CS 262 Lecture 7: Strings

GEORGE MASON UNIVERSITY

Overview of Lecture 7

String Functions

Recap of string basics

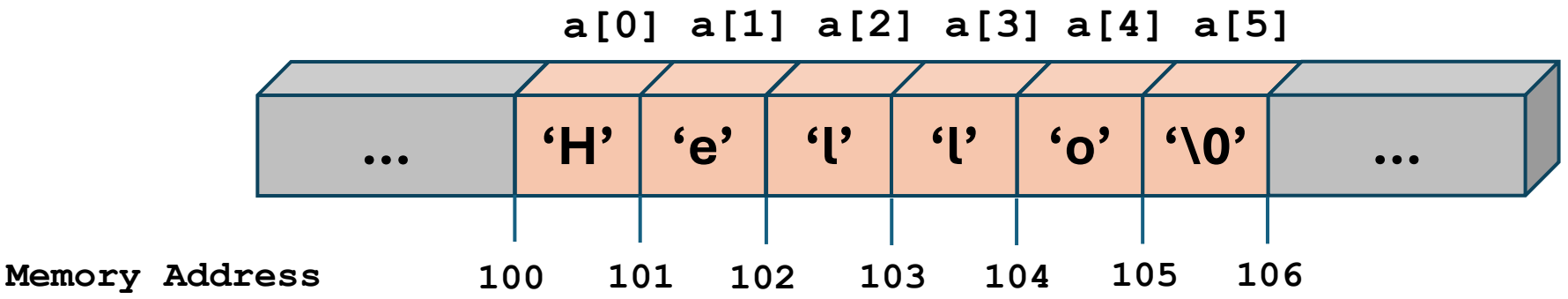
Revisiting familiar functions

Learning some new string functions

String Basics - Recap

Strings are **char** arrays that have a special character '\0' at the end known as the **Null Terminator**

```
char a[] = "Hello";
```



String Basics - Recap

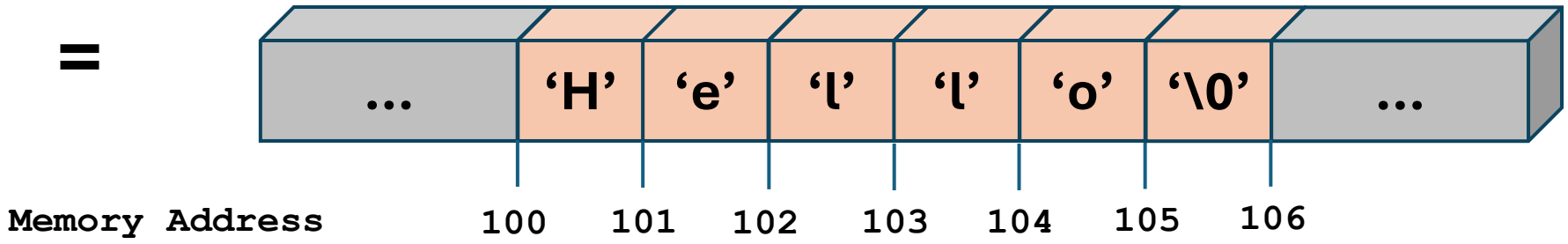
There are a couple ways of initializing a string

```
char arr[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

=

```
char arr[] = "Hello";
```

=



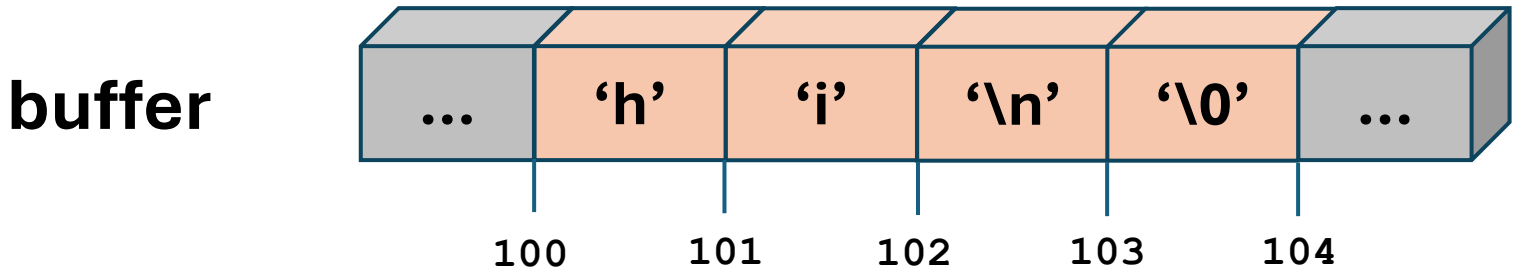
fgets: Looking At The Details

fgets stores all input characters, including the newline character '\n' which results from us hitting enter

```
char buffer[100];
int buffer_size = 100;
printf("Enter the word hi\n");
fgets(buffer, buffer_size, stdin);
printf("%s", buffer);
printf("hmm");
```

Terminal

```
hi
hmm
```



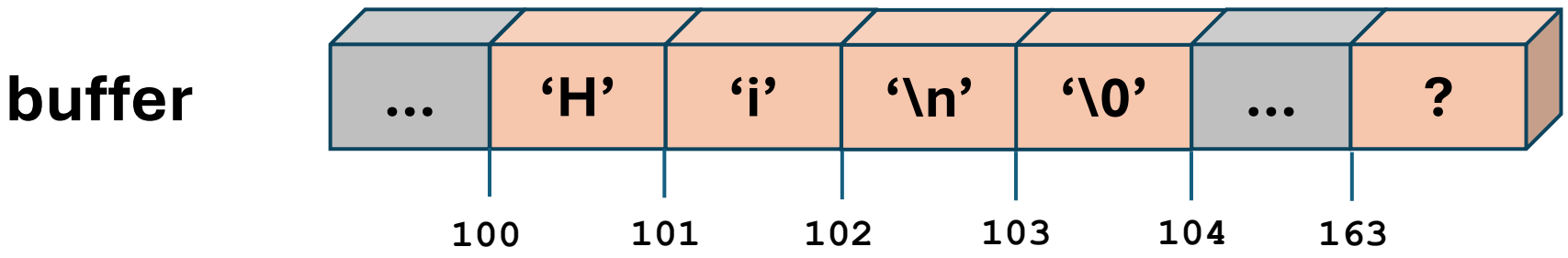
sscanf: Looking At The Details

Note: `sscanf` when used with `%s` won't read the newline character

```
char word[100];
sscanf(buffer, "%s\n", word);
```

Terminal

```
hi
```



sscanf: Looking At The Details

sscanf stops as soon as it sees a space

```
char buffer[100];
char phrase[100];
int buffer_size = 100;
printf("Enter the phrase: hello 262\n");
fgets(buffer, buffer_size, stdin);
sscanf(buffer, "%s\n", phrase);
printf("%s", phrase);
```

Terminal

```
hello
```

sscanf: Looking At The Details

If we want to print out everything, we just print directly from the buffer

```
char buffer[100];
char phrase[100];
int buffer_size = 100;
printf("Enter the phrase: hello 262\n");
fgets(buffer, buffer_size, stdin);
printf("%s", buffer);
```

Terminal

```
hello 262
```

.. Or we can use ..

sscanf: Looking At The Details

```
char buffer[100];
char phrase[100];
int buffer_size = 100;
printf("Enter the phrase: hello 262\n");
fgets(buffer, buffer_size, stdin);
sscanf(buffer, "%[^\n]", word);
printf("%s\n", phrase);
```

Terminal

```
hello 262
```

String Functions – strlen

int strlen(char *str)

This function returns the number of characters in a string, excluding the null terminator

```
char str[] = "Hello";
printf("length of str: %d\n", strlen(str); // This will print out 5
```

```
char str[] = {'H', 'e', 'l', 'l', 'o', '\0'};
printf("length of str: %d\n", strlen(str); // This will print out 5
```

String Functions – toupper, tolower

int toupper(char c)

This function converts a character to uppercase

```
char ch = 'e';
ch = toupper(ch);
printf("ch is now %c\n", ch); // This will print out E
```

int tolower(char c)

And this one converts a character to lowercase

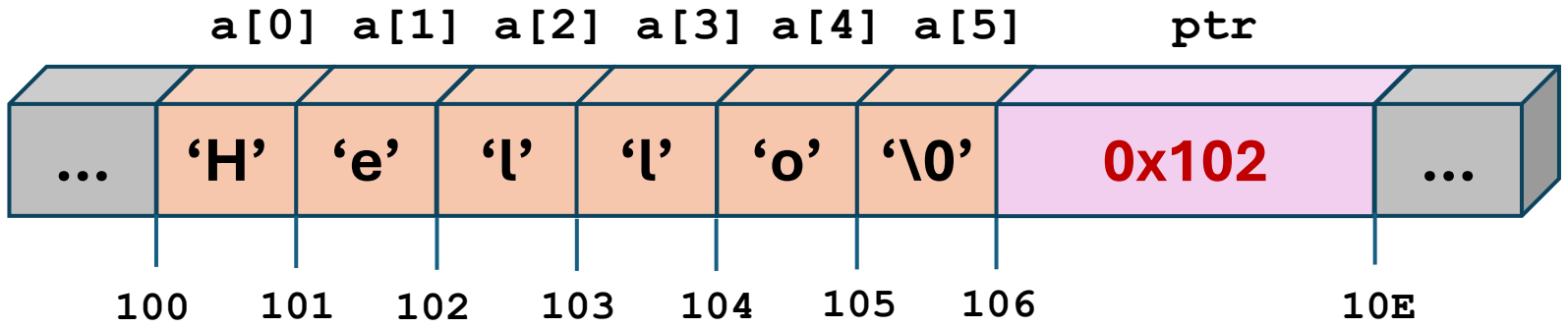
```
char ch = 'E';
ch = tolower(ch);
printf("ch is now %c\n", ch); // This will print out e
```

String Functions – strchr

char *strchr(char *str, char c)

This function returns a pointer to the first occurrence of the character **c**

```
char a[] = "Hello";
char *ptr = strchr(a, 'l');
```



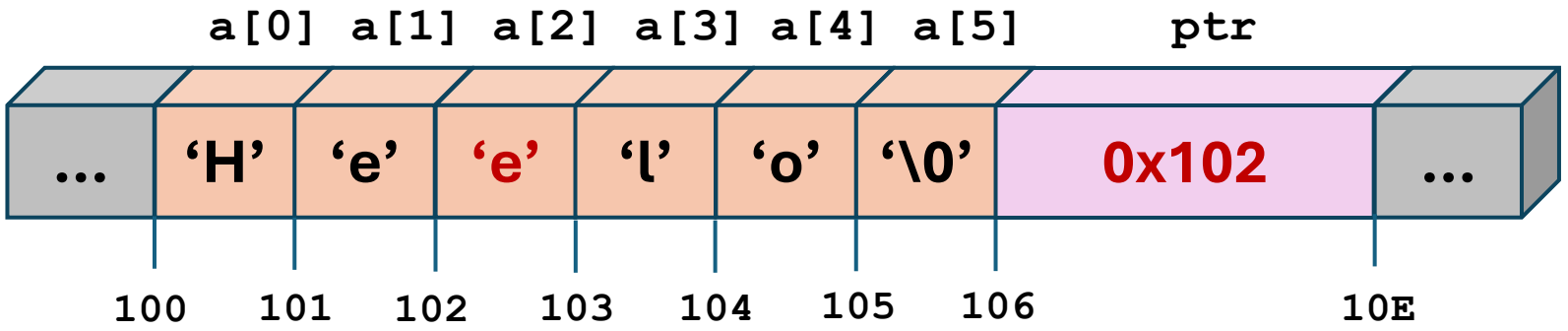
String Functions – strchr

We can even use this to change a specific character within a string

We can use **strchr** to find a character, then change the char at ptr

Assume we want to change the first ‘l’ to ‘e’

```
char a[] = "Hello";
char *ptr = strchr(a, 'l');
*ptr = 'e';
```

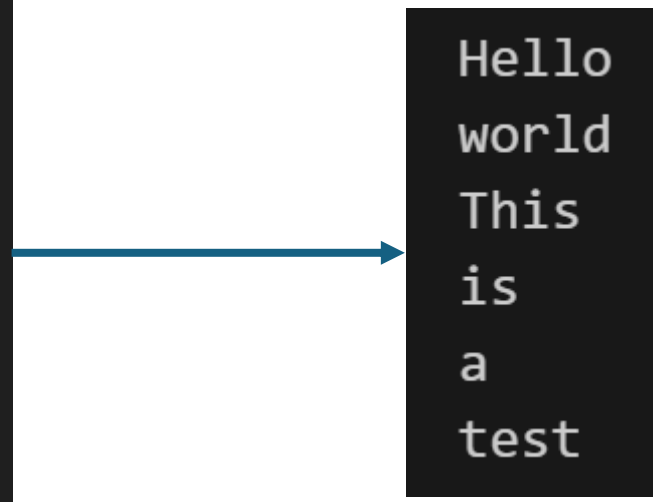


String Functions – strtok

char *strtok(char *str, char *delim)

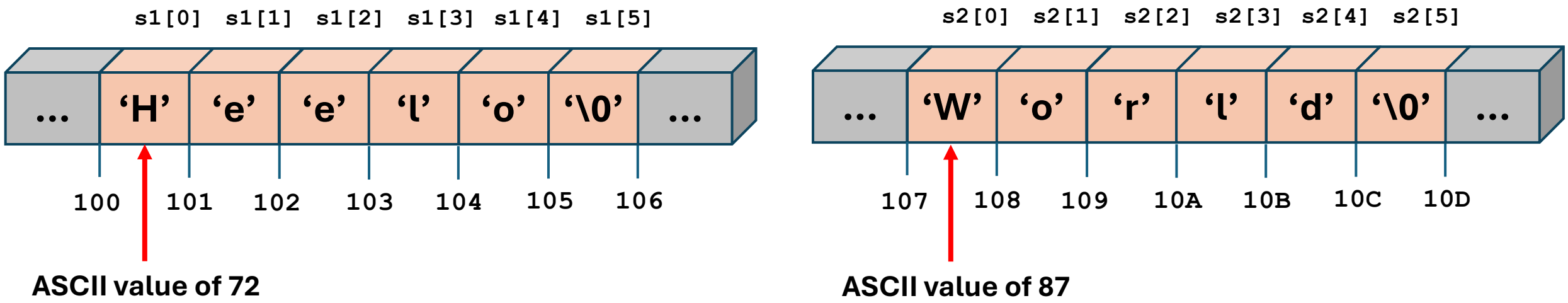
This function gets each ‘word’ in str separated by *any* character in string *delim

```c
int main() {
    char str[] = "Hello world\nThis is a test\n";
    char *token;
    // Get the first token
    token = strtok(str, " \n");
    // Walk through other tokens
    while (token != NULL) {
        printf("%s\n", token);
        token = strtok(NULL, " \n");
    }
    return 0;
}
```



String Functions – strcmp

```
char s1[] = "Hello";
char s2[] = "World";
printf("the result is %d", strcmp(s1, s2);
```



strcmp will compare the ASCII values of each character in the string until it finds two that are not equal

String Functions – strcmp

int strcmp(char *s1, char *s2)

This function compares strings s1 and s2

It returns 0 if s1 == s2, positive if s1 > s2, and negative if s1 < s2

.. But what does it mean for a string to be less than or greater than another string?

```
char s1[] = "Hello";
char s2[] = "World";
printf("the result is %d", strcmp(s1, s2);
```

String Functions – strstr

char *strstr(char *s1, char *s2)

This function is similar to **strchr**, except it returns a pointer to the **substring** matching s2 within s1

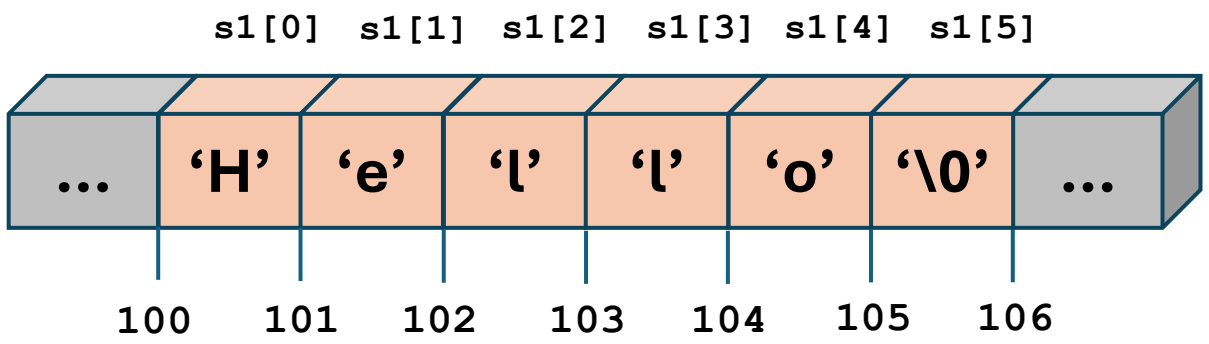
```
4   int main() {
5       const char *string = "Hello, and welcome to CS 262";
6       const char *substring = "CS";
7
8       char *result = strstr(string, substring);
9
10      if (result) {
11          printf("Substring found at position: %ld\n", result - string);
12      } else {
13          printf("Substring not found.\n");
14      }
15
16      return 0;
17  }
```

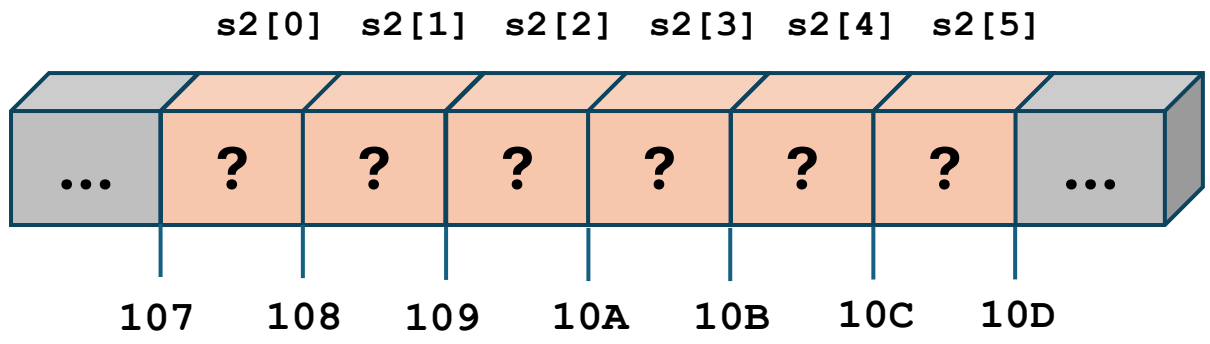
String Functions – strcpy

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

This function copies s1 into s2

```
char s1[] = "Hello";
char s2[100];
// strcpy(s2, s1);
```



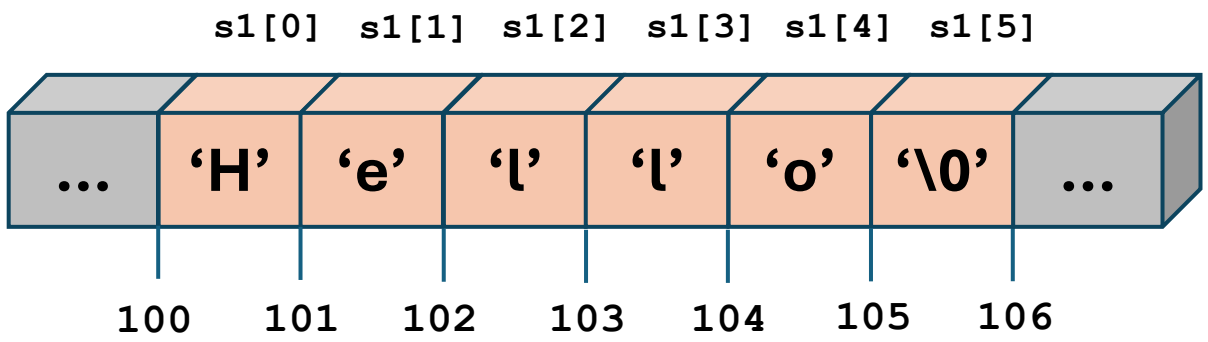


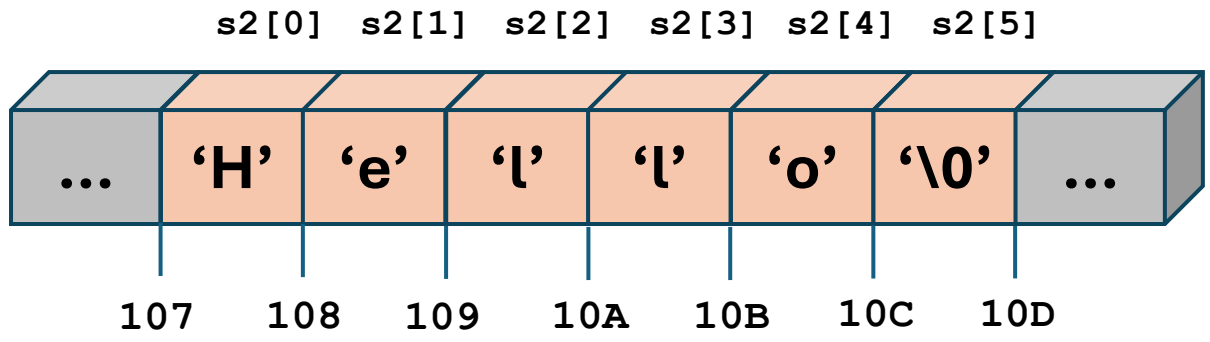
String Functions – strcpy

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

This function copies s1 into s2

```
char s1[] = "Hello";
char s2[100];
strcpy(s2, s1);
```





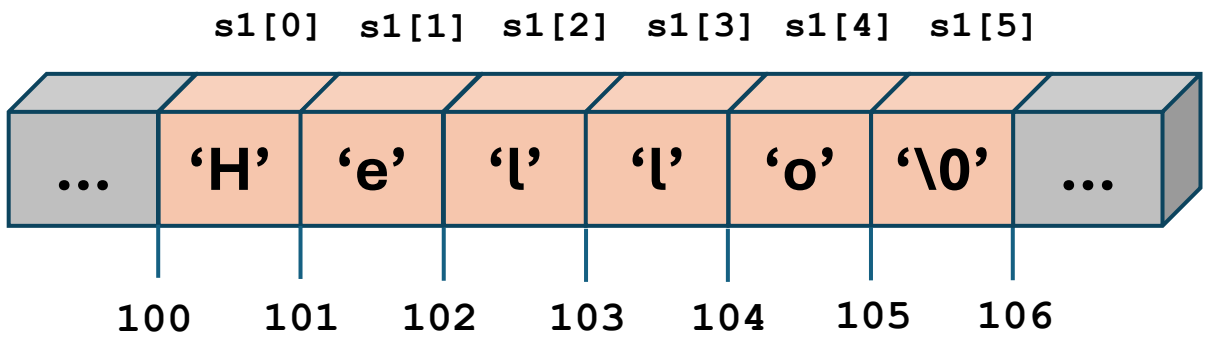
Note: strcpy is unsafe, as it keeps copying until it sees '\0'

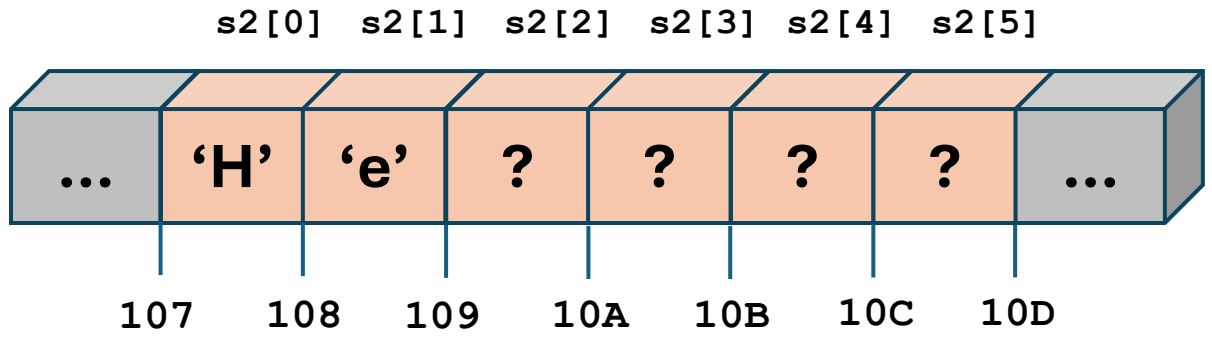
String Functions – strncpy

char *strncpy(char *s2, char *s1, int n)

This is a safer option, as it copies n chars from s1 into s2

```
char s1[] = "Hello";
char s2[100];
strncpy(s2, s1, 2);
```





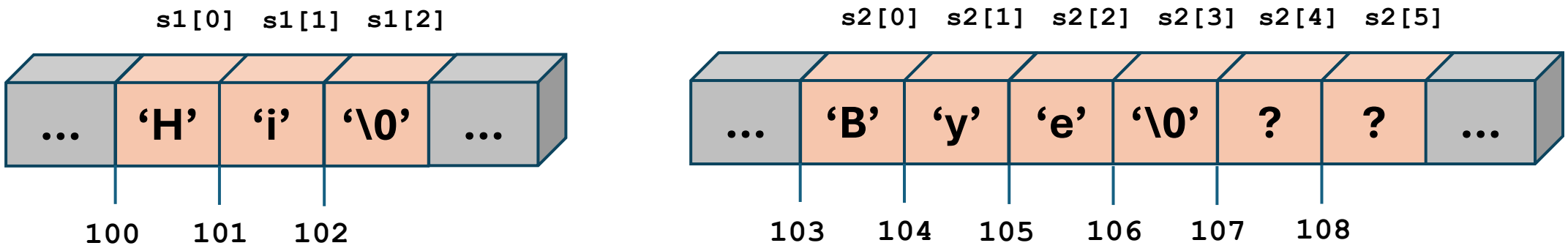
Note: strncpy doesn't add a null terminator after copying n characters

String Functions – strcat

char *strcat(char *s2, char *s1)

This function concatenates s1 onto the end of s2

```
char s1[] = "Hi";
char s2[10] = "Bye";
// strcat(s2, s1);
```

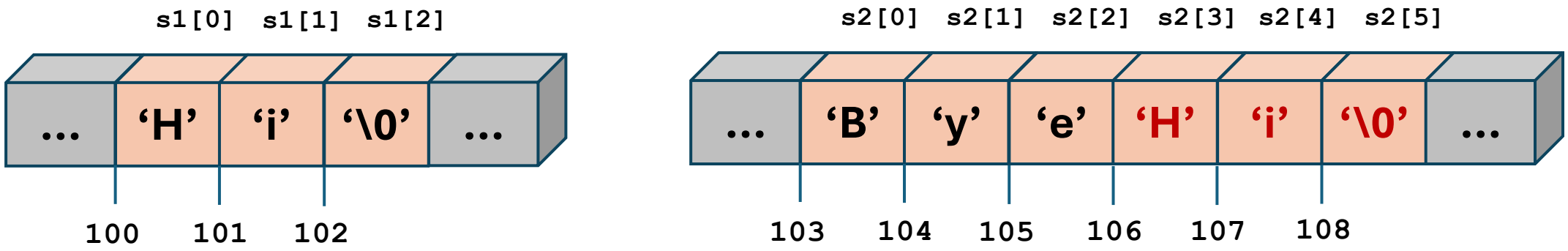


String Functions – strcat

char *strcat(char *s2, char *s1)

This function concatenates s1 onto the end of s2

```
char s1[] = "Hi";
char s2[10] = "Bye";
strcat(s2, s1);
```



But you guessed it .. strcat is unsafe, and keeps copying until it sees a null terminator

String Functions – strncat

char *strncat(char *s2, char *s1, int n)

This function concatenates n+1 bytes from s1 onto the end of s2

```
char s1[] = "Hi";
char s2[10] = "Bye";
strcat(s2, s1, 3);
```