

C Programming

Learning Package 7

Pointers and Arrays

Dr Scott Turner

School of Science and Technology

University of Northampton



**Introduction**

In this Learning Package, we are going to look at 'variables' that can store than one item, and 'variables' that can store where in memory another variable is.

**Learning outcomes**

At the end of this Learning Package, the reader should be able to:

* State what an array;
* State what a pointer is;
* Implement programs that use arrays and pointers;
* Implement single and multidimensional arrays

**Study guide**

Session 1:

Section 7.1

SAQ 7.1

Task 7.1

Section 7.2

SAQ 7.2 and 7.3

Task 7.2

Session 2:

Section 7.3

SAQ 7.4 and 7.5

Task 7.3

Section 7.4

Task 7.4

Session 3:

Section 7.5

Section 7.6

Session 4:

Exercises

* 1. **Introduction to pointers**

Read from page 166 from *8.1 introduction* to the end of *8.2 pointers* of the module textbook.

SAQ 7.1 Fill in the missing words

1. A **p r** is a symbolic **a s** in the computer's **m y**.
2. If we have **v** called example, then **&** is its address.
3. The **a s** of a variable can be assigned to a **p r** variable.
4. The **i n o r** is used to gain access to the value stored at a particular **a s**.
5. Each pointer is associated with a particular **d a t e**, it is not permissible to use a **p r** of one **t e** to point to a **v e** of a different **t e**.

Task 7.1

Type in the program below and execute it, then put comments within the code to explain what it does.

#include <stdio.h>

#include <stdlib.h> main()

{

int \*p1,\*p2; int a,b,c; a=1;

b=4;

printf("\na=%d b=%d",a,b); p1=&a;

p2=&b;

printf("\n%Address of a=%d Address of b=%d\n",p1,p2); c=\*p1;

a=\*p2;

\*p2=c;

printf("\na=%d b=%d",a,b);

printf("\n%Address of a=%d Address of b=%d\n",p1,p2); system("PAUSE");

return 0;

}

* 1. **Pointers and Functions**

Starting at page 169 of the module textbook, read the section labelled *8.3 pointers and functions - call by reference*.

SAQ 7.2

Fill in the missing words

In the method of 'call by **v e**' information is passed to and from a

**f n** by the values in the **a t l t**. In the method of 'call by **r e**' the address of a variable is passed into the **f n** via a **p r** within the **a t l t**.

SAQ 7.3

1. Which of the function could be used to swap the contents of variables x and y which are local to another function?
	1. int hello(int \*x, int \*y)
	2. int hello(int x, int y)
2. Referring to program 8.2 (page 171)
	1. What does the line \***a=\***b; mean?
	2. What does the line temp=\*a; mean?

Task 7.2

Type in and execute program 8.2 (page 171) and add your own comments. Write your comments so that somebody new to the program can understand what a line or block of code does from your comments alone.

* 1. **Arrays and pointers**

Read on pages 172- 179 the sections labelled *8.4 Arrays* and *8.5 Arrays and Pointers* of the module textbook.

SAQ 7.4

Fill in the missing words:

1. An array can be thought of as a **v e** which is **i d** so as to refer to successive **e s**.
2. An array is declared by using **s e b s**, which may be e y, or enclosing a **c t** defining the **n r** of **e s** in the array.
3. In C the first **e t** of an array is defined as **e t \_**.
4. An array, as in a **p r**, is a **s c** representation of an

**a s**. For an array is a **p r** that remains **c t** and **p ts** to the **s t** of the array (the **f t** element of the array, element **\_**).

1. Once a **p r** is **a d** to an element of an array it can be used to **i x** the array. This means the **p r** can be made to **p t** to any element of the array by increasing the value in the **p r**.
2. **P r a ic** works in units that match the size of the **d a t e** they are defined to point to. Therefore, if a **p r** is defined as a double pointer when one is added to the pointer, the value stored does not increase by 1 but by the **n r** of **b s** needed to store a double value.

SAQ 7.5

1. For the array definition float val\_1[]={2.5,3.4,2.44,45.6} what is stored at the following array elements:
	1. val\_1[0]
	2. val\_1[1]
	3. val\_1[4]
2. Using the code below

int y[4] int \*p1;

y[0]=3;y[1]=2;y[2]=1;y[3]=0;

p1=y;

printf("\n\nValue=%d Value=%d \nAddress=%d Address=%d\n",y[0],\*p1,&y[0],p1);

p1=p1+1;

printf("\n\nValue=%d Value=%d \nAddress=%d Address=%d\n",y[0],\*p1,&y[0],p1);

p1++;

printf("\n\nValue=%d Value=%d \nAddress=%d Address=%d\n",y[0],\*p1,&y[0],p1);

p1--;

printf("\n\nValue=%d Value=%d \nAddress=%d Address=%d\n",y[0],\*p1,&y[0],p1);

Answer the following questions:

* 1. What changes in address would you expect?
	2. What does p1++; do?
	3. What does p1=p1+1; do?
	4. What does p1--; do?

Task 7.3

1. Type in and execute program 8.3 (page 177).
2. Alter program 8.3 to use floating-point numbers instead of integers.
3. Make the code in SAQ 7.5(b) into an executable program.
4. Alter the program in (c) to use characters instead of integers.
	1. **Arrays and Functions**

Read on pages 179-187 the section labelled *8.6 Arrays and functions* of the module textbook.

Task 7.4

1. Type in and execute the program below:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define SIZE 100

#define MAXDBLE 1.0e37

#define MINDBLE -1.0e38

void maximum(float a[], int n, float \*max), minimum(float a[], int n, float \*min),

output\_result(float av, float max, float min, float std, int n); float mean(float a[], int n),

st\_deviation(float a[], float mean, int n); int input\_data(float a[]);

int main()

{

float data[SIZE], max=MINDBLE, min=MAXDBLE,av,std; int n;

printf("\nPlease enter the data seperated by commas.\n"); printf("Use an \* followed by a newline to terminate\n\n"); n=input\_data(data)+1;

if (n>0)

{

av=mean(data,n); maximum(data,n,&max); minimum(data,n,&min); std=st\_deviation(data,av,n); output\_result(av,max,min,std,n);

}

else printf("\n\nError in data entry\n"); system("PAUSE");

return(0);

}

int input\_data(float a[])

{

int i=-1; float x;

while (scanf("%f",&x)==1)

{

i++; if(i>=SIZE)

{

}

else

}

printf("\n\*\*too many numbers entered\*\*"); return(-1);

a[i]=x;

return(i);

}

float mean(float a[], int n)

{

int i;

float total=0; printf("\n\nn=%d",n); for (i=0;i<n;i++){

total+=a[i];

printf("\n\na[%d]=%lf av=%lf",i,a[i],total);} return(total/n);

}

void maximum(float a[], int n, float \*max)

{

int i;

for (i=0;i<n;i++) if (a[i]>\*max)

\*max=a[i];

}

void minimum(float a[], int n, float \*min)

{

int i;

for (i=0;i<n;i++) if (a[i]<\*min)

\*min=a[i];

}

void output\_result(float av, float max, float min, float std, int n)

{

printf("\n The statistics are as follows:\n\n"); printf("\t\t There are %d data values\n",n); printf("\t\t Average:\t\t%f\n",av);

printf("\t\t Maximum:\t\t%f\n",max); printf("\t\t Minimum:\t\t%f\n",min); printf("\t\t Standard deviation:\t%f\n",std);

}

float st\_deviation(float a[], float mean, int n)

{

int i;

float sum=0,st\_dev; for (i=0;i<n;i++)

sum+=(a[i]-mean)\*(a[i]-mean); st\_dev=sqrt(sum/n); return(st\_dev);

}

1. What did this program do?
2. Add comments to the program to explain what the program does. You do not need to comment every line, put enough comments in to the code to explain clearly what the program. For example above a function definition you might put in a comment that explains what the function does, and within the function definition add comments where you feel an explanation would be helpful.
	1. **Strings**

Read pages 187-197 the sections labelled *8.7 Strings, 8.8 String Library functions,* and *8.9 Arrays of Strings* of the module textbook.

SAQ 7.6

Fill in the missing words:

1. A string in C consists of a consecutive collection of **c rs** ended with the NULL **c r** ('\**\_**’). A string can also be though of as one- dimensional **a y** of **c s**.
2. String **h g** functions can be found in the **h r** file **s .h**. The functions either **t t** strings or **m e** the string.
3. Ragged arrays have strings of **u n l s** in **o\_e a y**.

Task 7.5

Type in and execute program 8.9 (you will need to use also the functions in programs 8.5 to 8.8).

* 1. **Multi-dimensional Arrays and pointers**

Read pages 197-200 sections labelled *8.10 Two dimensional Arrays*, and *8.11 More on Pointers* of the module textbook.

**Exercises**

1. Work through section labelled 8.12 Using Arrays - the bridge tutor (pages 201-203 of the module textbook).
2. Do the following exercises from pages 204-205 of the module textbook: 1, 2, 4, 5, 6, 7, 8, 9, 10, 11.

**SAQ Answers**

SAQ 7.1

1. A **pointer** is a symbolic **address** in the computer's **memory**.
2. If we have **variable** called example, then **&example** is its address.
3. The **address** of a variable can be assigned to a **pointer** variable.
4. The **indirection operator** is used to gain access to the value stored at a particular **address**.
5. Each pointer is associated with a particular **data type**, it is not permissible to use a **pointer** of one **type** to point to a **variable** of a different **type**.

SAQ 7.2

In the method of 'call by **value**' information is passed to and from a **function** by the values in the **argument list**. In the method of 'call by **reference**' the address of a variable is passed into the **function** via a **pointer** within the **argument list**.

SAQ 7.3

(a)(i) int hello(int \*x, int \*y)

(b)

(i)The contents of address pointed by the pointer b are copied into the address pointed by the pointer a. So what was in the address pointed to by pointer b is now also at the address pointed to by pointer a.

(ii) The contents of address pointed by pointer a are copied into the integer variable temp.

SAQ 7.4

Filling the missing words:

1. An array can be thought of as a **variable** that is **indexed** to refer to successive **elements**.
2. An array is declared by using **square brackets** which may be **empty** or enclosing a **constant** defining the **number** of **elements** in the array.
3. In C the first **element** of an array is defined as **element 0**.
4. An array, as in a **pointer**, is a **symbolic** representation of an **address**. For an array is a **pointer** that remains **constant** and **points** to the **start** of the array (the **first** element of the array, element **0**).
5. Once a **pointer** is **assigned** to an element of an array it can be used to **index** the array. This means the **pointer** can be made to **point** to any element of the array by increasing the value in the **pointer**.
6. **Pointer arithmetic** works in units that match the size of the **data type** they are defined to point to. So if a **pointer** is defined as a double pointer when one is added to the pointer, the value stored does not increase by 1 but by the **number** of **bytes** needed to store a double value.

SAQ 7.5

(a) For the array definition float val\_1[]={2.5,3.4,2.44,45.6} what is stored at the following array elements:

(i) 2.5

1. 3.4
2. val\_1[4] is not an element in the array val\_1 so it will not have had a value allocated to it by the program, but a value might be at the memory location that would have been used to store it. We can not predict easily what is in there so avoid this.

(b)

1. The address of y[0] and the address point to by p1 will the same initially. In the next printf line the address are different, the address of y[0] is the same but the address point to by p1 is different (increased) by 4. In the next printf line the address are different the address of y[0] is the same as before but the address point to by p1 is further increased by 4. The final printf line the address are different the address of y[0] is the same but the address point to by p1 is this time decreased by 4, this line should have the same effect as was seen by the second printf line.
2. Increases the address the pointer points to by enough to store another integer (in this case), it does not add 1 to value in the pointer.
3. as (i)
4. Decreases the address the pointer points to by enough to store another integer (in this case).

SAQ 7.6

Fill in the missing words:

1. A string in C consists of a consecutive collection of **characters** ended with the NULL **character** ('\**0**'). A string can also be though of as one- dimensional **array** of **characters**.
2. String **handling** functions can be found in the **header** file **string.h**. The functions either **test** strings or **manipulate** the string.
3. Ragged arrays have strings of **uneven lengths** in **one array**.

Selected Tasks Task 7.1

The program swaps the contents of a and b around indirectly. The address of a and b do not change but the values stored at these addresses does.

#include <stdio.h>

#include <stdlib.h> main()

{

int \*p1,\*p2; /\*p1 and p2 are integer point variables\*/ int a,b,c; /\*a,b,c are integer variables\*/

a=1;

b=4; /\*initially a =1 and b=4 printf("\na=%d b=%d",a,b);

p1=&a; /\* p1 and p2 store the addresses of a and b\*/ p2=&b;

/\*display the addresses of a and b\*/ printf("\n%Address of a=%d Address of b=%d\n",p1,p2);

c=\*p1; /\*the contents of the address pointed to by p1\*/

/\*are assigned to the integer variable c p1 points to\*/

/\*the variable a\*/

a=\*p2; /\*the contents of the address pointed to by p2\*/

/\*are assigned to the integer variable c p2 points to\*/

/\*the variable b\*/

\*p2=c; /\*the contents of variable c are assigned to the address\*/

/\*pointed to by p2\*/ printf("\na=%d b=%d",a,b);

printf("\n%Address of a=%d Address of b=%d\n",p1,p2); system("PAUSE");

return 0;

}