Programming in C

Table of Contents

[*1* *Part A* 2](#_Toc31200366)

[1.1 Prerequisites 2](#_Toc31200367)

[1.2 Pointer arithmetic on array 2](#_Toc31200368)

[1.3 Generic array add 3](#_Toc31200369)

[1.4 Generic array add (part 2) 4](#_Toc31200370)

[1.5 Generic array (part 3) 4](#_Toc31200371)

[1.6 Fifo buffer 4](#_Toc31200372)

[1.7 Mini quiz: pointers & arrays 4](#_Toc31200373)

# Part A

## Prerequisites

Please study the modules about arrays.

## Pointer arithmetic on array

Write a small program that declares an array of 5 integers and declare a pointer that points to the last element of the array. Write a loop that traverses the array via the pointer in reverse direction:

int array[] = {10,11,12,13,14};

int\* p = &array[4];

// Declare a pointer to the last element of array.

for(int i = 0; i < 5; i++);

{

Printf(“array value:%d”, \*p);

p--;

// Print the value at the pointer value

// Set the pointer to the previous array value

}

## Generic array add

I want to have a generic function that can add all values in an array of doubles and return the result. The prototype for this function could be:

double arrayAdd(double\* array);

This prototype has a problem. Please improve the prototype and implement the function. Test if it works properly.

Double aArray[10] {};

double Result = arrayAdd(&aArray[0], double addToArray);

If (result == addToArray) {printf(“Gelukt/n”);}

Else {printf(“Niet Gelukt/n”);}

Double arrayAdd(double\* p, double val) {

double arrayLocation;

P = P + arrayLocation ;

\*p = val ;

arrayLocation = arrayLocation + 8  ;

return val ;

}

## Generic array add (part 2)

Another way of writing the function in the previous assignment would be by returning the result via a parameter. A reason to do this is if you need to return more than one thing. For example: what would arrayAdd() do if it is called with a NULL pointer? It has no way of telling the caller that an error occurred.

Please improve the arrayAdd() function:

* The function will get a new name: arrayAddParameter()
* Return type is int: it returns 0 if the result is computed correctly, -1 if an error occurs
* The function will get a new parameter named: result. Please define the variable type yourself.

Double aArray[10] {};

Result = 1;

arrayAddParameter(&aArray[0], double addToArray, int result);

int arrayAddParameter(double\* p, double val) {

double arrayLocation;

P = P + arrayLocation ;

\*p = val ;

arrayLocation = arrayLocation + 8  ;

int\* check = (int\*) calloc (result,sizeof(int));

if (check == NULL) {return -1;}

else {return 0;}

}

## Generic array functions

### Generic array functions

Write a function that returns the average of an array of int’s  
It must be possible to use the function to calculate the average of any given array with known length.

Write testcases.

Int array[10] {};

Int\* p = array[0];

Int gemiddeld;

int aantal = sizeof(array) / sizeof(array[0]);

for (int I = 0; I < aantal; i++) {

gemiddeld += \*p;

p++;

}

Gemiddeld /= aantal;

### Generic array (part 4

Write a function that finds a value in an array of int’s. It must be possible to pass the array to the function.

Define an appropriate prototype for this function.

Write testcases.

Int array[10] {};

Int\* p = array[0];

Int findValue;

Int arrayNumber;

int aantal = sizeof(array) / sizeof(array[0]);

for (int I = 0; I < aantal; i++) {

if (array[i] == findValue) {

arrayNumber = i;

}

}

## Fifo buffer

This is a slightly complexer challenge.

Note: please first study the buffer. E.g. see <https://helloacm.com/how-do-you-design-a-circular-fifo-buffer-queue-in-c/>

More important note: the information in the provided link does describe the functionality of the fifo buffer correctly. However: the implementation has (design) errors in it, so you cannot use them. You can use them for inspiration only.

Implement read and write functions of a FIFO buffer that holds values of the type “int”.

If the buffer is full the oldest element will be overwritten.

The prototype for the read function is as follows:

// pre: readvalue not null

// post : function returnvalue:

// 0 : read succeeded

// -1: readvalue null or buffer empty

int fiforead(int\* readvalue);

Please define a prototype for the write function taking in to account the following

* Return value 0 if element has been added.
* Return value 1 if element has been added to the buffer, but the buffer was already full
* Return value -1 if the element has not been added.

Write testcases and perform the tests.

Use the function you have developed earlier to calculate the average of the fifo buffer.

Note: you can use this function in ES assignments. It is up to you to discover where you can use them.

### Fifo buffer

This challenge will combine structs and the previous assignment.

In the previous challenge you probably developed a function that works on only one fixed array.

Now you are asked to make the function reusable so it can be used with many different instances of the FIFO buffer data.

Step 1:

* Wrap the variables that you use to keep track of head, tail and the array itself in a struct.
* Rewrite your FIFO functions to use the data belonging to this struct.

Step 2:

* Now rewrite this functions so they can use any instance of this struct.
* Write testcases

Step 3 (advanced)

* Add functionpointers to the struct that point to the read and write functions
* Use these functionpointers in your testcases
* Question: if you look at this struct. What other programming concept does it resemble?

## Mini quiz: pointers & arrays

Please try to predict the following questions theoretically. You may, of course, check your answers afterwards in a test program.

The following variables are given:

int a[6] = { 3, 6, -3, 1, 8, 3 };

int i = 3;

int j = 8;

int\* p = a;

int\* q;

int\* r = &a[1];

Please describe if the following statements are correct, and if so what the statement's result is.

1. i = \*(p + 5); klopt p wijst naar eerste 3 en gaat 5 plekken naar rechts wat ook 3 is ( in de array )
2. j = sizeof (a); is 24 aantal ints x 4 (een int is 4 bits) dus klopt niet
3. j = a[a[a[0]]]; niet klopt is 6
4. r[2] = p[2] + a[1]; de r[2] = 1 omdat hij wijst nar 6 plus 2 naar rechts is 1. De p[2] wijst naar 3 en gaat 2 plekken naar rechts dus -3 en A[1] = 6 6 +-3 (6 -3) = 3 dus klopt niet
5. \*q += \*p \* \*r; das een mooie error (q wijst nergens naar dus kan niet +=) maar \*p \* \*r = 3 x 6 = 18