C introduction

Arrays

Contents

Hello World explosion

Remember the program that printed "Hello World!" by passing 13 chars to *printf*?

- Write a program that prints the following text by passing chars to *printf*.

```
Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
sed diam nonumy eirmod tempor invidunt ut labore et
dolore magna aliquyam erat, sed diam voluptua. At vero
eos et accusam et justo duo dolores et ea rebum.
```

Hello World explosion

Remember the program that printed "Hello World!" by passing 13 chars to *printf*?

- Write a program that prints the following text by passing chars to *printf*.

```
Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
sed diam nonumy eirmod tempor invidunt ut labore et
dolore magna aliquyam erat, sed diam voluptua. At vero
eos et accusam et justo duo dolores et ea rebum.
```

Just kidding

Hello World explosion

Remember the program that printed "Hello World!" by passing 13 chars to *printf*?

- Write a program that prints the following text by passing chars to *printf*.

```
Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
sed diam nonumy eirmod tempor invidunt ut labore et
dolore magna aliquyam erat, sed diam voluptua. At vero
eos et accusam et justo duo dolores et ea rebum.
```

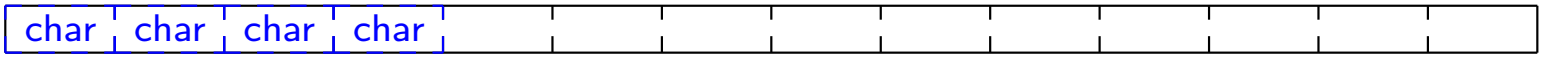
Just kidding
If you thought about iterating through the 200+ variables you defined to solve the problem, let me tell you: There is no way.

Let's talk about memory

- Consider a program using 4 characters.

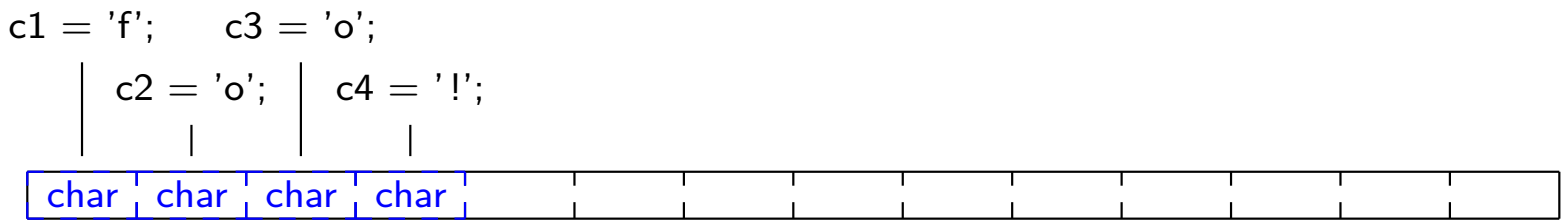
Let's talk about memory

- Consider a program using 4 characters.



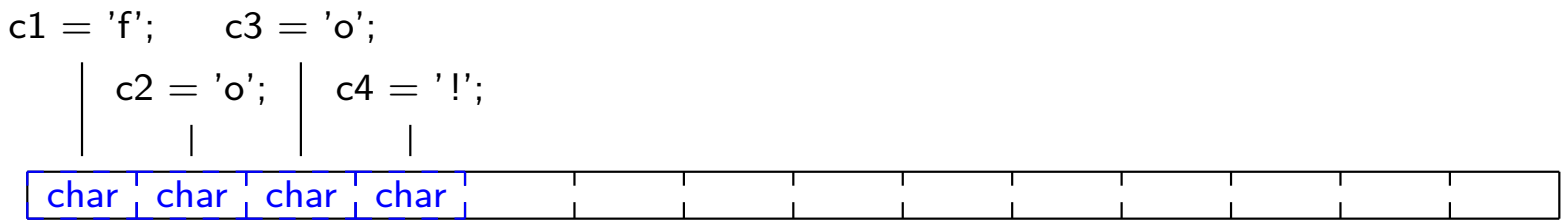
Let's talk about memory

- 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?



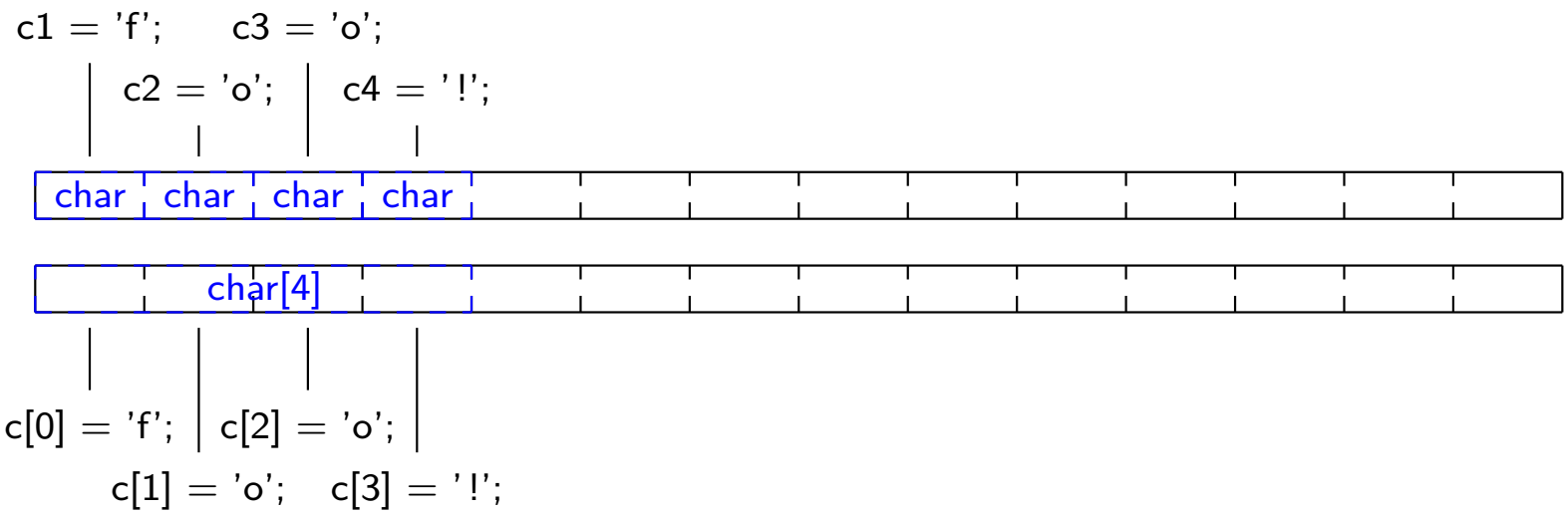
Let's talk about memory

- 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

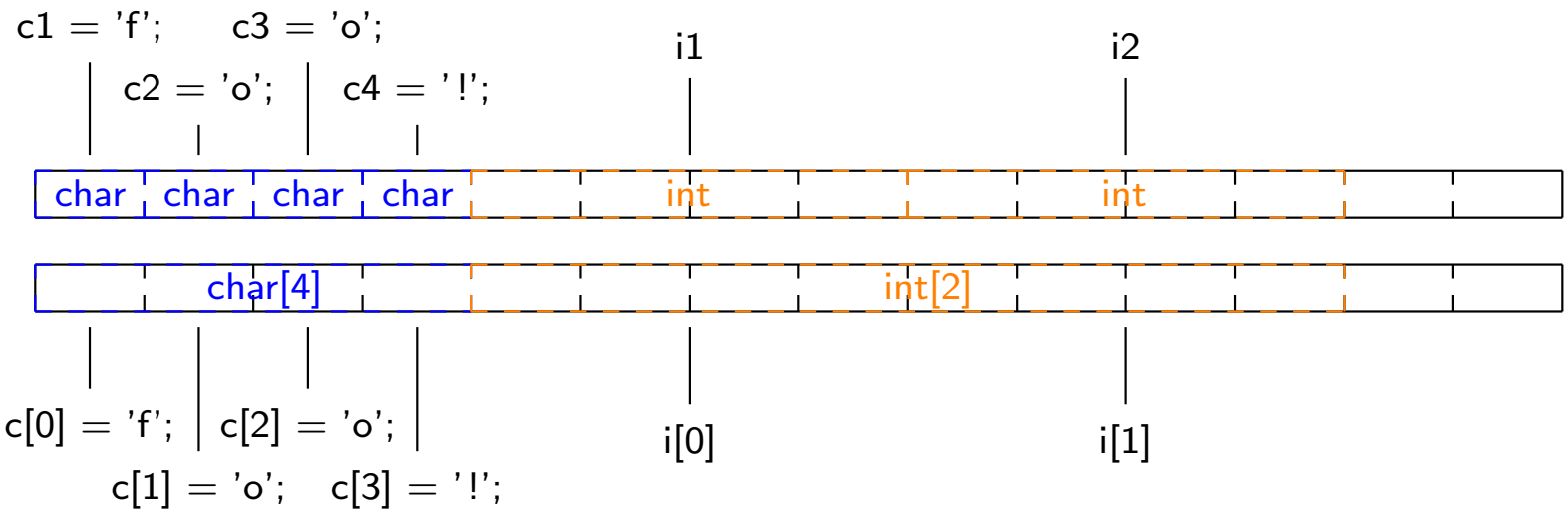
- Consider a program using 4 characters.
- (Maybe they are meant to say "foo!")
- Would it not be nice to iterate through these chars?



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.

Remember, Remember

Old and busted :

- The size of variables may differ on different architectures
- The size of a char is **always** 1 byte

New hotness:

- You can get the size of a variable with the operator *sizeof*
- You can also get the size of a type with sizeof

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

Note: You can **and should** leave the parentheses when passing an identifier but you have to use them in case you pass a type.

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 total numbers of bytes, the array reserves. Not the number of Elements it consists of. In case of a character array, these numbers are equal, but not for other array types.

To get the number of Elements you have to divide the size of the array by the size of the array type (the size of one single element).

```
int leet[] = {1, 3, 3, 7};
int nrOfElements = sizeof leet / sizeof (int);
```

A little bit of magic

Usually, you should know the element type of the array. But if, for some good reason, you do not know what the element type of an array is, you can use this little trick:

```
1 int nrOfElements = sizeof leet / sizeof leet[0];
```

Iterate through every array

With this previous knowledge we now can build a for loop for iterating through every kind of array:

```
long array[24];
...                                     /* fill array */
for(int i = 0; i < (sizeof array / sizeof (long)); i++){
    printf("%l\n", array[i]);    /* print array */
}
```

Exercises

- You are now able to solve tasks 15 and 16.

Oh, hello again

In C, strings are handled as Zero terminated Character Arrays.
In memory, the string "Hello World!" would look as follows:

'H'	'e'	'l'	'l'	'o'	' '	'W'	'o'	'r'	'l'	'd'	'!'	'\0'	

- Note that '\0' is different from the number '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

There is a placeholder for strings in printf: *%s*
It expects just the name of the array:

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

- Now, take your Lorem ipsum program and reduce it to 2 lines.

Printing strings

There is a placeholder for strings in printf: *%s*
It expects just the name of the array:

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

- Now, take your Lorem ipsum program and reduce it to 2 lines.
  - Hint: For a fancier loremipsum look at `http://slipsum.com`

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 and of the array input.
- It stops reading after a newline.
- *stdin* is the standard input

Array of arrays

- Every sub-array is of same type
- Declaration:

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

- Assignment:

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

- Definition:

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

Passing arrays

An array is passed by its name, 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.
  → You may want to pass the size as a 2nd value

Exercises

- You are now able to solve tasks 17, 18 and 19.