CEng 140

Strings and Pointers

Strings (and Pointers)

- [As we know] C uses NULL terminated arrays of chars to represent strings
- To create a string variable you must **allocate** sufficient space for the number of characters and the NULL character '\0'.
  - Using arrays
  - Using pointers

Using arrays for strings

```
char robot[5];   // declaration
```

- **Assignment** of a string to an array: two ways
- First way: each array element assigned to a char

```
robot[0] = ‘g’;
robot[1] = ‘o’;
robot[2] = ‘o’;
robot[3] = ‘d’;
robot[4] = ‘\0’;
```

robot[0]	g
robot[1]	o
robot[2]	o
robot[3]	d
robot[4]	\0

Using arrays for strings

char robot[5];   // declaration

- **Assignment** of a string to an array: two ways
- second way: via strcpy func

//strcpy copies the chars one by one from

// source str to destination str

strcpy(robot, "good");

String constant

robot[0]	g
robot[1]	o
robot[2]	o
robot[3]	d
robot[4]	\0

Using arrays for strings

- You can also store a string in an array during the **initialization**

```
char robot[5];   // declaration
char robot[5] = {‘g’, ‘o’, ‘o’, ’d’, ‘\0’};   // or
char robot[5] = "good";
```

robot[0]	g
robot[1]	o
robot[2]	o
robot[3]	d
robot[4]	\0

When a char array is intialized to a string constant:

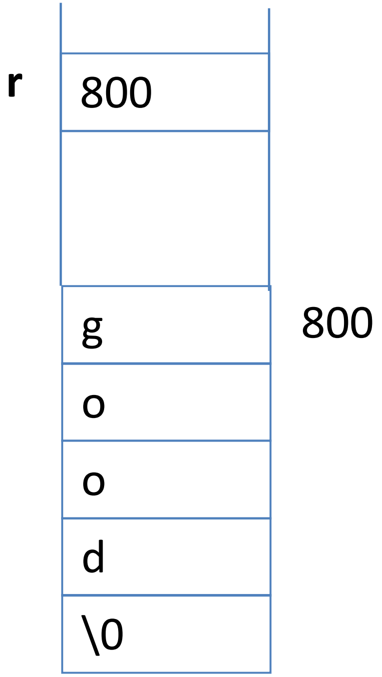
- Same name (robot) always refers to the same storage

- Individual chars can be modified by assignments!

```
robot[0] = ‘w’; // works
```

Using pointers for arrays

```
char *r;  // declaration
              // normally, alloc space for string before assignment
r = (char *) malloc(sizeof(char) * 5);
// Assignment: first way
r[0] = ‘g’;
r[1] = ‘o’;
…                         r[0] = ‘w’; // works
r[4] = ‘\0’;
// Assignment: second way
strcpy(r, "good");
```



Using pointers for arrays

- You can also store a **string constant** in a ptr via **(initialization, or) direct assignment**

```
char *r;  // declaration

char *r = "good";  //or

char *r;

r = "good";
```

Hey! You did not allocate any storage for the string, how is this possible?

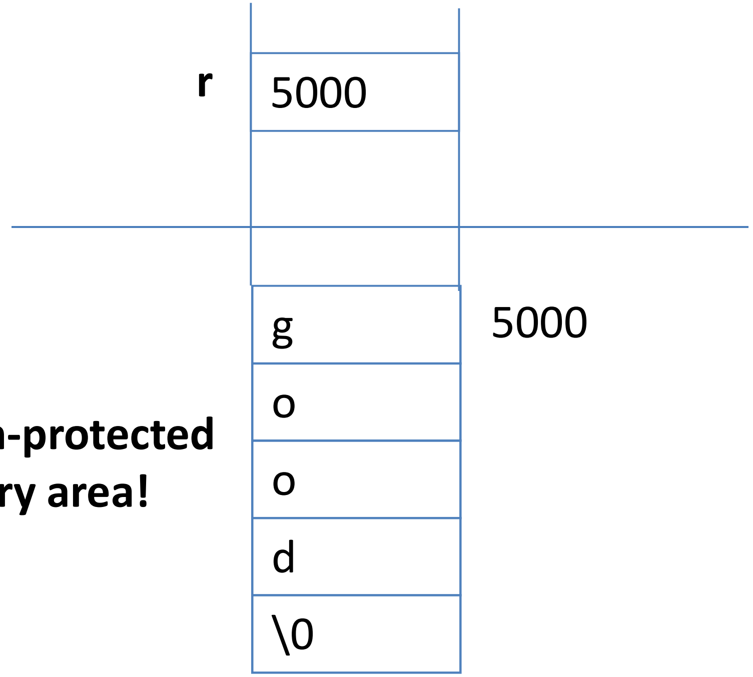
[More about the] String Constants

- A string constant is a sequence of chars in " " and compiler automatically adds NULL character at the end.
- When a string constant appears anywhere (**except** as an initializer of a char array or an argument to the sizeof operator) the chars making up the string (together with NULL) are stored in contiguous memory locations, and **string contant** becomes a **pointer** to the first char of the stored string.
  - Usually stored in a **system-protected memory area!**

Mystery solved!

```
char *r = "good";  //or
char *r;
r = "good";
```

String constant;
a pointer to where "good" is stored!
Both sides are of type **pointer to char**!



system-protected memory area!

Mystery solved!

```
char *r = "good";  //or
char *r;
r = "good";
```

String constant;
a pointer to where "good" is stored!
Both sides are of type **pointer to char**!

When a char pointer is intialized/assigned to a string constant:

- Pointer var may be assigned to point somewhere else

- But can **NOT** modify the string pointed by it!

```
  r [0] = ‘w’; // fails! Result is undefined!
```

Let’s recall again cases with a string constant:

- If your variable has its own memory and you copy string constant there, you can modify it as you wish, as in:
  - char robot[5];
    strcpy(robot, "good");
  - char robot[5] = "good";
  - char *r;
    r = (char *) malloc(sizeof(char) * 5);
    strcpy(r, "good");

Otherwise…!

```
char *r;  //
r = (char *) malloc(sizeof(char) * 5);
r = "good";
r [0] = ‘w’; // What will happen?
```

Result is undefined! Bec you are not using the allocated memory but pointing to a string constant, which is not modifiable!

Otherwise…!

char robot[5];

robot = "good"; // What will happen?

RECALL this is not array initialization (where string constant behaves exceptionally), so you are simply trying to change where an array name points to!

→ compile-time error!

strcpy()

- Now that we know what a string constant really is (i.e., a ptr to char)…
- what should be the prototype of strcpy() function?
  - char robot[5];

    strcpy(robot, "good");
  - char *r;

    r = (char *) malloc(sizeof(char) * 5);

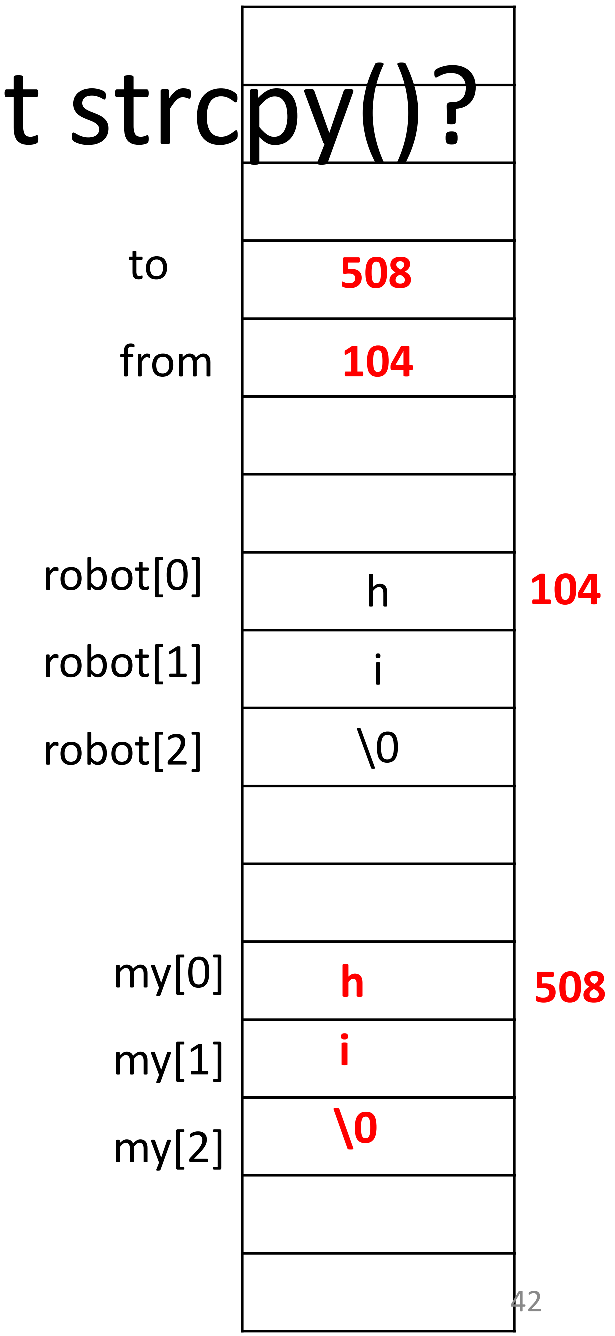
    strcpy(r, "good");

    or, strcpy(r, robot);

How can we implement strcpy()?

```
void strcpy(char *to, char *from)
{  while (*to = *from)
        to++ , from++ ; }

int main()
{
char my[3], robot[3]="hi";
strcpy(my, robot); }
```



to	508	
from	104	
robot[0]	h	104
robot[1]	i	
robot[2]	\0	
my[0]	h	508
my[1]	i	
my[2]	\0	

How can we implement strcpy()?

```
void strcpy(char *to, char *from)
{  while (*to = *from)
        to++ , from++ ; }
```

Shorter:

```
void strcpy(char *to, char *from)
{  while (*to++ = *from++) ; }
```

```
char robot[5], my[8];
strcpy(robot, "good"); strcpy(my, robot); …
```

How can we implement strlen()?

```
int my_strlen(char str[])
{ int i;
   for (i=0; str[i] != ‘\0’; i++) ;
   return i;
}
```

C Library Functions

Declared in string.h

size_t → unsigned integral type

size_t **strlen**(const char *s); (length of s w.o. NULL)

char ***strcpy**(char *s1, const char *s2);

(copies s2 to s1 including NULL, returns s1)

Sec. 7.4.1:

…strncpy..

…strcat..

…strcmp…

Array of strings

- True 2D, initialized

char robot[5] = {‘g’, ‘o’, ‘o’, ’d’, ‘\0’}; // or
char robot[5] = "good";

char day[7][5] = {"mon",…, "sun"};

// **OR,** I could first declare array and then assign as:

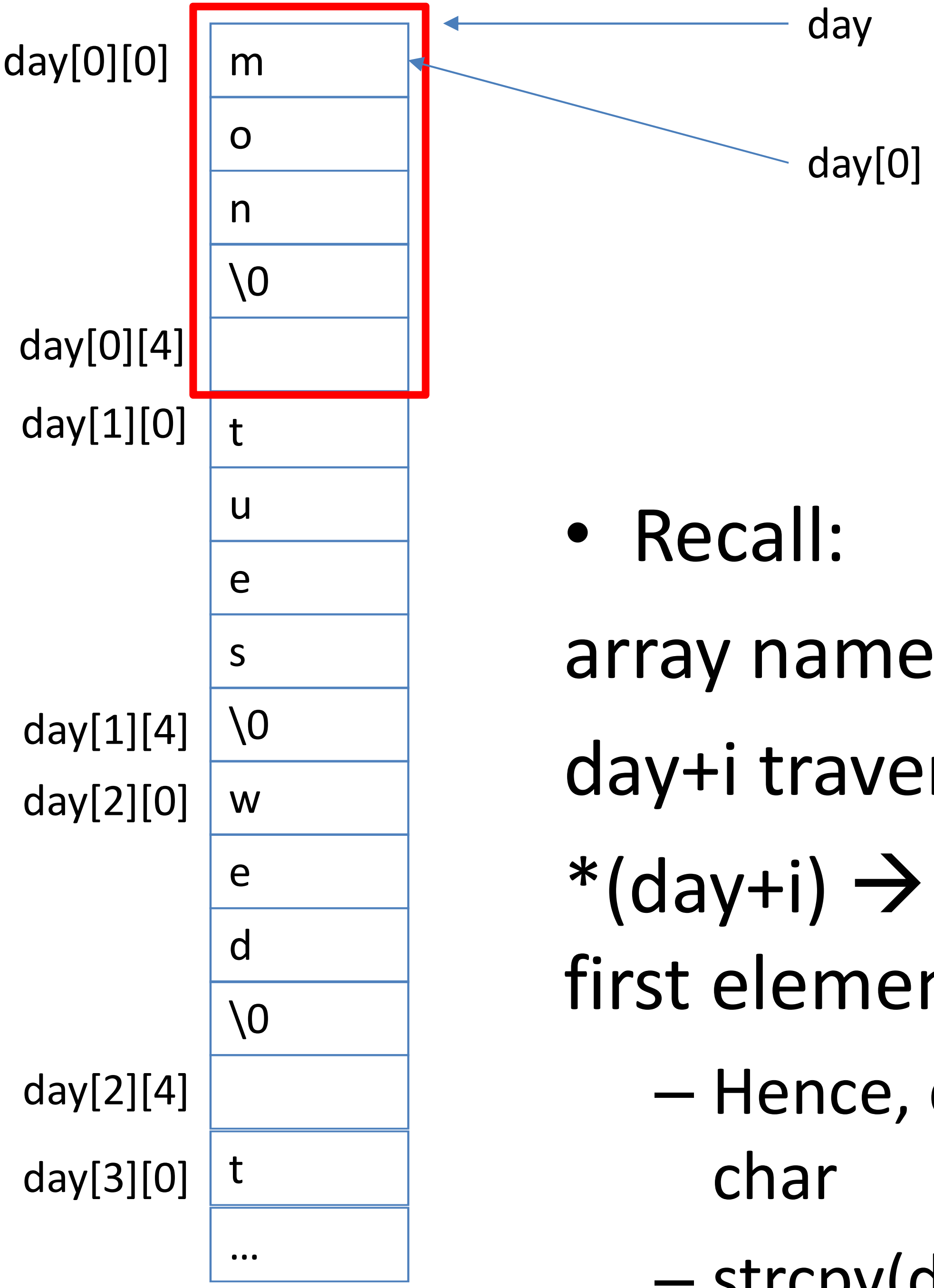
char day[7][5];

day[0][0] = ‘m’; …

// **OR,** assign as:

strcpy(day[0], "mon");

// In all cases, strings in the array are **modifiable**!



- Recall:

array name is a ptr to first array!

day+i traverses arrays

*(day+i) → day[i] is a ptr to the first element in the ith array!

  – Hence, day[i] is of type ptr to char
  – strcpy(day[0], "mon"); is OK!

Pop-up quiz

char day[7][5] = {"mon",…, "sun"}; OK

char day[7][5];

day[0]="hey";

day[0][1]= ‘m’; What will happen?

a) Compile-time error

b) Run-time error: string is not modifiable

c) Undefined: string is not modifiable

d) String becomes mey

We have seen

char robot[5] = "good"; OK

char robot[5] ;

robot= "good"; COMPILE ERROR

Array of strings

- True 2D, passing as a parameter:

Rewritten as: **char (*d)[5]**

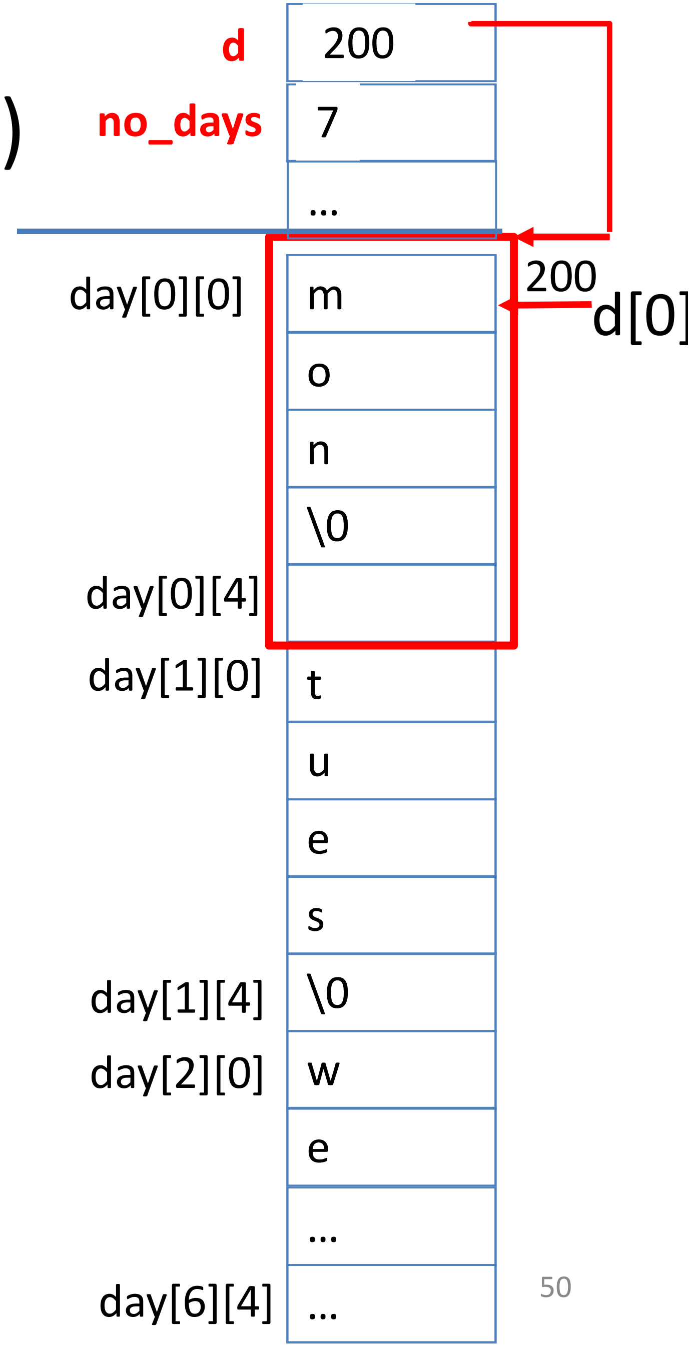
```
void list_days (char d[][5], no_days)
{ int i;
  for (i=0; i<no_days; i++)
     printf("%s\n", d[i]); }
```

No need for second dim length, as each array is ended with NULL!

Rewritten as: **char (*d)[5]**

```
void list_days (char d[][5], no_days)
{ int i;
   for (i=0; i<no_days; i++)
      printf("%s\n", d[i]); }

int main(void)
{char day[7][5] = {"mon",..., "sun"};
  list_days(day,7);}
```



d	200
no_days	7
	…
day[0][0]	m
	o
	n
	\0
day[0][4]	
day[1][0]	t
	u
	e
	s
day[1][4]	\0
day[2][0]	w
	e
	…
day[6][4]	…

Array of strings

- Dynamic 2D, iliffe vector, can be:

```
char *day[7] = {"mon",…, "sun"};
```

Not-modifiable

Or:

In what cases,
strings are **modifiable**?

```
char *day[7];
day[0] = "mon";
```

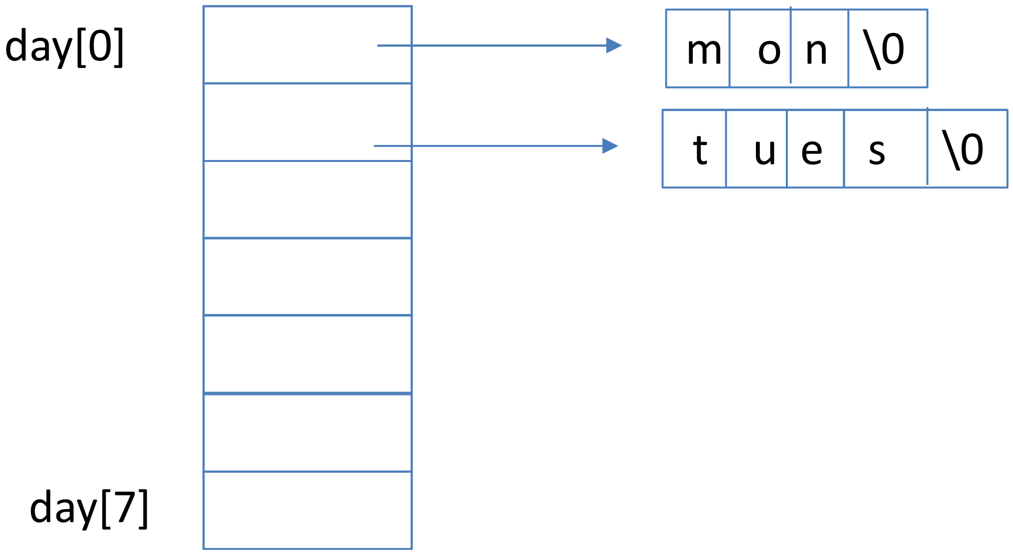
Not-modifiable

Or:

```
char *day[7];
day[0] = (char *) malloc (sizeof("mon"));
strcpy(day[0], "mon"); // or: day[0][0] = ‘m’; …
```

modifiable!

Array of strings



- day[i] is of type ptr to char
- Note that pointed memory space is either explicitly allocated, or system-area (if ptr is assigned to a str constant)

Array of strings

- Dynamic 2D, iliffe vector, passing as a parameter:

Rewritten as: **char **d**

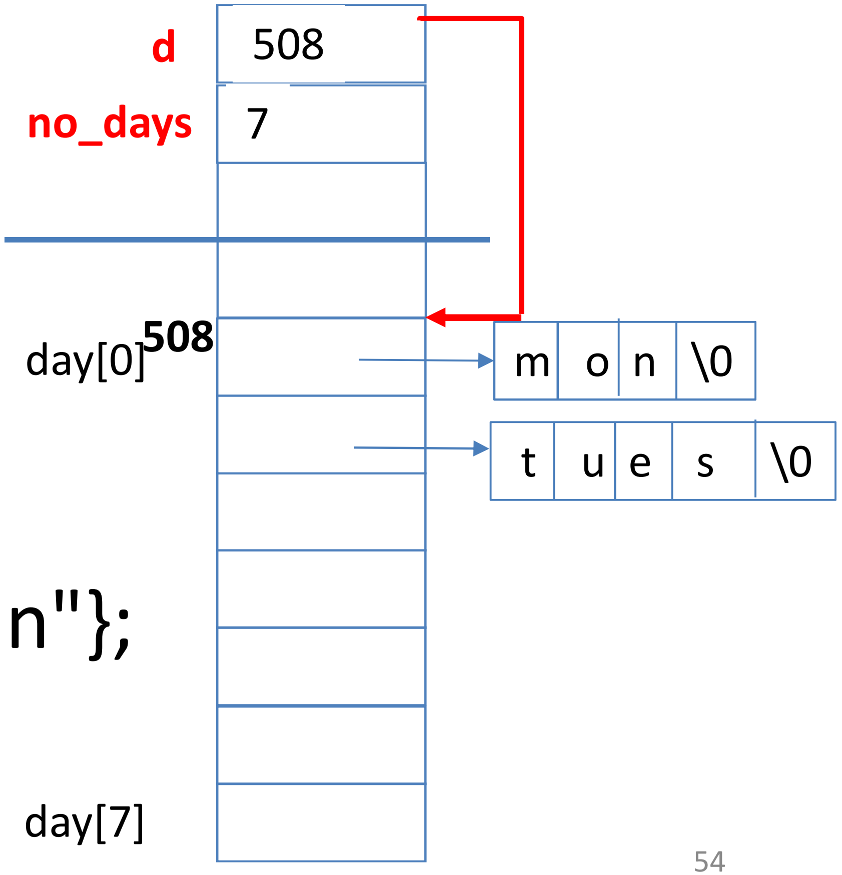
```
void list_days (char *d[], no_days)
{ int i;
   for (i=0; i<no_days; i++)
       printf("%s\n", d[i]); }
```

Array of strings

Rewritten as: **char **d**

```
void list_days (char *d[], no_days)
{ int i;
   for (i=0; i<no_days; i++)
       printf("%s\n", d[i]); }

int main(void)
{char *day[7] = {"mon",..., "sun"};
  list_days(day,7);}
```



d 508

no_days 7

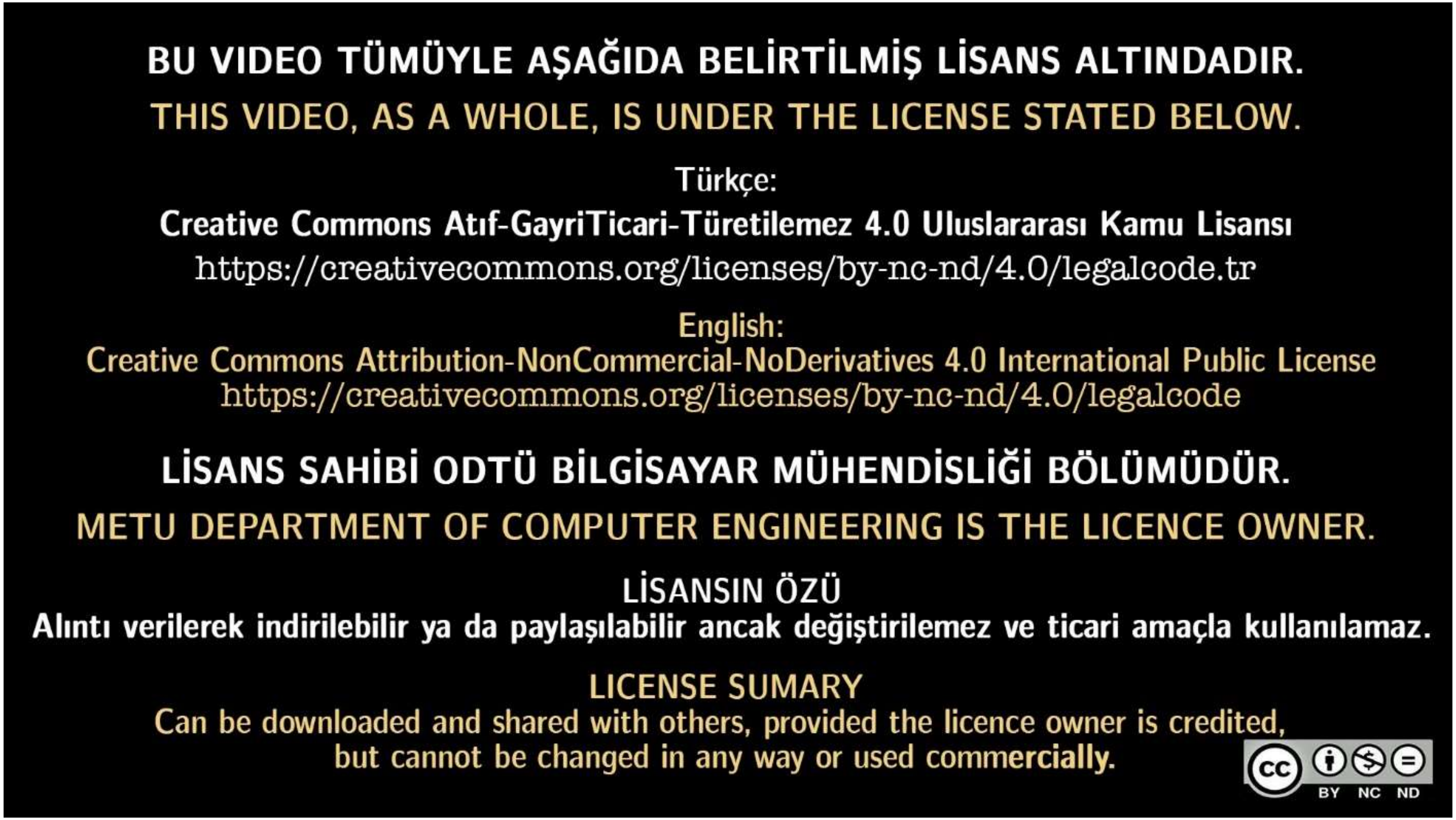
day[0] **508**

m o n \0

t u e s \0

day[7]

Warning



BU VIDEO TÜMÜYLE AŞAĞIDA BELİRTİLMİŞ LİSANS ALTINDADIR.

THIS VIDEO, AS A WHOLE, IS UNDER THE LICENSE STATED BELOW.

Türkçe:
Creative Commons Atıf-GayriTicari-Türetilemez 4.0 Uluslararası Kamu Lisansı
https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode.tr

English:
Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International Public License
https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode

LİSANS SAHİBİ ODTÜ BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜDÜR.

METU DEPARTMENT OF COMPUTER ENGINEERING IS THE LICENCE OWNER.

LİSANSIN ÖZÜ
Alıntı verilerek indirilebilir ya da paylaşılabilir ancak değiştirilemez ve ticari amaçla kullanılamaz.

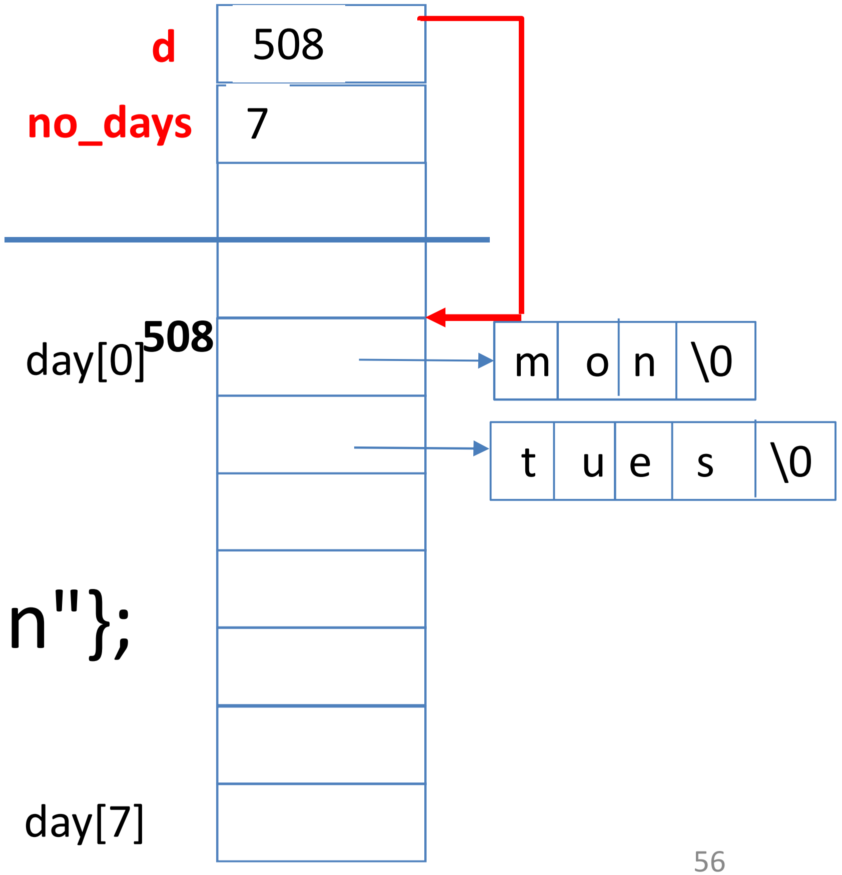
LICENSE SUMARY
Can be downloaded and shared with others, provided the licence owner is credited, but cannot be changed in any way or used commercially.

Recap: Array of strings

Rewritten as: **char **d**

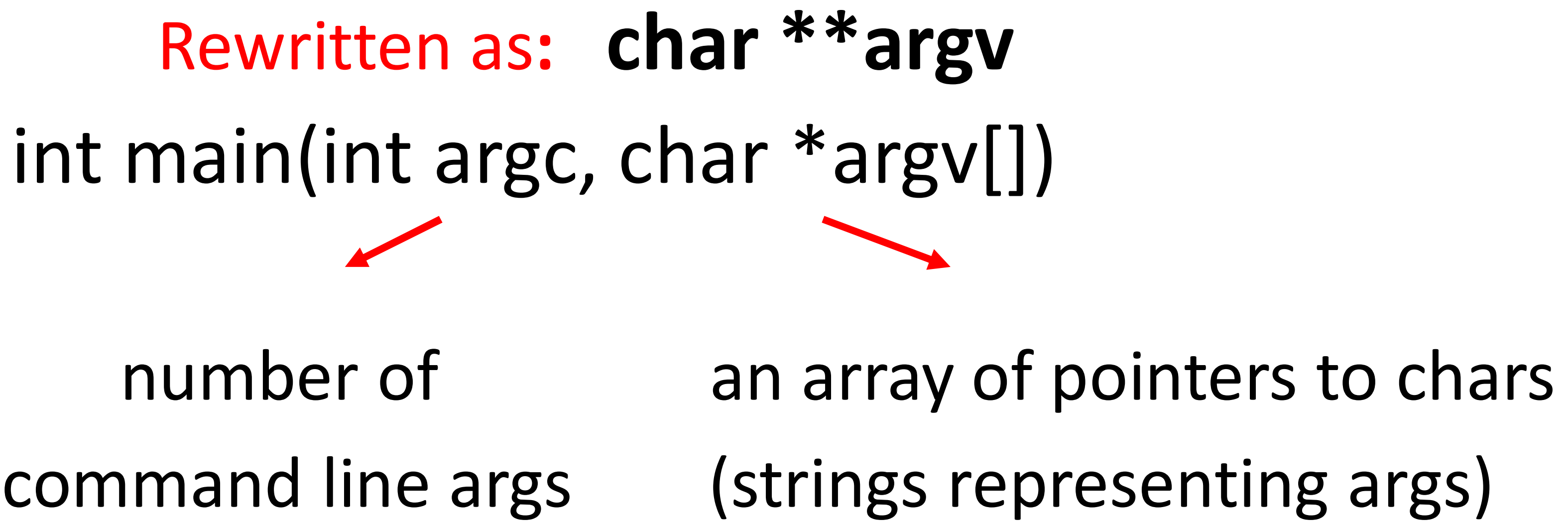
```
void list_days (char *d[], no_days)
{ int i;
   for (i=0; i<no_days; i++)
      printf("%s\n", d[i]); }

int main(void)
{char *day[7] = {"mon",..., "sun"};
  list_days(day,7);}
```



Parameters of main()

- main can be defined with formal parameters so that it can accept command-line arguments
  - main defined as having two parameters, typically called as argc and argv, as follows:



Rewritten as: **char **argv**

int main(int argc, char *argv[])

number of command line args

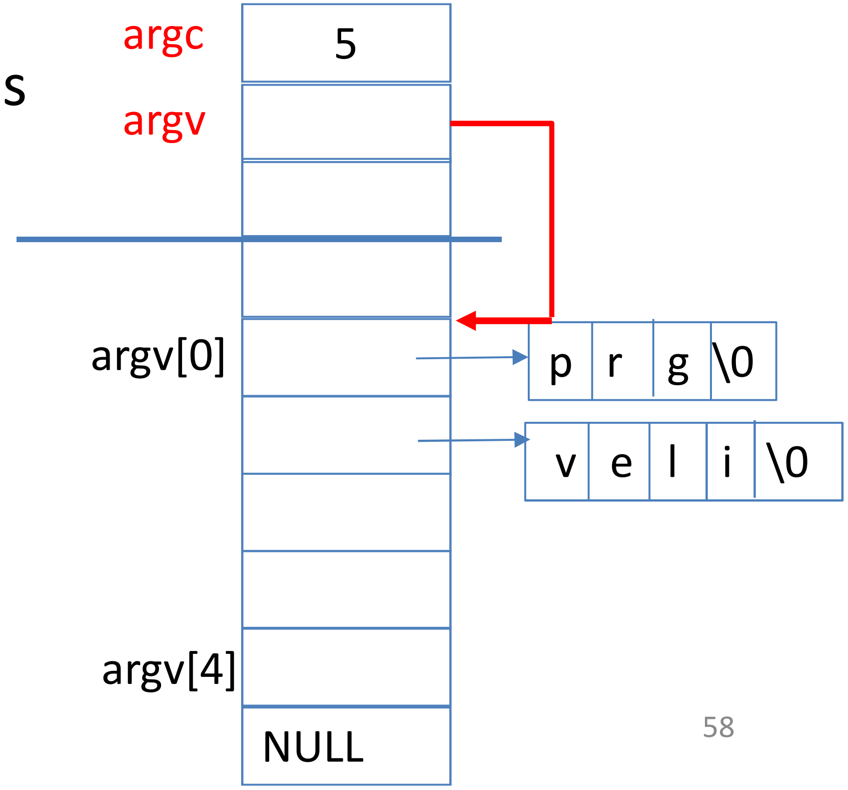
an array of pointers to chars (strings representing args)

Parameters of main()

- Compile your prg.c as executable prg

  ./prg veli ali ayse fatma

- argc: 5, argv is as shown in figure:
  - argv[0] points to the name of the program
  - argv[1] to argv[argc-1] point to args
  - argv[argc] is NULL by convention



Parameters of main()

- So, the command line arguments are strings
  - if needed you can convert them to other types
  - long int atoi(char *) → string to int
  - more functions in Section Appendix A.6 of the textbook