C-Lessons

Arrays

Lecturers: Mirko Jantschke, Pascal Scholz

4. Dezember 2018

Contents

Arrays

Let’s talk about memory

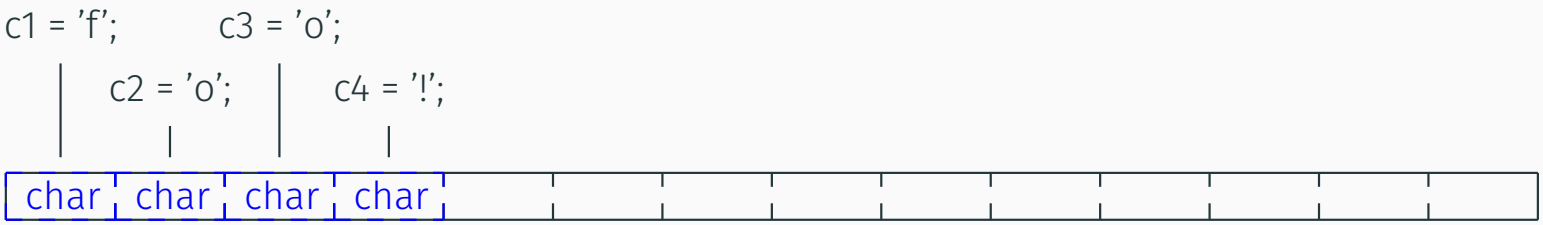
- Consider a program using 4 characters.

Let’s talk about memory

- Consider a program using 4 characters.

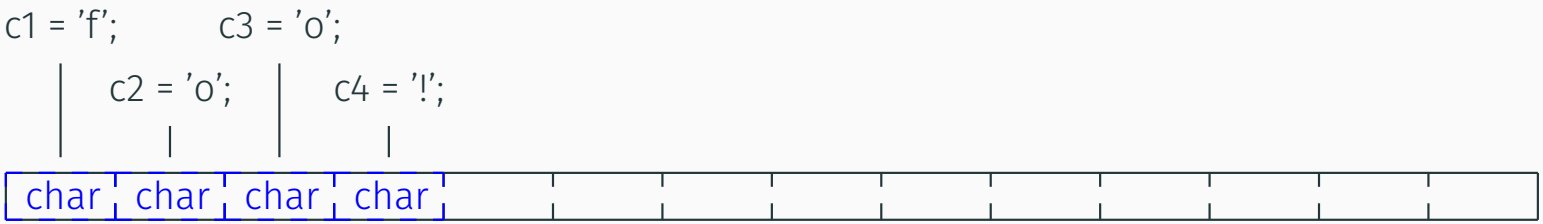
Let’s talk about memory

- Consider a program using 4 characters.
- (Maybe they are meant to say ”foo!”)



Let’s talk about memory

- Consider a program using 4 characters.
- (Maybe they are meant to say ”foo!”)
- Would it not be nice to iterate through these chars?



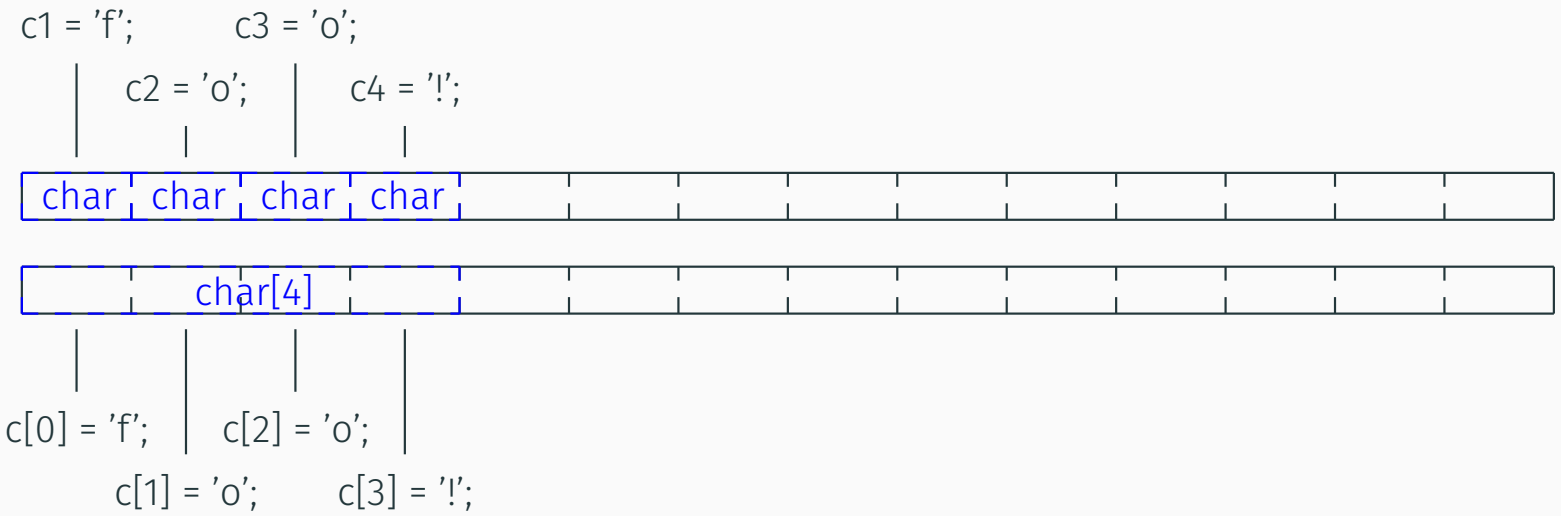
Let’s talk about memory

- Consider a program using 4 characters.
- (Maybe they are meant to say ”foo!”)
- Would it not be nice to iterate through these chars?

C offers an opportunity to access variables through an index: arrays

Let’s talk about memory

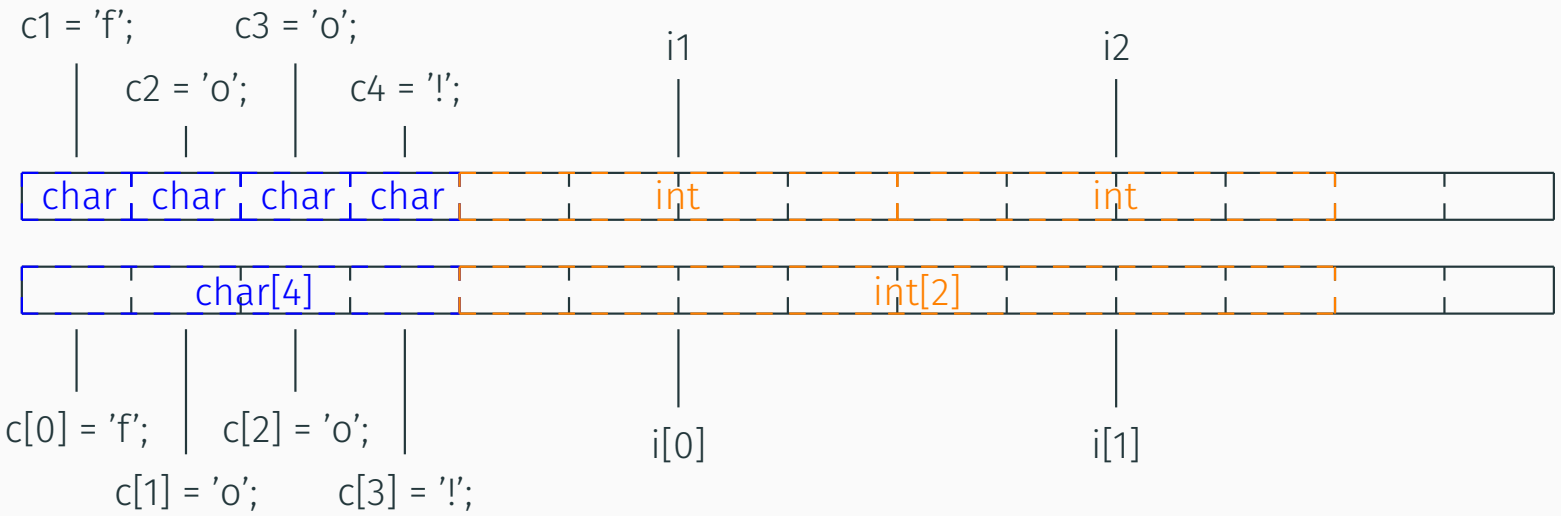
- Consider a program using 4 characters.
- (Maybe they are meant to say ”foo!”)
- Would it not be nice to iterate through these chars?



C offers an opportunity to access variables through an index: arrays

Let’s talk about memory

- Consider a program using 4 characters.
- (Maybe they are meant to say ”foo!”)
- Would it not be nice to iterate through these chars?



C offers an opportunity to access variables through an index: arrays

Definition

To only declare an array, you have to specify the size.

```
type identifier[size];
```

You can leave the size out, by defining all elements. The compiler will count.

```
int leet[] = {1, 3, 3, 7};
```

You can define a single Element through its index.

```
int leet[4];
leet[0] = 1;
leet[1] = 3;
leet[2] = 3;
leet[3] = 7;
```

Note: The first index is 0, the last one is *size* - 1

Partial initialization

If you declare an array, memory gets reserved. This memory is full of trash data. Before you initialize the array, you can not determine what is in.

You can initialize an array partially. If you do so, the rest gets initialized with 0.

```
int arr[5] = {1, 3};    /* initializes {1, 3, 0, 0, 0} */
```

To clear an array during its initialization use the following syntax:

```
char clear[255] = {0};
```

Access

You access an array element, like defining it, through its index.

```
int leet[] = {1, 3, 3, 7};
int sum = leet[0] + leet[1] + leet[2] + leet[3];
```

You don't have to use a discrete number. It can be a statement to.

```
int a = 0, b = 1, c = 2, d = 3;
int leet[] = {1, 3, 3, 7};
int sum = leet[a] + leet[b] + leet[c] + leet[d];
```

But:

```
int leet[] = {1, 3, 3, 7};
int whatIf = leet[4];   /* feel free to try out */
```

Looping through arrays

The fact, that you can use a statement as index allows us to easily loop through an array.

```
int leet[] = {1, 3, 3, 7};
for(int i = 0; i < 4; ++i)
    printf("%d\n", leet[i]);
```

Looping through arrays

The fact, that you can use a statement as index allows us to easily loop through an array.

```
int leet[] = {1, 3, 3, 7};
for(int i = 0; i < 4; ++i)
    printf("%d\n", leet[i]);
```

There also is a way to determine the length of an array. But it's a little tricky and requires previous knowledge.

sizeof

Remember, Remember

Old and busted:

- The size of types may differ on different architectures
- The size of a char is **always** 1 byte
- Keep in mind that 1 byte $\neq$ 8 bits on some machines

New hotness:

- You can get the size (in bytes) of a variable or type with the operator *sizeof*

```
char a = 'a';
int sizeOfA = sizeof a;       /* defined as 1 */
int sizeOfInt = sizeof(int); /* depending on architecture */
```

Note: Parentheses are required when passing types and recommended when passing identifiers.

Sizeof an array

What if?

```
int leet[] = {1, 3, 3, 7};
printf("%d\n",sizeof leet);
```

Sizeof an array

What if?

```
int leet[] = {1, 3, 3, 7};
printf("%d\n",sizeof leet);
```

The output is the actual size of the array, not the number of its elements.

You have to divide that size by the size of a single element.

```
int leet[] = {1, 3, 3, 7};
int nLeet = sizeof leet / sizeof (int);   /* too specific */
int nLeet = sizeof leet / sizeof leet[0]; /* better */
```

Sizeof an array

What if?

```
int leet[] = {1, 3, 3, 7};
printf("%d\n",sizeof leet);
```

The output is the actual size of the array, not the number of its elements.

You have to divide that size by the size of a single element.

```
int leet[] = {1, 3, 3, 7};
int nLeet = sizeof leet / sizeof (int);   /* too specific */
int nLeet = sizeof leet / sizeof leet[0]; /* better */
```

Now we can build for loops for iterating through every kind of array:

```
short range[24];
for(int i = 0; i < (sizeof range / sizeof range[0]); ++i) {
    range[i] = i;
}
```

Strings n’ more

Oh, hello again

In C, strings are handled as zero terminated Character arrays.
In memory, the string ”Hello World!”would look as follows:

- Note that ’\0’ is different from the the character ’0’

To represent it in an array, you could do the following:

```
char hello[] = {'H', 'e', 'l', 'l', 'o', ' ',
                'W', 'o', 'r', 'l', 'd', '!', '\0'};
```

But there is a short form for that:

```
char hello[] = "Hello World!";
```

Printing strings

You can print strings with printf with the placeholder *%s*.
It expects an array of chars as an argument:

```
char hello[] = "Hello World!";
printf("%s\n", hello);
```

If you don't need any formatting, you can also use *puts*, which has a simpler syntax:

```
char hello[] = "Hello World!";
puts(hello);
```

String input

You **could** use *scanf()* for string input, but it is kind of a mess. You better use *fgets*:

```
char input[42];
fgets(input, sizeof input, stdin);
```

- *fgets()* reads up to size-1 characters (here: 42-1), and writes the '\0' at the end of the array input.
- It stops reading after a newline.
- *stdin* is the standard input

Passing arrays

An array is passed by its identifier, no big deal. The real question is, how a function is defined, that takes an array as argument.

```
int foo(char arr[]) {
    printf("%d\n", arr[0]);
}
```

- **But:** If you want to get the size of the array in this function, you'll get a false value. The reason for this is discussed later.
- Because the size is unknown, you don't know whether an element is safe to access or already "out of bounds"
- So you may want to pass the size as a second argument

Arrays of arrays

- Every sub-array is of the same type
- All dimensions apart from the first must be specified
- Declaration:

```
char arr[3][4];
/* an array containing 3 arrays containing 4 chars */
```

- Assignment:

```
arr[0][0] = 'f'; /* first element of arr[0] is 'f' */
arr[1] = "bar";  /* first element of arr is "bar" */
arr[2] = {'b', 'o', 'z', '\0'};
```

- Definition:

```
char arr[][4] = {{'f', 'o', 'o', '\0'},
                 {'b', 'a', 'r', '\0'}};
```

Passing arrays of arrays

- You can also pass arrays of arrays to functions
- Once again, we need to specify the dimensions
- Either use fixed values or include dimension arguments in the declaration of the array like shown below:

```
int foo(char arr[][42]) {
    /* code */
}
int bar(int dim2, char arr[][dim2]) {
    /* code */
}
```

- The dimension arguments must be passed before the array itself

Related task

Array printing function

Task as online:

Write a function that prints an Integer array.

Experts: Write a function that prints an integer matrix.

Experts: Simulate “real” randomness that cannot be recreated.