WARM UP

POINTERS

QUIZ

What will this code print?

```
int* elements = {1, 2, 3};

printf("%d", *(elements + 1));
```

A 1

B 2

C 0x200

D Error

ANSWER

What will this code print?

```
int* elements = {1, 2, 3};

printf("%d", *(elements + 1));
```

We cannot initialize a pointer with a list of numbers

A 1

B 2

C 0x200

D Error

QUIZ

What will this code print?

```
int elements[] = {1, 2, 3};
int* ptrElements = elements;

*(ptrElements + 1) = 10;

printf("%d", elements[1]);
```

A 1

B 2

C 10

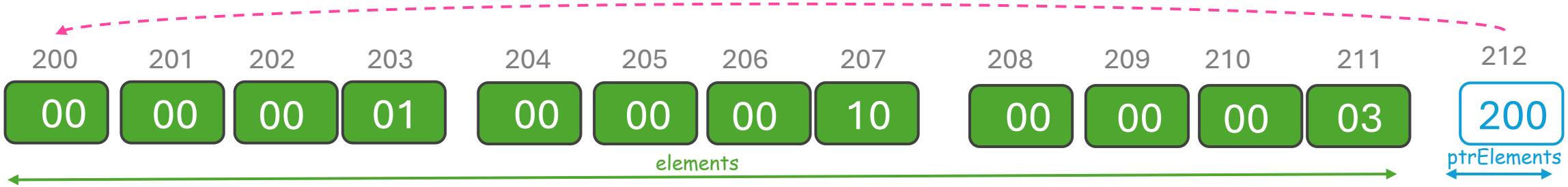
D Error

ANSWER

What will this code print?

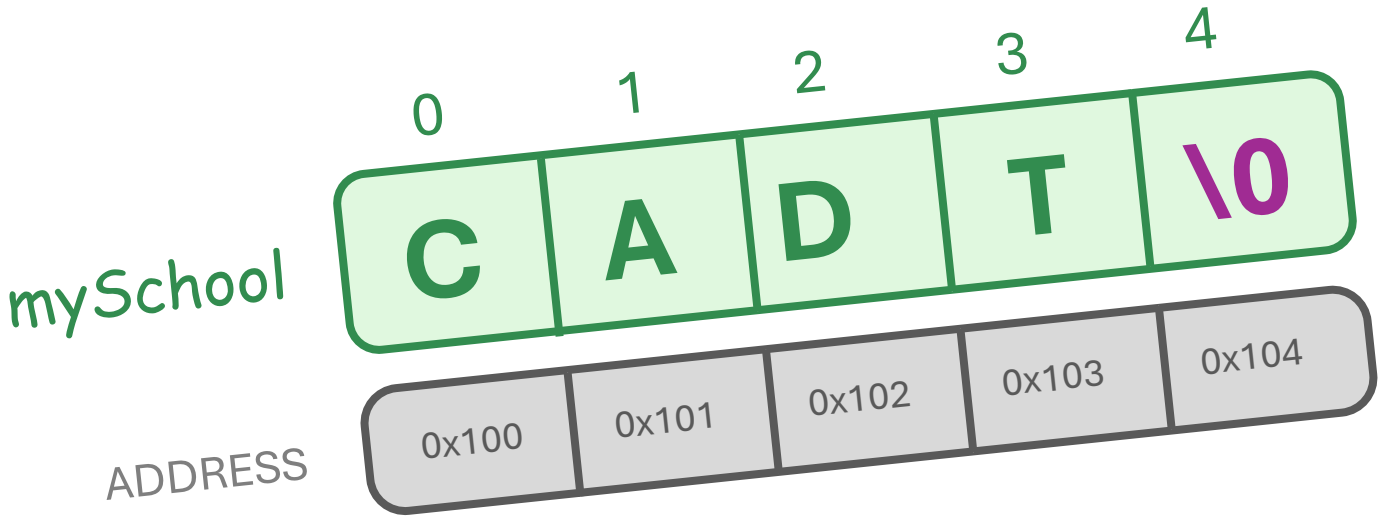
```
int elements[] = {1, 2, 3};
int* ptrElements = elements;

*(ptrElements + 1) = 10;

printf("%d", elements[1]);
```

A 1 B 2 C 10 D Error



ALGORITHM AND COMPUTATIONAL THINKING 2

WEEK 6 – Strings

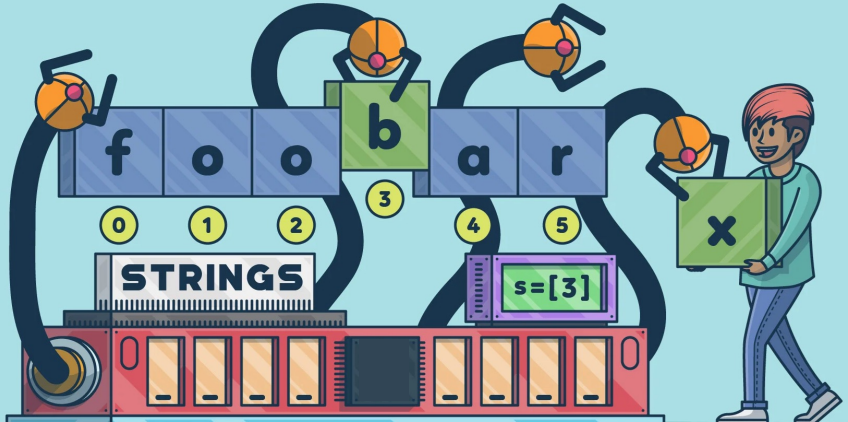


Quick Discussion (Work in pair)

- ❖What is a string in C? Its syntax?
- ❖How do you get one letter from a string?
- ❖Can we modify the character in the string by using a pointer? How?

Session **objectives**

- ✓ Represent **strings** using **arrays** and **null-terminators**.
- ✓ Use **pointers** to access and modify **strings**.
- ✓ Read and print strings using **standard I/O functions**.
- ✓ Handle multiple **strings** using **2D arrays** or arrays of **pointers**.
- ✓ Convert **strings** to numeric values using standard functions.
- ✓ Manipulate **strings** using **built-in C library functions**.



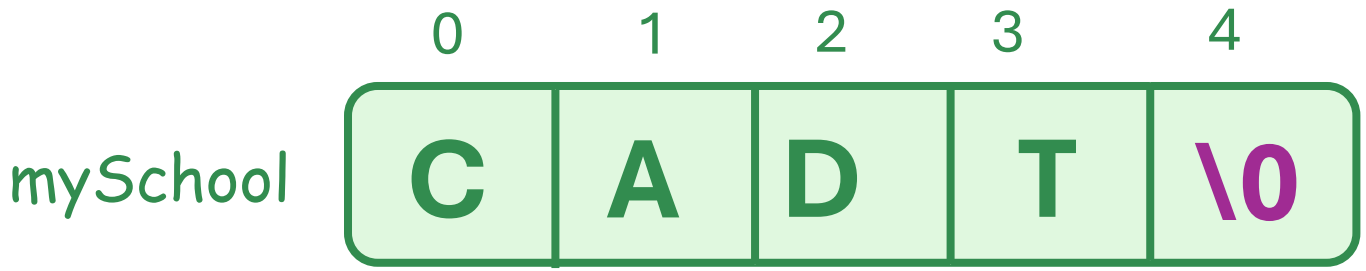
EXPLAIN

Strings **representation**

A string is just an array of characters ending with a NULL character.

```
// Create a string
char mySchool[5] = {'C', 'A', 'D', 'T', '\0'};
```

NULL
Control
character



EXPLAIN

Why a **NULL** character ?

- ✓ The NULL character ('\0') is a special character used to mark **the end of a string**.

Code syntax	ASCII value
`\0`	**0**

It is not the same as the character '0' (which has ASCII value 48)

- ✓ If the NULL character is not found, some C library functions may not produce the desired result.

```
char s[3] = {'B','A','C'};
```

```
char s[4] = {'B','A','C','\0'};
```

EXPLAIN

String **literal**

A string variable can also be created and initialized with a **string literal**.

```
// Initialize a string with an array of char
char mySchool[5] = {'C', 'A', 'D', 'T', '\0'};


// Initialize a string with a string literal
char mySchool[5] = "CADT";
```

surrounded by
double quotes

QUIZ

What will this code print?

```
char myString[] = "Mike";
printf("%d", sizeof(myString));
```

A 4

B 5

C 6

D 7

ANSWER

What will this code print?

```
char myString[] = "Mike";
printf("%d", sizeof(myString));
```

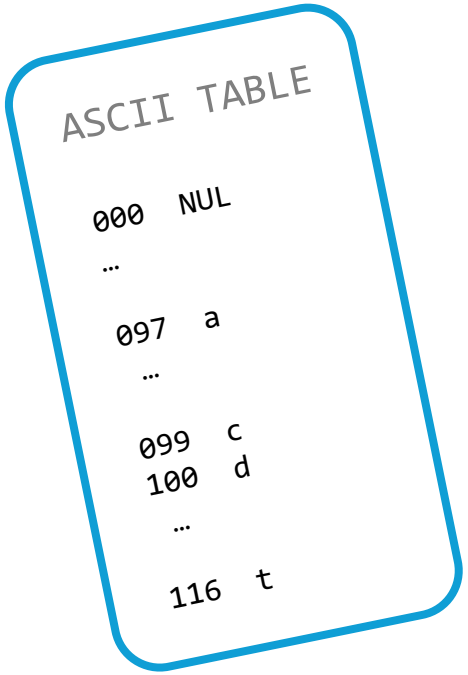
A 4 B 5 C 6 D 7

4 char + NULL terminator

QUIZ

What will this code print?

```
char text[] = "cadt";
for (int i = 0; i < sizeof(text); i++) {
    printf("%d ", text[i]);
}
```



A. 99 97 100 116 0

B. c a d t \0

C. 99 97 100 116

D. c a d t

ANSWER

What will this code print?

```
char text[] = "cadt";
for (int i = 0; i < sizeof(text); i++) {
    printf("%d ", text[i]);
}
```

ASCII value

Null terminator value

ASCII TABLE

000 NUL
…
097 a
…
099 c
100 d
…
116 t

A. 99 97 100 116 0

B. c a d t \0

C. 99 97 100 116

D. c a d t

EXPLAIN

Display **strings**

- To output the string, we can use the printf format specifier **%s**

```
char greetings[] = "Hello CADT!";

printf("%s", greetings);
```

```
Hello CADT!
```

- You can also access **each character**, using the square brackets []

```
char greetings[] = "Hello CADT!";

printf("%c", greetings[4]);
```

```
o
```

ACTIVITY

Let's **Try** !

ONLINE DEBUGGER

✓Complete the code to print the string **character by character**

```
char text[] = "I love IT";
int index =0;

while ( ____________________) {
    printf("%c", text[index]);
    index++;
}
```

```
I love IT
```

You cannot use the %s formatter

ANSWER

Let's **Try** !

✓Complete the code to print the string **character by character**

```
char text[] = "I love IT";
int index =0;

while (text[index] != '\0') {
    printf("%c", text[index]);
    index++;
}
```

```
I love IT
```

We stop at the Null terminator value

You cannot use the %s formatter

EXPLAIN

Strings and **Pointers**

Like any array, a string name in C acts as a **pointer** to its first character.

```
char text[] = "ABC";
char* ptr = text;

char first = *ptr;                  // A
char second = *(ptr + 1);           // B
char third = *(ptr + 2);            // C

while (*ptr != '\0') {
    printf("%c", *ptr);             // ABC
    ptr++;
}
```

QUIZ

What will this code print?

```
char text[12] = "ABC";
char* ptr = text;

printf("%d ", sizeof(text));
printf("%d ", sizeof(ptr));
```

Note : a **pointer size** is 8 bytes

A. 12 8

B. 3 8

C. 3 3

D. 12 12

ANSWER

What will this code print?

```
char text[12] = "ABC";
char* ptr = text;

printf("%d ", sizeof(text));
printf("%d ", sizeof(ptr));
```

Note : a **pointer size** is 8 bytes

A. 12 8

- 12 bytes allocated
- The pointer size is 8 bytes

B. 3 8

C. 3 3

D. 12 12

EXPLAIN

Input **strings**

To input a string, use scanf with the **string format specifier** (%s) :

Allocate enough space

Scan with the string format

```
char color[12] = {};
printf("Enter your color: ");

scanf("%s", color);
printf("You entered: %s", color);
```

Store the result from this pointer

scanf does **not read space :**

```
Input: Red Green
Value of color: "Red"
```

Scan stopped At the space

Note: we will see another function (fgets) to scan strings including spaces.

EXPLAIN

Memory **allocation** and **modifiability**

String in char array (char t[]) lets you modify the text, while pointer to string literal (char* t) does not.

```
char text[] = "HELLO";
```

- **Copy** the text "HELLO" into an array
- We **allocate 6 bytes** in memory

- We can **change** the character in the text":

```
text[1] = '9';  // works fine
```

```
char* ptrChar = "HELLO";
```

- ptrChar **points to** the original text "HELLO"
- We **allocate space for the pointer only**

- We cannot **change** the character in the text

```
ptrChar[1] = '9';  // error
```

"HELLO" is a **string literal**
= a constant which cannot be modified

DISCUSS

Why do we have a **segmentation fault** here?

```
char *color;
printf("\nEnter your favorite color: ");

scanf("%s", color);
printf("\nYou entered: %s", color);
```

```
Enter your favorite color: yellow!!!!!!!!!!!!!!!!!!!!!!!!!!

Segmentation fault
```

ANSWER

Why do we have a **segmentation fault** here?

```
char *color;
printf("\nEnter your favorite color: ");

scanf("%s", color);
printf("\nYou entered: %s", color);
```

We are using a pointer that points nowhere.

scanf("%s", color) tries to write to **invalid memory**,

segmentation fault

```
Enter your favorite color: yellow!!!!!!!!!!!!!!!!!!!!!!!!!

Segmentation fault
```

To fix it :

```
char color[100];
```

We use a character array (which allocates space)

EXPLAIN

List of strings

- We can represent a list of strings using an array 2D:

```
char colors[3][10] = { "Red", "Green", "Blue" };

for (int i = 0; i < 3; i++) {
    printf("%s\n", colors[i]);
}
```

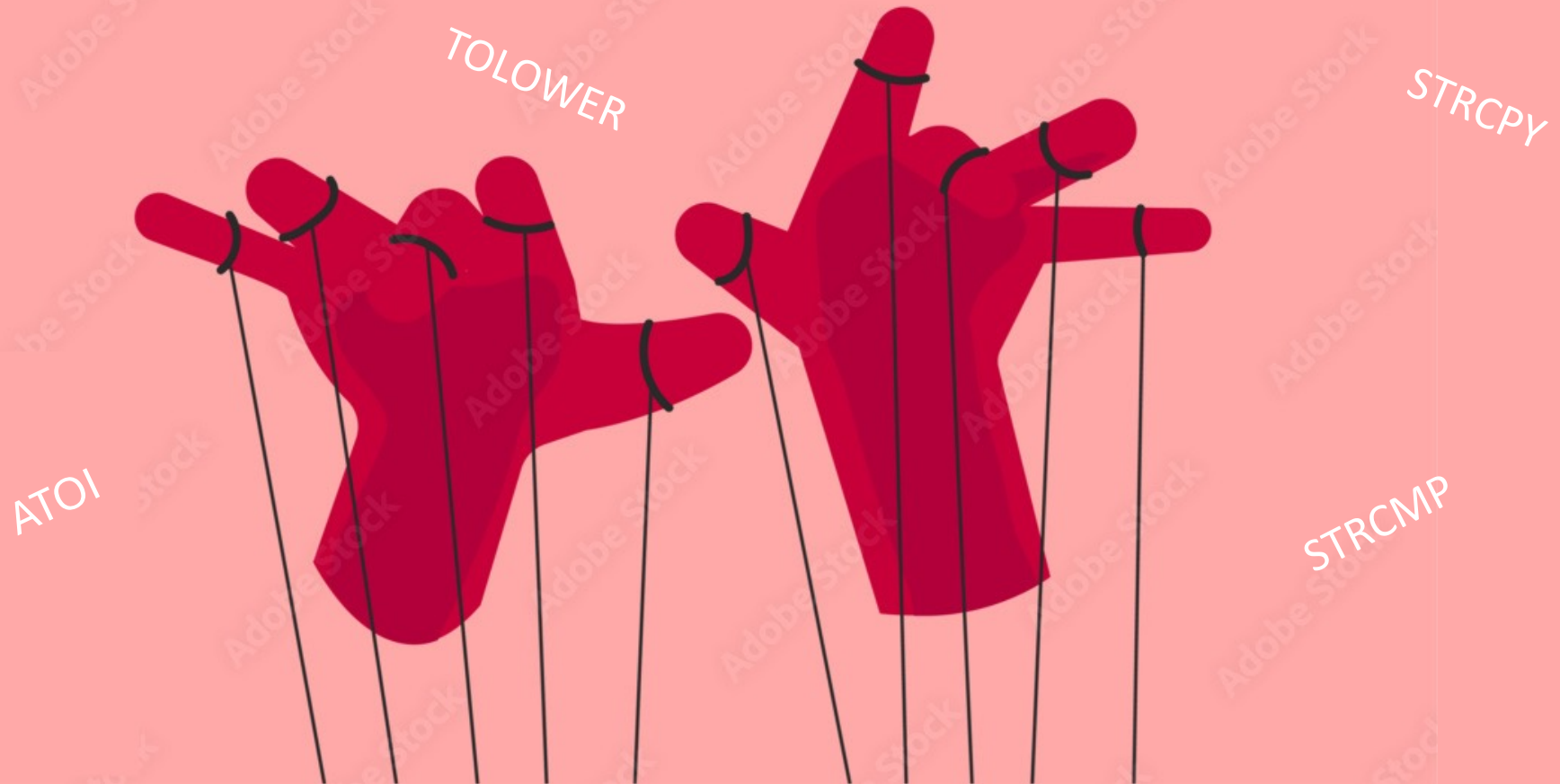
```
Red
Green
Blue
```

- We can scan each string of the list using the scanf approach:

```
char colors[3][10] = {};

scanf("%s %s %s", colors[0], colors[1], colors[2]);
```

Libs for String **manipulations**

`string.h` `stdlib.h` `ctype.h`



EXPLAIN

Convert **Strings** to **Numbers**...

The library `stdlib.h` provides functions to convert **strings to numbers**.

CHECK THE LIB !

`stdlib.h`

Function	Parameter	Return	Comment
atoi	const char *str	int	Convert a string to a integer
atof	const char *str	double	Convert a string to a float

```
#include <stdlib.h>

..

char str1[] = "123";
char str2[] = "45.67";

int num1 = atoi(str1);        //   "123" to int → 123
double num2 = atof(str2);     //   "45.67" to double → 45.67
```

Why const parameters

EXPLAIN

Compute the **length** of a string

The length of the string is the number of its characters,
excluding null terminator

CHECK THE LIB !

`string.h`

Function	Parameters	Return Type	Description
strlen	const char* str	int	Returns the **length of the string** - *Excluding null terminator*

```
#include <string.h>

..

char text[] = "abcd";
int lenght  = strlen(text);     // 4 characters
```

Why const parameters

EXPLAIN

Convert to **upper** or **lower** case

CHECK THE LIB !

The character-handling library <ctype.h> provides many character manipulation functions such as tolower() and toupper().

ctype.h

Function	Parameters	Return Type	Description
tolower	char	char	Converts a character to **lowercase** if it's an uppercase
toupper	char	char	Converts a character to **uppercase** if it's a lowercase

```
#include <ctype.h>

..

char a = 'H';
char b = 'm';

char lowerA = tolower(a);          // 'H' → 'h'
char upperB = toupper(b);          // 'm' → 'M'
```

EXPLAIN

Copy String

The function **strcpy** copies the contents of one string into another

CHECK THE LIB !

string.h

Function	Parameters	Return Type	Description
strcpy	char* dest, const char* src	char*	Copies the **string** from **src** to dest - *Including null terminator*

```
#include <string.h>

..

char src[] = "Hello";
char dest[20];

strcpy(dest, src);  // Copy src into dest
printf("After strcpy, dest = %s\n", dest);
```

Why const parameters for scr and not for dest?

EXPLAIN

Compare Strings

strcmp compares two strings and indicates their lexicographical order or equality.

string.h

Function	Parameters	Return Type	Description
strcmp	const char* s1, const char* s2	int	**Compares** two strings • *0 if equal,* • *-1 if s1 < s2* • *1 if s1 > s2*

```
#include <string.h>

..

char str1[] = "cat";
char str2[] = "dog";

int result = strcmp(str1, str2);     // -1 as dog > cat
```

Why const parameters ?

EXPLAIN

Concatenate Strings

strcat appends one string to the end of another.

string.h

Function	Parameters	Return Type	Description
strcat	char* dest, const char* src	char*	**Appends** the string src to the end of dest

```
#include <string.h>

..

char greeting[] = "Hello, ";
char name[] = "Alice";

strcat(greeting, name);

printf("%s\n", greeting);  // Output: Hello, Alice
```

Why const parameters for scr and not for dest?

QUIZ

QUIZ

What will this code print?

```
char str1[10] = "Hi";
char str2[] = "there";

strcat(str1, str2);
printf("%s", str1);
```

strcat()

concatenates or glues one string to another

A. Hi there

B. Hithere

C. Compilation error

D. Segmentation fault

ANSWER

What will this code print?

```
char str1[10] = "Hi";
char str2[] = "there";

strcat(str1, str2);
printf("%s", str1);
```

strcat()

concatenates or glues one string to another

A. Hi there

B. Hithere

C. Compilation error

D. Segmentation fault

QUIZ

What will this code print?

```
printf("%d", strcmp("abc", "Abc"));
```

strcmp(s1, s2)

0	if equal,
-1	if s1 < s2
1	if s1 > s2

A. 0

B. -1

C. 1

D. Undefined behavior

ANSWER

What will this code print?

```
printf("%d", strcmp("abc", "Abc"));
```

strcmp(s1, s2)

0	if equal,
-1	if s1 < s2
1	if s1 > s2

A. 0

B. -1

C. 1

D. Undefined behavior

QUIZ

What will this code print?

```
char s[] = "Hello";
s[2] = '\0';
printf("%s", s);
```

It's tricky!

A. Hello

B. He

C. H\0llo

D. Error

ANSWER

What will this code print?

```
char s[] = "Hello";
s[2] = '\0';
printf("%s", s);
```

It's tricky!

A. Hello

B. He

C. H\0llo

D. Error

QUIZ

What will this code print?

```
char str[] = "Code";
printf("%d", sizeof(str));
```

A. 4

B. 5

C. 6

D. It depends on the compiler

ANSWER

What will this code print?

```
char str[] = "Code";
printf("%d", sizeof(str));
```

A. 4

B. 5

C. 6

D. It depends on the compiler

QUIZ

What will this code print?

```
char str[] = "game";
char *p = str + 2;
printf("%c", *p);
```

A. g

B. a

C. m

D. e

E. /0

F. Compilation Error

QUIZ

What will this code print?

```
char str[] = "game";
char *p = str + 2;
printf("%c", *p);
```

A. g

B. a

C. m

D. e

E. /0

F. Compilation Error

QUIZ

Which declaration creates a copy of the string that you can safely modify?

A)
```
char* t = "abc";
```

B)
```
char t[] = "abc";
```

C)
```
char* t;
t = "abc";
```

ANSWER

Which declaration creates a copy of the string that you **can safely modify**?

```
A)  char* t = "abc";

B)  char t[] = "abc";

C)  char* t;
    t = "abc";
```

Create a string literal
(= constant in memory)

And link the pointer to this constant

Allocate 4 bytes in memory
Which can be modified later on

Wrap Up !

- ✓ We create and initialize a **string** using a **character array** and a terminating **NULL** character.
- ✓ **String literals** are a series of characters surrounded by **double quotes**.
- ✓ We use printf() function with the **%s** conversion specifier to print a string

- ✓ **C libraries** provide various functions to manipulate char and string :

• atoi() atof()	Converts a string to an integer, double
• strlen()	takes a reference to a string and returns the numeric string length
• tolower() toupper()	convert a single character to lowercase, uppercase
• strcpy()	copies the contents of one string into another string
• strcat()	concatenates or glues one string to another
• strcmp()	compare two strings for equality

What you **know now**

- ✓ Represent **strings** using **arrays** and **null-terminators**.
- ✓ Use **pointers** to access and modify **strings**.
- ✓ Read and print strings using **standard I/O functions**.
- ✓ Handle multiple **strings** using **2D arrays** or arrays of **pointers**.
- ✓ Convert **strings** to numeric values using standard functions.
- ✓ Manipulate **strings** using **built-in C library functions**.

Go further after the class…

Strings
https://www.w3schools.com/c/c_strings.php

String functions
https://www.w3schools.com/c/c_strings_functions.php

C Standard Library Functions
https://www.programiz.com/c-programming/library-function

C Standard library functions or simply C Library functions are inbuilt functions in C programming.

The prototype and data definitions of these functions are present in their respective header files. To use these functions we need to include the header file in our program. For example,

If you want to use the `printf()` function, the header file `<stdio.h>` should be included.

```
#include <stdio.h>
int main()
{
   printf("Catch me if you can.");
}
```

Run Code »

Explore C standard library functions