birth: %d\n", student.year_of_birth);
    printf(" Course Code: %d\n", student.course_code);
}
```

- Also useful to have a function display all student data.

```c
void display_all_students(student_type students[10], int num_students) {
    int i;
    for (i = 0; i < num_students; i++)
        display_student(students[i]);
}
```
Getting the student data

```c
int get_student_data(student_type students[10])
{
    int i, num_students, check;
    char name[40],
    printf("How many students do you want to enter? (<= 11): ");
    scanf("%d", &num_students);
    if (num_students > 10)
    {
        printf("Unsorry. Too many students. Exiting program.\n");
        exit(0);
    }
    for (i = 0; i < num_students; i++)
    {
        printf("Enter student's last name (type \'\' to quit): ");
        scanf("%s", name);
        if (name[0] == '\0')
            break;
        else
            strcpy(students[i].family_name, name);
        printf("Enter student's first name: ");
        scanf("%s", students[i].given_name);
        printf("Enter student's year of birth: ");
        scanf("%d", &students[i].year_of_birth);
        printf("Enter student's course code: ");
        scanf("%d", &students[i].course_code);
        check++;
    }
    if (check != num_students)
        num_students = check;
    return num_students;
}
```

Main program

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

// defining data type 'student_type'
typedef struct long_student_type
{
    char family_name[40];
    char given_name[40];
    int year_of_birth;
    int course_code;
} student_type;

// function prototypes
void display_student(student_type student);
void display_all_students(student_type students[10], int num_students);
int get_student_data(student_type students[10]);

int main(void)
{
    int num_students;
    student_type students[10];

    num_students = get_student_data(students);
    display_all_students(students, num_students);
    return 0;
}
```
Input and output of functions

• So far we have seen how to get functions to return a calculated value
  – Passing by value
  – When a variable is passed to a function a copy is made

• When we want the function to return more than one calculated value we have used an array
  – Passing by reference (but disguised!)

• What are variables (inside the computer)?
  – Memory addresses and contents

Example: Passing by Value, part 1

```c
#include <stdio.h>

int MaxOfArrayByValue(int array[100], int num_items);

/* returns the maximum value in an integer array
   Demonstrates passing-by-value */

int MaxOfArrayByValue(int array[100], int num_items)
{
    int i, max;
    max = array[0];
    for (i = 1; i < num_items; i++)
    {
        if (array[i] > max)
            max = array[i];
    }
    return max;
}
```
Example: Passing by Value, part 2

```c
int main(void)
{
    int my_array[100] = {-10, 12, 7, -5, 14};
    int max_by_value, num_items = 5;
    max_by_value = MaxOfArrayByValue(my_array, num_items);
    printf("Maximum item in array is %d (passing_by_value)\n", max_by_value);
    return 0;
}
```

Example: Passing by Reference, part 1

```c
#include <stdio.h>

void MaxOfArrayByReference(int array[100], int num_items, int *maximum);

/* returns the maximum value in an integer array
 * Demonstrates passing-by-reference
 */
void MaxOfArrayByReference(int array[100], int num_items, int *maximum)
{
    int i;
    *maximum = array[0];
    for (i = 1; i < num_items; i++)
    {
        if (array[i] > *maximum)
            *maximum = array[i];
    }
}
Example: Passing by Reference, part 2

```c
int main(void)
{
    int my_array[100] = { -10, 22, 7, -5, 14};
    int max_by_ref, num_items = 5;
    MaxOfArrayByReference(my_array, num_items, &max_by_ref);
    printf("Maximum item in array is %d (passing_by_reference)\n", max_by_ref);
    return 0;
}
```

Passing by Reference: under the hood

```c
void MaxOfArrayByReference(int array[100], int num_items, int *maximum)
{
    int i;
    *maximum = array[0];
    for (i = 1; i < num_items; i++)
    {
        if (array[i] > *maximum)
            *maximum = array[i];
    }
}
```

This says. Set the contents of the variable whose address is *maximum to ...
Passing by Reference: The call

MaxOfArrayByReference(my_array, num_items, &max_by_ref);

This says. Pass the address in memory of the variable max_by_ref

Pointers

• A pointer is a variable that holds a memory address (usually the address of another variable)
  e.g. type *name;
• This tells the compiler that name is a pointer
• Where
  – type is any valid C data type
  – name is the name of the pointer variable.
  – Type defines what type of variables the pointer can point to.
• Pointers should be initialized with the address of a variable or memory location (otherwise your program will probably crash)
& (address-of) and * (contents-of)

- int a=1, b=2, *p;

- p=&a; //p is assigned the address of a
- Pointer variable p now points to a

- b=*p; //b is assigned the value pointed to by p
- This is equivalent to
- b=a;

Pointers have types

- Pointers point to specific data types
  int *p; double *x; char *r;
- Consider:
  int *p, fred;
  char *r, c;
  r=&c; //This is OK
  r=&fred; //This is NOT OK, as fred
    //is an integer pointer
  p=&fred; // This is OK
Remember this last week?

- Suppose we want to write a function that allows us to correct the contents of NewReader? Using the passing-by-reference idea.

```c
#include <stdio.h>

struct READERINFO
{
    char lastname[30];
    char initial;
    int books_out;
    double fines_due;
};

typedef struct READERINFO READER;

int main(void)
{
    READER NewReader = {"Miller", 'J', 2, 2.25};
    printf("Lastname is %s\n", NewReader.lastname);
    printf("Initial is %c\n", NewReader.initial);
    printf("Number of books borrowed %d\n", NewReader.books_out);
    printf("Fines due %.2f\n", NewReader.fines_due);
    return 0;
}
```

Passing structures by reference

```c
void correct_reader(READER *r)
{
    int glastname, ginitial, gbooks_out, gfines_due;
    char glastname[30], ginitial;
    int gbooks_out;
    double gfines_due;

    printf("Do you want to correct the lastname? ");
    scanf("%d", &glastname);
    if (glastname)
    {
        printf("What is the correct lastname? ");
        scanf("%s", lastname);
        strcpy((r).lastname, lastname);
    }
    printf("Do you want to correct the initial? ");
    scanf("%d", &ginitial);
    if (ginitial)
    {
        printf("What is the correct initial? ");
        scanf("%s", &ginitial);
        (r).initial = ginitial;
    }
}
```

And so on...
(*some).thing is the same as some -> thing when some points to a structure

```c
void new_correct_reader(READER *r)
{
    int qlastname, qinitial, qbooks_out, qfines_due;
    char Lastname[30], initial;
    int books_out;
    double fines_due;

    printf("Do you want to correct the lastname? ");
    scanf("%d", &qlastname);
    if (qlastname)
    {
        printf("What is the correct lastname? ");
        scanf("%s", lastname);
        strcpy(r->lastname, lastname);
    }
    printf("Do you want to correct the initial? ");
    scanf("%d", &qinitial);
    if (qinitial)
    {
        printf("What is the correct initial? ");
        scanf("%c", &initial);
        r->initial = initial;
    }
}
```

Summary

- **Passing by reference**
  - A new way of communicating with functions
  - Uses addresses of variables
  - Introduced the address-of operator &
  - Introduced the contents-of operator *
- **Pointers are variables that hold the address of another variable of a particular type**
- **Passing structures by reference**
  - `r -> fieldname` is equivalent to `(*r).fieldname`
- **NEXT WEEK:**
  - Arrays and pointers
  - Variable dimension arrays
  - Some other aspects of pointers