

# C: an introduction

## Arrays - basics

1

## Why arrays?

```

1/*
2average_grade_bruteforce.c
3Averaging ten grades - storing values the hard way
4taken from I. Horton: Beginning C 5th Ed
5*/
6
7#include <stdio.h>
8int main(void)
9{
10 int grade0 = 0, grade1 = 0, grade2 = 0, grade3 = 0, grade4 = 0;
11 int grade5 = 0, grade6 = 0, grade7 = 0, grade8 = 0, grade9 = 0;
12 long sum = 0; // Sum of the grades
13 float average = 0.0f; // Average of the grades
14
15 // Read the ten grades to be averaged
16 printf("Enter the first five grades,\n");
17 printf("use a space or press Enter between each number.\n");
18 scanf("%d%d%d%d%d", &grade0, &grade1, &grade2, &grade3, &grade4);
19 printf("Enter the last five numbers in the same manner.\n");
20 scanf("%d%d%d%d%d", &grade5, &grade6, &grade7, &grade8, &grade9);
21
22 // Calculate the average
23 sum = grade0 + grade1 + grade2 + grade3 + grade4 +
24 grade5 + grade6 + grade7 + grade8 + grade9;
25 average = (float)sum/10.0f;
26 printf("\nAverage of the ten grades entered is: %.2f\n", average);
27 return 0;
28}

```

```

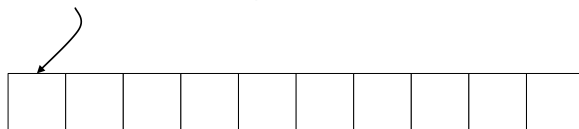
1/*
2average_grade_array.c
3Averaging ten grades - storing the values the easy way
4taken from I. Horton: Beginning C 5th Ed
5*/
6
7#include <stdio.h>
8
9int
10main (void)
11{
12 int grades[10]; // Array storing 10 values
13 unsigned int count = 10; // Number of values to be read
14 long sum = 0; // Sum of the numbers
15 float average = 0.0f; // Average of the numbers
16
17 printf ("\nEnter the 10 grades:\n"); // Prompt for the input
18 // Read the ten numbers to be averaged
19 for (unsigned int i = 0; i < count; ++i)
20 {
21     printf ("%2u ", i + 1);
22     scanf ("%d", &grades[i]); // Read a grade
23     sum += grades[i]; // Add it to sum
24 }
25
26 // calculate the average
27 average = (float) sum / count; // Calculate the average
28 printf ("\nAverage of the ten grades entered is: %.2f\n", average);
29 return 0;
30}

```

2

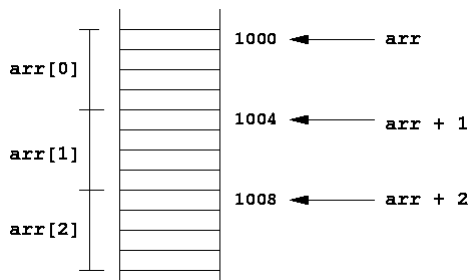
## Array basics

- Array: a block of memory that holds one or more objects of a given type (consecutive memory addresses).
- Array: store multiple data with common characteristics
  - Same name
  - Same type
  - Accessed by specifying subscript(s) (indexing starts at index 0!)



## Array basics

- Example  
`int arr[ 10 ] ;`



## Array basics

- Declare an array:
  - Declare **name**
  - Declare the **type** of elements
  - Declare the **length** (**maximum** number of elements).

```
double empty[0]; /* Error: cannot be empty */
int an_array[10]; /* allocate for 10 ints. */

a = an_array[0]; /* first element */
b = an_array[9]; /* last element */
c = an_array[10]; /* Error: but will compile */
```

## Array basics

- Elements of an array are stored at consecutive locations in memory (continuous memory)
  - Easy access
  - Difficult for large arrays
- Access to arrays is performed **without boundary checking**. Boundary checks must be applied explicitly by the programmer.
- First element of array is stored at index 0
- Array identifier alone is a variable storing the address of the first element of the array

# Array Initialisation

- An array is not initialised by default.
- Can explicitly initialise an array using an initialiser list enclosed in braces {}.

```
int days[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

- If the number of elements in the initialiser list is less than the size of the array, the remainder of an array is initialised to zero.

```
int local_array[50] = {0};
```

- If the number of elements is greater, it is an error.
- An array with an initialiser list may be sized automatically by the compiler.

```
int days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

8

KU LEUVEN

8

```
1#include <stdio.h>
2/*
3array_init.c
4based on http://gribblelab.org/cbootcamp/6_Complex_Data_Types.html
5*/
6
7int main ()
8{
9    int grades[5] = {11, 9, 14, 15, 13};
10    // int grades[5] = {11, 9, 14, 15, 13, 12}; // error?
11    int grades2[5] = {[0]=1, [2]=3, [4]=5};
12    int local_arr[10] = {-1};
13    int days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
14
15    int i;
16
17    // what are the initial values?
18    for (i=0; i<5; i++) {
19        printf("grades[%d]=%d\n", i, grades[i]);
20    }
21
22    // what are the initial values?
23    for (i=0; i<5; i++) {
24        printf("grades2[%d]=%d\n", i, grades2[i]);
25    }
26
27    // out of the boundary?
28    printf("grades[5]=%d\n", grades[5]);
29    printf("grades[100]=%d\n", grades[100]);
30
31
32    // what are the initial values?
33    for (i=0; i<10; i++) {
34        printf("local_arr[%d]=%d\n", i, local_arr[i]);
35    }
36
37    // what are the initial values?
38    for (i=0; i<12; i++) {
39        printf("days[%d]=%d\n", i, days[i]);
40    }
41    return 0;
42}
```

```
frankvp@CRD-L-08004:~/Arrays$ gcc array_init.c -o array_init
frankvp@CRD-L-08004:~/Arrays$ ./array_init
grades[0]=11
grades[1]=9
grades[2]=14
grades[3]=15
grades[4]=13
grades2[0]=1
grades2[1]=0
grades2[2]=3
grades2[3]=0
grades2[4]=5
grades[5]=0
grades[100]=1269202072
local_arr[0]=-1
local_arr[1]=0
local_arr[2]=0
local_arr[3]=0
local_arr[4]=0
local_arr[5]=0
local_arr[6]=0
local_arr[7]=0
local_arr[8]=0
local_arr[9]=0
days[0]=31
days[1]=28
days[2]=31
days[3]=30
days[4]=31
days[5]=30
days[6]=31
days[7]=31
days[8]=30
days[9]=31
days[10]=30
days[11]=31
```

9

```

1 #include <stdio.h>
2 /*
3 array_bounds.c
4 based on http://gribblelab.org/cbootcamp/6_Complex_Data_Types.html
5 */
6
7 int main ()
8 {
9     int grades[5];
10    int i;
11
12    // what are the initial values?
13    for (i=0; i<5; i++) {
14        printf("grades[%d]=%d\n", i, grades[i]);
15    }
16
17    // out of the boundary?
18    printf("grades[5]=%d\n", grades[5]);
19    printf("grades[10]=%d\n", grades[10]);
20
21    // assign a value
22    for (i=0; i<5; i++) {
23        grades[i]=i;
24    }
25    for (i=0; i<5; i++) {
26        printf("grades[%d]=%d\n", i, grades[i]);
27    }
28
29    return 0;
30 }

```

```

frankvp@CRD-L-08004:~/Arrays$ gcc array_bounds.c -o array_bounds
frankvp@CRD-L-08004:~/Arrays$ ./array_bounds
grades[0]=0
grades[1]=0
grades[2]=1074737280
grades[3]=21967
grades[4]=669659408
grades[5]=32765
grades[10]=1617907891
grades[0]=0
grades[1]=1
grades[2]=2
grades[3]=3
grades[4]=4
frankvp@CRD-L-08004:~/Arrays$

```

KU LEUVEN

10

```

1 /*
2 out-of-bounds array
3 */
4
5 #include <stdio.h>
6
7 int main() {
8     int brr[2] = {10, 11};
9     int arr[5] = {1, 2, 3, 4, 5};
10
11
12     printf("address arr = %p \n", arr);
13     printf("address brr = %p \n", brr);
14
15     // Correct access within bounds
16     for (int i = 0; i < 5; i++) {
17         printf("arr[%d] = %d\n", i, arr[i]);
18     }
19     // Correct access within bounds
20     for (int i = 0; i < 2; i++) {
21         printf("brr[%d] = %d\n", i, brr[i]);
22     }
23
24     // Out-of-bounds access
25     brr[5] = 100; // This is out-of-bounds since valid indices are 0 to 4
26     for (int i = 0; i < 5; i++) {
27         printf("arr[%d] = %d\n", i, arr[i]);
28     }
29
30     return 0;
31 }

```

```

frankvp@CRD-L-10275:~/training-c/Arrays$ gcc -o array_bounds_2 array_bounds_2.c
frankvp@CRD-L-10275:~/training-c/Arrays$ ./array_bounds_2
address arr = 0x7fffa52957b0
address brr = 0x7fffa52957a8
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
brr[0] = 10
brr[1] = 11
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 100
arr[4] = 5
frankvp@CRD-L-10275:~/training-c/Arrays$

```

KU LEUVEN

11

## Assigning / Getting values

- Assignment of values:

```
array_x[1] = 61; /* 2nd element gets 61*/
```

```
array_y[2] = 1.14;
```

- Accessing values from arrays:

```
val1 = array_x[2];
```

```
val2 = array_y[0];
```

- Accessing variable `array_x[n]` = (n+1)th element!
- Tip: for-loops are ideal for processing array elements

## Arrays and pointers

- Arrays and Pointers are strongly related.
- Whenever an array name appears in an expression, it is automatically converted to a pointer to its first element.

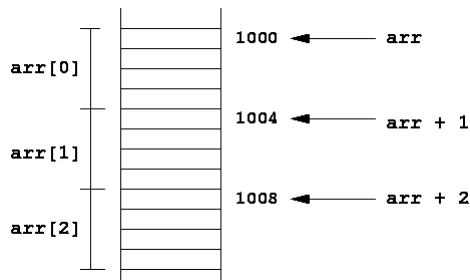
```
1#include <stdio.h>
2/*
3array_pointer_1.c
4based on Computer programming in C for beginners
5*/
6
7int main ()
8{
9    int grades[5]={10, 12, 11, 16, 7};
10
11    printf("The address contained in grades is %p \n", grades);
12    printf("The address of grades[0] is %p \n", &grades[0]);
13
14    return 0;
15}
```

```
frankvp@CRD-L-08004:~/Arrays$ gcc array_pointer_1.c -o array_pointer_1
frankvp@CRD-L-08004:~/Arrays$ ./array_pointer_1
The address contained in grades is 0x7ffea01920c0
The address of grades[0] is 0x7ffea01920c0
frankvp@CRD-L-08004:~/Arrays$
```

## Arrays and pointers

- Array can be treated as a constant pointer that points to the first element in the array

- `int arr[ 10 ] ;`



## Pointers and Arrays are Different

- An array name is not a variable – its value cannot be changed.  

```
int a1[10], a2[10];  
int *pa = a1;  
a1 = a2; /* Error: won't compile. */  
a1++;   /* Error: won't compile. */
```
- An array name always refers to the beginning of a segment of allocated memory.  
A pointer may point anywhere (e.g., to allocated memory, to NULL, to free memory, to invalid locations).
- The size of an array is equal to the number of characters of memory allocated. The size of a pointer is just the size of the pointer-type.
- Pointers and array names may be used interchangeably for array indexing operations.

```

1#include <stdio.h>
2/*
3array_pointer_2.c
4based on Computer programming in C for beginners
5*/
6
7int main ()
8{
9    int grades[5]={10, 12, 11, 16, 7};
10   int points[5];
11   int * pa, * pb;
12   int sg, spa;
13
14   printf("The address contained in grades is %p \n", grades);
15   printf("The address of grades[0] is %p \n", &grades[0]);
16
17   pa = grades;
18   printf("The address contained in pa is %p \n", pa);
19
20   // points = grades; /* will this compile? */
21
22   grades[4] = 6; /* Equivalent indexes. */
23   printf("grades[4] updated %d \n", grades[4]);
24   pa[4] = 6;
25   printf("grades[4] updated %d \n", pa[4]);
26   *(grades + 4) = 6;
27   printf("grades[4] updated %d \n", *(grades+4));
28   *(pa + 4) = 6;
29   printf("grades[4] updated %d \n", *(pa+4));
30
31   pb = &grades[1]; /* Equivalent addresses. */
32   pb = &pa[1];
33   pb = grades + 1;
34   pb = pa + 1;
35
36   sg = sizeof(grades);
37   spa = sizeof(pa);
38   printf("The size of grades is %d \n", sg);
39   printf("The size of pa is %d \n", spa);
40
41
42   return 0;
43}

```

```

frankvp@CRD-L-08004:~/Arrays$ gcc array_pointer_2.c -o array_pointer_2
frankvp@CRD-L-08004:~/Arrays$ ./array_pointer_2
The address contained in grades is 0x7ffe5727f330
The address of grades[0] is 0x7ffe5727f330
The address contained in pa is 0x7ffe5727f330
grades[4] updated 6
grades[4] updated 6
grades[4] updated 6
grades[4] updated 6
The size of grades is 20
The size of pa is 8
frankvp@CRD-L-08004:~/Arrays$

```

KU LEUVEN

18

## Pointer arithmetic

- The variable name of an array is also a pointer to its first element.
  - `a == &a[0]`
  - `a[0] == *a`
- The pointer advances/retreats by that number of elements (of the type being pointed to)
  - `a+i == &a[i]`
  - `a[i] == *(a+i)`

KU LEUVEN

19



## Passing arrays to functions

- Array names are in fact pointers!
- Actually passing an array by reference, rather than by value
  - Passing the array name only to the called function (without the brackets).
  - Pass the size of the array, the calling function knows the size of the array
- The two prototypes below are exactly equivalent.

```
int count_days(int days[]);  
int count_days(int *days);
```

## Passing arrays to functions

- Pass by reference to avoid copying large arrays.
- Pass the array size to avoid out-of-bounds access.
- Use `const` for read-only arrays to prevent modifications.
- Handle multidimensional arrays carefully.

# Passing arrays to functions

```
1#include <stdio.h>
2#include <stdlib.h>
3/*
4array_passing_1.c
5based on Computer programming in c for beginners
6*/
7void double_it(int [], int); // prototype
8int main()
9{
10 int arr[10] = {0}, n;
11 for(n=0; n<10; n++)
12 printf("The content of cell %d is initially %d \n", n, arr[n]);
13 double_it(arr, 10);
14 printf("\n\n");
15 for(n=0; n<10; n++)
16 printf("The content of cell %d is now %d \n", n, arr[n]);
17 return 0;
18 }
19
20 void double_it(int a[], int i)
21 {
22 int k = 0;
23 a[0] = 1;
24 for(k=1; k<i; k++){
25 a[k] = a[k-1] * 2;
26 }
```

```
frankvp@CRD-L-08004:~/Arrays$ gcc array_passing_1.c -o array_passing_1
frankvp@CRD-L-08004:~/Arrays$ ./array_passing_1
The content of cell 0 is initially 0
The content of cell 1 is initially 0
The content of cell 2 is initially 0
The content of cell 3 is initially 0
The content of cell 4 is initially 0
The content of cell 5 is initially 0
The content of cell 6 is initially 0
The content of cell 7 is initially 0
The content of cell 8 is initially 0
The content of cell 9 is initially 0

The content of cell 0 is now 1
The content of cell 1 is now 2
The content of cell 2 is now 4
The content of cell 3 is now 8
The content of cell 4 is now 16
The content of cell 5 is now 32
The content of cell 6 is now 64
The content of cell 7 is now 128
The content of cell 8 is now 256
The content of cell 9 is now 512
frankvp@CRD-L-08004:~/Arrays$
```

KU LEUVEN

22

```
1/*
2generate_array.c
3test function to generate an array
4*/
5
6#include<stdio.h>
7
8void generate_array(int size, int dummy[], int x);
9
10int main() {
11
12 int i;
13 int dummy1[10];
14 int dummy2[5];
15
16 generate_array(10, dummy1, 100);
17 for (i=0; i<10; i++)
18 printf("dummy1 - %d = %d \n", i, dummy1[i]);
19
20 generate_array(5, dummy2, 33);
21 for (i=0; i<5; i++)
22 printf("dummy1 - %d = %d \n", i, dummy2[i]);
23
24 return 0;
25 }
26
27 void generate_array(int size, int dummy[], int x)
28 {
29 int i;
30
31 for (i=0; i<size; i++){
32 dummy[i] = i + x;
33 }
34 }
```

```
frankvp@CRD-L-08004:~/Arrays$ gcc generate_array.c -o generate_array
frankvp@CRD-L-08004:~/Arrays$ ./generate_array
dummy1 - 0 = 100
dummy1 - 1 = 101
dummy1 - 2 = 102
dummy1 - 3 = 103
dummy1 - 4 = 104
dummy1 - 5 = 105
dummy1 - 6 = 106
dummy1 - 7 = 107
dummy1 - 8 = 108
dummy1 - 9 = 109
dummy1 - 0 = 33
dummy1 - 1 = 34
dummy1 - 2 = 35
dummy1 - 3 = 36
dummy1 - 4 = 37
frankvp@CRD-L-08004:~/Arrays$
```

KU LEUVEN

23

## Pointer passing revisited - const

- Declaring function parameters `const` indicates that the function promises not to change these values.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 /*
4  array_passing_const.c
5
6  based on https://azrael.digipen.edu/~mmead/www/Courses/CS170/Const-1.html#ARGUMENTS
7 */
8 int find_largest2(const int a[], int size);
9
10 int
11 main ()
12 {
13     int arr[10] = { 0, 1, 2, 3, 4, 4, 4, 1, 1, 1 };
14     int n;
15     int max_array;
16     const int *parr;
17
18     max_array = find_largest2(arr,10);
19     printf("largest value in array = %d \n", max_array);
20     return 0;
21 }
22
23
24 int find_largest2(const int a[], int size)
25 {
26     int i;
27     int max = a[0];
28     // a[0] = 0; /* ILLEGAL: elements are const, compiler prevents it */
29     for (i = 1; i < size; i++)
30     {
31         if (a[i] > max)
32             max = a[i];
33     }
34     // a[i] = 0; /* ILLEGAL: elements are const, compiler prevents it */
35     return max;
36 }
```

24

## Passing arrays to functions

```
3 /*
4  array_passing_2.c
5  http://www.cs.yale.edu/homes/aspnes/classes/223/examples/pointers/sumArray.c
6 */
7 void double_it(int [], int); // prototype
8 int sumArray(int n, const int *a);
9
10 int main()
11 {
12     int arr[10] = {0};
13     int n;
14     int sum_array;
15     const int * parr;
16     // put values
17     parr = arr;
18     for(n=0; n<10; n++){
19         arr[n] = n;
20         printf("The content of cell %d is %d \n", n, arr[n]);
21     }
22     // calculate the sum
23     sum_array = sumArray(10, parr);
24
25     printf("\n\n");
26     printf("The sum of the array elements is %d \n", sum_array);
27     return 0;
28 }
29
30 /* compute the sum of the first n elements of array a */
31 int
32 sumArray(int n, const int *a)
33 {
34     int i;
35     int sum;
36
37     sum = 0;
38     for(i = 0; i < n; i++) {
39         sum += a[i];
40     }
41
42     return sum;
43 }
```

```
(base) frankvp@CRD-L-08004:/mnt/c/Temp/Develop/CDev/Arrays$ gcc array_passing_2.c -Wall -o array_passing_2
The content of cell 0 is 0
The content of cell 1 is 1
The content of cell 2 is 2
The content of cell 3 is 3
The content of cell 4 is 4
The content of cell 5 is 5
The content of cell 6 is 6
The content of cell 7 is 7
The content of cell 8 is 8
The content of cell 9 is 9

The sum of the array elements is 45
(base) frankvp@CRD-L-08004:/mnt/c/Temp/Develop/CDev/Arrays$
```

- Note the use of `const` to promise that `sumArray` won't modify the contents of `a`.

25

## 2D arrays

- `a[i][j]`, *not* `a[i, j]`

- Initialize:

- Row dominant

- Use `{}`

```
int a[3][4]={
    {0, 1, 2, 3},
    {4, 5, 6, 7},
    {8, 9, 10, 11}};
```

or

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

[https://www.tutorialspoint.com/cprogramming/c\\_multi\\_dimensional\\_arrays.htm](https://www.tutorialspoint.com/cprogramming/c_multi_dimensional_arrays.htm)

KU LEUVEN

26

```
1/*
2array_2dim.c
3taken from http://gribblelab.org/cbootcamp/6_Complex_Data_Types.html
4*/
5
6#include <stdio.h>
7
8int main ()
9{
10    int grades[2][2] = {1,2,3,4}; // C is row dominant!
11    int i,j;
12    for (i=0; i<2; i++) {
13        for (j=0; j<2; j++) {
14            printf("grades[%d][%d]=%d\n", i, j, grades[i][j]);
15        }
16    }
17    return 0;
18}
```

```
frankvp@CRD-L-08004:~/Arrays$ gcc array_2dim.c -o array_2dim
frankvp@CRD-L-08004:~/Arrays$ ./array_2dim
grades[0][0]=1
grades[0][1]=2
grades[1][0]=3
grades[1][1]=4
frankvp@CRD-L-08004:~/Arrays$
```

KU LEUVEN

27

## 2D arrays and double pointers

- Fundamental notion for arrays
  - `vector[i] == *(vector + i)`
  - Content of `vector[i] == *(vector + i)`
- Apply the notion twice for 2D arrays
  - `int Arr[i][j]`
  - Arr is a 2D int array, consider it as a pointer to a pointer to an integer: `int**`
  - Consider a 2D array as an array of pointers to an array of integers.
    - `Arr[0]` address of row 0
    - `Arr[1]` address of row 1...
  - Apply fundamental notion twice
    - `*(Arr + i)[j]` (apply once for the row)
    - `*(*(Arr + i) + j)` (apply twice for the column in the vector)

## Notation

- `int (*pt)[MAX];` // one pointer to MAX ints
- `int *ptrs[MAX];` // array of pointers to int

## Tip: Matrix calculations

- multidimensional arrays to represent matrices
- better to make use of one of the pre-existing APIs for matrix algebra, rather than coding up this yourself.
- Common choices:
  - The GNU Scientific Library GSL [Vectors and Matrices](#)
  - [LAPACK](#) (and [BLAS](#)) libraries