Operating Systems

A (Soft) Introduction to C, Part III

Vassilis Markos, Mediterranean College

Week 07

Contents

1. Pointers
2. Arrays
3. Fun Time!

Pointers

Stars And C

What will this program print? (Do not execute it!)

```c
// source/stars_001.c
#include <stdio.h>

int main() {
    double x;
    printf("Enter a double: ");
    scanf("%f", &x);
    double *y = &x;
  printf("%d\n", (*y == x));
}
```

What Are * and &?

- What is `*y` in the above program at line 9?

What Are `*` and `&`?

- What is `*y` in the above program at line 9?
    - `*y` denotes the memory location of a `double` variable, `x` in our case. Such variables are called **pointers**.

What Are * and &?

- What is `*y` in the above program at line 9?
  - `*y` denotes the memory location of a `double` variable, `x` in our case. Such variables are called **pointers**.
- What is `*y` in the above program, at line 10?

What Are `*` and `&`?

- What is `*y` in the above program at line 9?
  - `*y` denotes the memory location of a `double` variable, `x` in our case. Such variables are called **pointers**.
- What is `*y` in the above program, at line 10?
  - `*y` denotes the content at the memory location stored by pointer `y`. Retrieving a pointer's pointed value is often called **dereferencing**.

What Are * and &?

- What is `*y` in the above program at line 9?
  - `*y` denotes the memory location of a `double` variable, `x` in our case. Such variables are called **pointers**.
- What is `*y` in the above program, at line 10?
  - `*y` denotes the content at the memory location stored by pointer `y`. Retrieving a pointer's pointed value is often called **dereferencing**.
- What is `&x` in the above program at line 9?

What Are * and &?

- What is `*y` in the above program at line 9?
  - `*y` denotes the memory location of a `double` variable, `x` in our case. Such variables are called **pointers**.
- What is `*y` in the above program, at line 10?
  - `*y` denotes the content at the memory location stored by pointer `y`. Retrieving a pointer's pointed value is often called **dereferencing**.
- What is `&x` in the above program at line 9?
  - `&x` indicates the memory location where the content of variable `x` is stored in the computer's memory, i.e., it is a **reference** of `x`.

Can You Predict The Output?

```c
// source/stars_002.c
#include <stdio.h>

int main() {
    int x;
    printf("Enter an int: ");
    scanf("%d", &x);
    int *y = &x;
  printf("%d\n", y);
}
```

Can You Predict The Output?

```c
// source/stars_003.c
#include <stdio.h>

int main() {
    int x;
    printf("Enter an int: ");
    scanf("%d", &x);
    int *y = &x;
  printf("%d\n", *( &x ));
}
```

Can You Predict The Output?

```c
// source/stars_004.c
#include <stdio.h>

int main() {
    int x;
    printf("Enter an int: ");
    scanf("%d", &x);
    int *y = &x;
  printf("%d\n", y);
}
```

Can You Predict The Output?

```c
// source/stars_005.c
#include <stdio.h>

int main() {
    int x;
    printf("Enter an int: ");
    scanf("%d", &x);
    int *ptrx = &x;
    (*ptrx)++;
    printf("%d\n", x);
}
```

Can You Predict The Output?

```c
// source/stars_006c.
#include <stdio.h>

int main() {
    int x, y;
    printf("Enter two ints: ");
    scanf("%d", &x);
    scanf("%d", &y);
    int *ptrx = &x;
    int *ptry = &y;
    ptry = ptrx;
    (*ptrx)--;
    ptry = &y;
  printf("%d, %d\n", x, y);
}
```

Can You Predict The Output?

```c
// source/stars_007.c
#include <stdio.h>

int main() {
    int x, *y;
    printf("Enter an int: ");
    scanf("%d", &x);
    y = &x;
    (*y)++;
  printf("%d\n", x);
}
```

A `const` Interlude

Consider the program shown right. Which of the following inserted in line 12 will raise an error?

1. `(*p1)++;`
2. `(*p2)++;`
3. `p1 = &y;`
4. `p2 = &y;`

```c
 1 // source/stars_008.c
 2 #include <stdio.h>
 3 
 4 int main() {
 5     int x, y;
 6     printf("Enter two ints: ");
 7     scanf("%d", &x);
 8     scanf("%d", &y);
 9     const int *p1 = &x;
10     int * const p2 = &x;
11     // Enter your code here...
12 }
```

Pointers To Constant Variables

In this case, pointer `p1` is pointing to a **constant** integer variable. This means that at the memory location `p1` is pointing to we can make no modifications!

```c
// source/stars_009.c
#include <stdio.h>

int main() {
    int x, y;
    printf("Enter two ints: ");
    scanf("%d", &x);
  scanf("%d", &y);
    const int *p1 = &x;
    int * const p2 = &x;
    (*p1)++;
}
```

Pointers To Constant Variables

In this case, pointer p2 is a **constant** pointer pointing to an integer variable. This means that at the memory location p2 is pointing to we can make any modifications we want to. What we can't change is the value of the pointer itself.

```
// source/stars_010.c
#include <stdio.h>

int main() {
    int x, y;
    printf("Enter two ints: ");
    scanf("%d", &x);
  scanf("%d", &y);
    const int *p1 = &x;
    int * const p2 = &x;
    (*p2)++;
}
```

All `const`?

What about this one?

```
const int * const ptr;
```

In this case:

All `const`?

What about this one?

```
const int * const ptr;
```

In this case:

- We cannot change where the pointer points to (rightmost `const`).

All `const`?

What about this one?

```
const int * const ptr;
```

In this case:

- We cannot change where the pointer points to (rightmost `const`).
- We cannot change the content of the memory location the pointer points to (leftmost `const`).

Arrays

Arrays In C++

Can you guess what the following will print?

```c
// source/arrays_001.c
#include <stdio.h>

int main() {
    int arr[3];
    arr[0] = 4;
    arr[1] = 6;
    arr[2] = -5;
    for (int i = 0; i < 3; i++) {
        printf("arr[%d] == %d\n", i, arr[i]);
    }
}
```

Array Initialisation

We can also provide array elements all at once, as follows:

```c
// source/arrays_002.c
#include <stdio.h>

int main() {
    int arr1[5] = { 4, -2, 0, 4, 6 };
    int arr2[] = { 6, 5, 7, 9 };
    printf("arr1[3] == %d\narr2[1] == %d\n", arr1[3], arr2[1]);
}
```

Dynamic Initialisation

We can also initialise the values of an array based on others' input (e.g., users, another process):

```c
// source/arrays_003.c
#include <stdio.h>

int main() {
    char arr[3];
    for (int i = 0; i < 3; i++) {
        printf("Please, enter a character: ");
        scanf(" %c", &arr[i]);
    }
    printf("%c%c%c\n", arr[0], arr[1], arr[2]);
}
```

What Will This Print?

```c
// source/arrays_004.c
#include <stdio.h>

int main() {
    char arr[3];
    for (int i = 0; i < 3; i++) {
        printf("Please, enter a character: ");
        scanf(" %c", &arr[i]);
    }
    printf("%c%c%c\n", arr[0], arr[1], arr[2]);
}
```

Dynamic Initialisation And Array Size

The above must have printed something along the following lines:

```
arrays_004.c: In function ‘int main()’:
arrays_004.c:6:10: error: storage size of ‘arr’
isn’t known
6 |     char arr[];
  |          ^~~
```

This actually means that in order to **refer to an array’s element by its index** you must **first determine the array’s size!**

Pointers And Arrays

What will the following print?

```c
// source/arrays_005.c
#include <stdio.h>

int main() {
    int arr[] = {2, 6, 5, 1};
    int* ptr = arr;
    ptr++;
    printf("%d\n", *ptr);
}
```

Pointers And Arrays

Do you observe something strange in the following?

```c
    int arr[] = {2, 6, 5, 1};
    int* ptr = arr;
    ptr++;
    printf("%d\n", *ptr);
}
```

Heads up!

Pointers And Arrays

Do you observe something strange in the following?

```
1     int arr[] = {2, 6, 5, 1};
2     int* ptr = arr;
3     ptr++;
4     printf("%d\n", *ptr);
5 }
```

- **Line 3:** We declare an integer pointer and store the **array** there, **not a reference!**

Heads up!

Pointers And Arrays

Do you observe something strange in the following?

```
1       int arr[] = {2, 6, 5, 1};
2       int* ptr = arr;
3       ptr++;
4       printf("%d\n", *ptr);
5   }
```

- **Line 3:** We declare an integer pointer and store the **array** there, **not a reference!**
- Why does it work?

Pointers And Arrays

- In C, arrays of type `<T>` are actually pointers to items of type `<T>`.
- This means that, when declaring an array, we are actually declaring a pointer to the first memory location occupied by its first element.
- So, arrays are actually of **pointer type!**
- This means that using `&` to get their memory address is of no use, since they already represent a memory address.

Pointer Tricks

What will the following print?

```c
// source/arrays_006.c
#include <stdio.h>

int main() {
    int arr[] = {2, 6, 5, 1};
    float* ptr = (float*) arr;
    ptr++;
    printf("%f\n", *ptr);
}
```

Pointer Tricks

What will the following print?

```c
// source/arrays_007.c
#include <stdio.h>

int main() {
    int arr[] = {2, 6, 0, 0, 4};
    double* ptr = (double*) arr;
    ptr++;
    printf("%f\n", *ptr);
}
```

Pointer Tricks

What will the following print?

```c
// source/arrays_008.c
#include <stdio.h>

int main() {
    int arr[] = {2, 6, 0, 1, 4};
    double* ptr = (double*) arr;
    ptr++;
    printf("%f\n", *ptr);
}
```

Looping Over An Array

```c
// source/arrays_009.c
#include <stdio.h>

int main() {
    int arr[] = {2, 6, 0, 1, 4};
    for (int i = 0; i < 5; i++) {
        printf("arr[%d] == %d\n", i, arr[i]);
    }
}
```

Looping Over An Array With Pointers

Can you loop over the same array without using the `arr[i]` syntax?

Looping Over An Array With Pointers

Can you loop over the same array without using the `arr[i]` syntax?

```c
// source/arrays_010.c
#include <stdio.h>

int main() {
    int arr[] = {2, 6, 0, 1, 4};
    for (int i = 0; i < 5; i++) {
        printf("arr[%d] == %d\n", i, *(arr + i));
    }
}
```

Pointer Arithmetic (Again)

- In general, the expression `pointer + integer` is interpreted as: increment the `pointer` by the size of its pointing type times the `integer`.
- So, for an `int* ptr`, `ptr + 6` should be interpreted as “move the pointer `ptr` by `6 * sizeof(int)`, i.e., `6 * 4` bytes”.
- So, for a `double* ptr`, `ptr + 5` should be interpreted as “move the pointer `ptr` by `5 * sizeof(double)`, i.e., `5 * 8` bytes”.

Passing Arrays To Functions

What will this print?

```c
// source/arrays_011.c
#include <stdio.h>

void printArray(int arr[], int length) {
    for (int i = 0; i < length; i++) {
        printf("arr[%d] == %d\n", i, *(arr + i));
    }
}

int main() {
    int arr[] = {2, 6, 0, 1, 4};
    printArray(arr, 5);
}
```

Passing Arrays To Functions

What will this print?

```c
// source/arrays_012.c
#include <stdio.h>

void printArray(int* arr, int length) {
    for (int i = 0; i < length; i++) {
        printf("arr[%d] == %d\n", i, *(arr + i));
    }
}

int main() {
    int arr[] = {2, 6, 0, 1, 4};
    printArray(arr, 5);
}
```

Passing Arrays To Functions

What will this print?

```c
// source/arrays_013.c
#include <stdio.h>

void foo(int* arr, int length) {
    for (int i = 0; i < length; i++) {
        if (*(arr + i) == 0) {
            *(arr + i) = 4;
        }
    }
}

int main() {
    int arr[] = {2, 6, 0, 1, 4};
    printf("%d\n", arr[2]);
    foo(arr, 5);
    printf("%d\n", arr[2]);
}
```

Array Decay

- A common C catch–phrase is that “arrays decay into pointers”.
- This simply means that, whenever required, arrays are interpreted as pointers, as we have already discussed above.
- As a consequence, when passing an array to a function, we are actually passing a pointer.
- This means that an array is always **passed by reference**. So, in case we need to pass an array be value, we have to devise various tricks we shall see in upcoming lectures.

Fun Time!

Advanced Pointer Fun

While we have said enough about pointers, we have not explored pointer-land in full. The following tutorial will help you do so:

`https://learnmoderncpp.com/arrays-pointers-and-loops/`

Follow the tutorial step-by-step and pay attention to the "Experiments" it asks you to execute. Write down your observations in a document, which you will share with me at the end of the class at: `v.markos@mc-class.gr`.

Homework

Complete all exercises and problems in MIT's C++ course second assignment, found here:

`https://ocw.mit.edu/courses/6-096-introduction-to-c-january-iap-2011/797ebff419fa2cc3a10af2c5f19be961_MIT6_096IAP11_assn02.pdf`

For your convenience, you can also find the assignment file in this lecture's materials, at: `../homework/MIT6-096IAP11-assn02.pdf`. Submit all your work in the online form below as a single `.zip` file:

`https://forms.gle/rSq3VSpcouRAVjqMA`

or via email at: `v.markos@mc-class.gr`.

Any Questions?

Do not forget to fill in the questionnaire shown right!

`https://forms.gle/dKSrmE1VRVWqxBGZA`