Introduction to string

Definition:

String is a sequence of characters enclosed within double quotes.

For example :

“Hello everyone”

Double quotes are important

%s is a placeholder

```
Int main(){
printf(“%s”, ”Hello students”);
return 0;
}
```

Continuing string

Printf(“%s”, “Learning programming for the first time may be confusing.
but once you’ve learned it, you can master it.”);

Error

Printf(“%s”, “Learning programming for the first time may be confusing.\
but once you’ve learned it, you can master it.”);

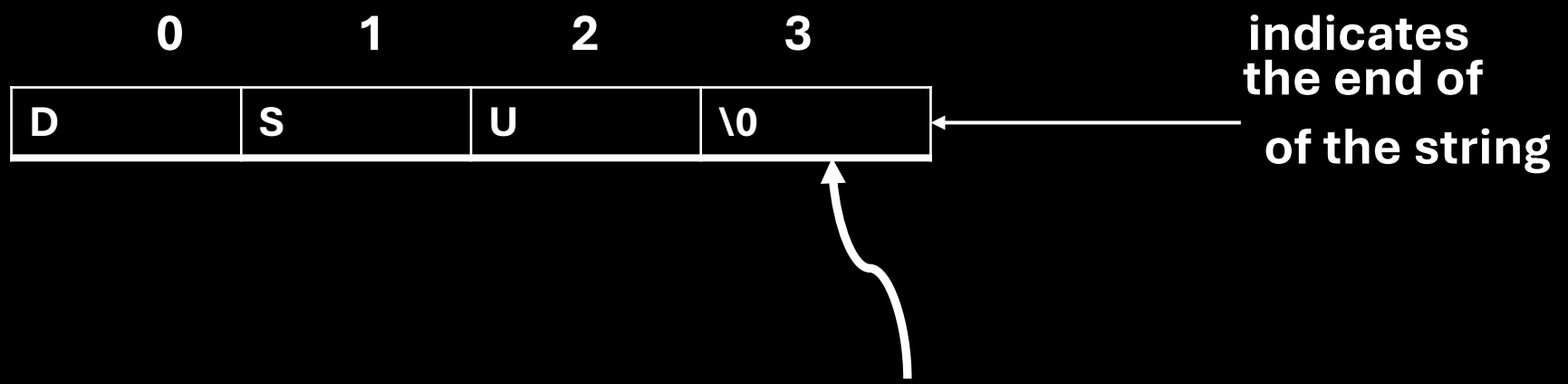
No Error

in C this is called “splicing”

Storing string

printf(“DSU”)

“DSU” is a string



First argument to printf and scanf function is always a string.

But what we are passing to the printf/scanf?

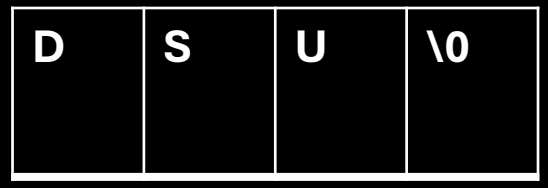
String are stored as an array of characters

Total 4 bytes of read-only memory is allocated to the above string literal

‘\0’ character must not be confused with 0 character.

Both have different Ascii codes. \0 has the code 0 while 0 has the code 48

But what we are actually passing to the printf/scanf?



In c compiler treats a string as pointer to the first character.

So, to the printf or scanf, we are passing a pointer to the first character of a string

Both printf and scanf functions expects a character pointer(char *) as an argument

passing DSU is equivalent to passing the pointer to letter D

Declaring and initializing the string variables

- A string is a one dimensional array of characters that is capable of holding a string at a time.

- For example: char s[6];

- Note: Always make the array one character longer than the string. If length of the string is 5 characters long than don't forget to make extra room for Null character.
- Failing to do the same may cause unpredictable results when program is executed as some C libraries assume the strings are Null terminated.

initializing the string variables

- For example: char s[6]=“Hello”;

- s

H	e	l	l	o	\0

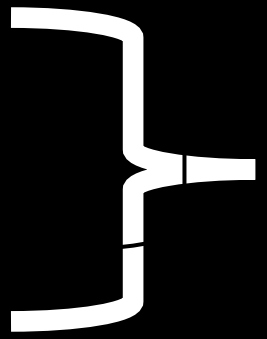
Although, it seems like "Hello" in the above example is a string literal, but it is not.

When a string is assigned to a character array, then this character array is treated like other types of arrays. We can modify its characters.

initializing the string variables

char s[6]=“Hello”;

char s[6]={‘H’ ,’e’ , ‘l’ , ‘l’ , ‘\0’};



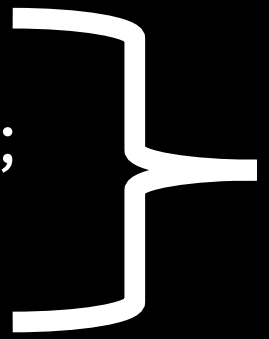
They both are equal

Recall: We cannot modify a string literal

char *ptr= “Hello”; Error

ptr=“m”

initializing the string variables

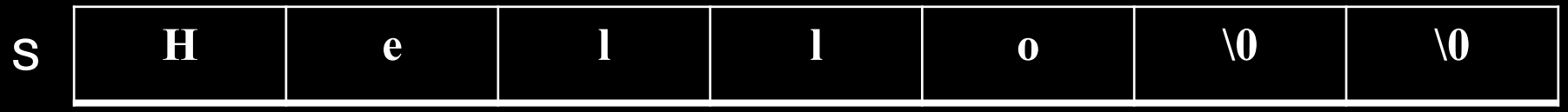


```
char s[6]="Hello";
char s[0]="M"
```

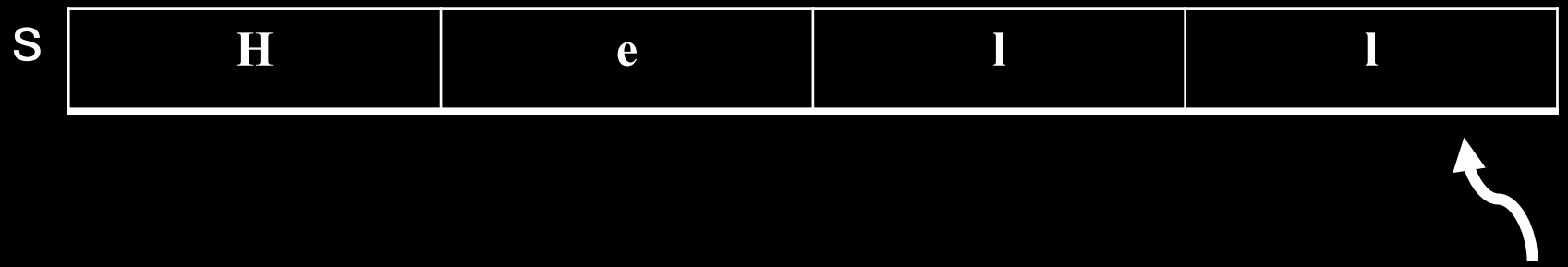
No problem

Short and long length initializer

char s[7]="Hello";



char s[4]="Hello";



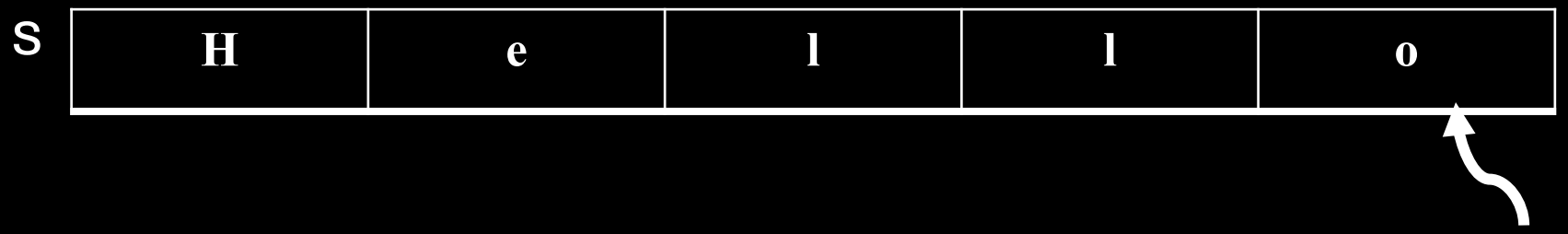
printf("%s",s);

Rest of the part is truncated

Warning: initializer-string for array of chars is too long

Equal length initializer

char s[4]=“Hello”;



printf(“%s”,s);

Warning: initializer-string for array of chars is too long

char s[]=“Hello”;

Automatically, the compiler sets aside 6 characters for s which is enough to store the string "Hello" with null character.

writing string using print

char s[4]=“Hello”;

printf(“%s”,s);

- In order to print the string is very simple.
- Like printf within this printf function the two arguments one is format specifier and the other one is string variable.

- “%.ns” is used to print just a part of the string where n is the number of characters to be displayed on the screen.

writing string using print

- “%m.ns” is used to printf just a part of the string where n is the number of characters to be displayed and m denotes the size of the field within which the string will be displayed .
- Char s[11]=“Programming”;
- Printf(“%.5s”,s);
- Printf(“%6.5s”,s);

- Output: Progra
- Progra

writing string using puts

- Puts() function is a function declared in <stdio.h> library and is used to write strings to the output screen.
- Also, puts() function automatically writes a newline character after writing the string to the output screen.
- Char s[11]=“Programming”;
- puts(s);
- puts(s);

- Output: Prog
- Prog

Reading string using scanf()

- Using scanf(),we can read a string into a string variable(character array)

```
char a[10];
printf(“Enter the string:\n”);
scanf(“%s”,a);
printf(“%s”,a);
```

Like any array name, a is treated as a pointer to the first element of the array . Therefore , there is no need to put &

Input: You are Learning String

Output: You

Why only You is stored?

Reading string using scanf()

- scanf () doesn't store the white space characters in the string variable
- It only reads characters other than white spaces and store them in the specified character array until it encounters a white-space character.

- Input: You are Learning String
-

When white-space is seen by scanf , it stops and hence, only "You' is stored in the specified character array.

Reading string using gets()

- To read an entire line of input, gets () function can be used.

- char a[10];
- printf(“Enter the string:\n”);
- gets(a);
- printf(“%s”,a);

- Input: You are Learning String

- Program may crash

Reading string using gets()

- Both, gets() and scanf() functions have no way to detect when the character array is full.

- Both never checks the maximum limit of input characters. Hence ,they may cause undefined behavior and probably lead to buffer overflow error which eventually causes the program to crash.

- Although, scanf has the way to set the limit for the number of characters to be stored in the character array.
- By using %ns, where n indicates the number of characters allowed to store in the character array.

Reading string using gets()

- For example,

```
char a[10];
printf("Enter the string:\n");
scanf("%9s", a);
printf("%s", a);
```

Input: YouareLearningString

Output : YouareLea

Reading string using gets()

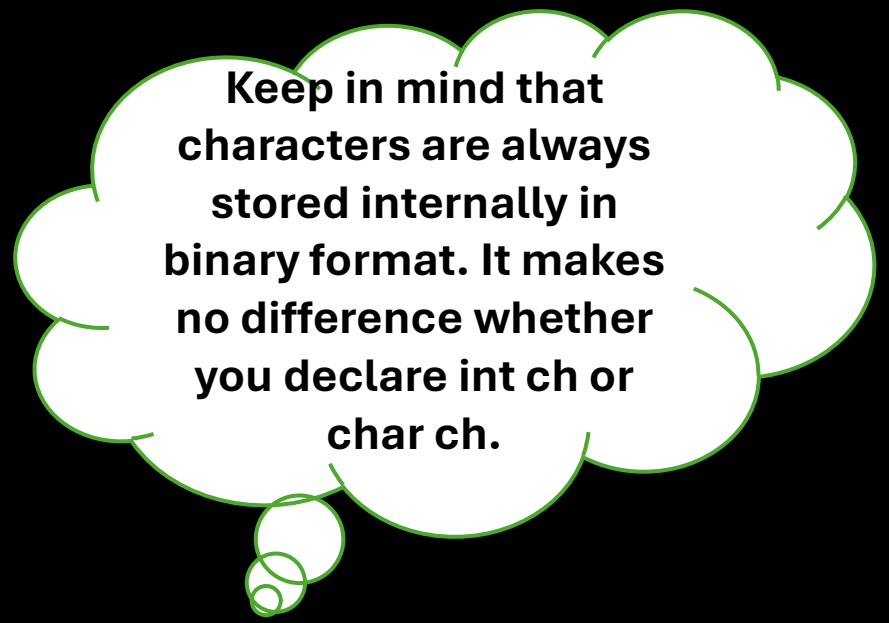
- But , unfortunately , gets() is still UNSAFE.
- It will try to write the characters beyond the memory allocated to the character array which is unsafe because it will simply overwrite the memory beyond the memory allocated to the character array.

Hence, it is advisable to not use the gets( ) function.

Putchar()

- Prototype: int putchar(int ch)

- putchar accepts an integer argument(which represents a character it wants to display) and returns an integer representing the character written on the screen.



Putchar()

```
#include <stdio.h>
int main(){
int ch;
for(ch = 'A'; ch <= 'Z'; ch++)
Putchar(ch);
return 0;
}
```

```
#include <stdio.h>

int main() {
    char ch;
    for (ch = 'A'; ch <= 'Z'; ch++) {
        printf("%c ", ch);
    }
    return 0;
}
```

String Library

There are some operations which we can perform on strings.

For example : Copy strings, concatenate strings, select substrings and so on.

- These are the operations which we can perform.
- Our program may require these operations.

- <string.h> library contains all the required functions for performing string operations.
- So, we just have to include this header file: #include <string.h>in our program and we are good to go.

STRLEN (String length) function

Prototype: size_t strlen(const char* str);

strlen() function is used to determine the length of the given string.

To the strlen function, we should pass the pointer to the first character of the string whose length we want to determine.

And as a result, It returns the length of the string.

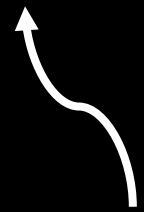
It doesn’t count Null character

STRLEN (String length) function

```
#include <stdio.h>
#include <string.h>
int main() {
char str[] = "Hello World";
printf("%d", strlen(str));
return 0;
}
```

STRCPY(String copy) function

Prototype : char* strcpy(char* destination , const char* source)



It isn’t modified. That is why it is constant.

Why the const

Because in the whole process a source it is not modified at all.

We are simply copying the content source to destination.

strcpy is used to copy a string pointed by source (including NULL character)to the destination (character array)

STRCPY(String copy) function

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
char str1[10] = "Hello";
char str2[10];
printf("%s\n", strcpy(str2, str1));
printf("%s", str2);
return 0;
}
```

strcpy returns the pointer to the first character of the string which is copied in the destination. Hence if we use %s, then whole string will be printed on the screen.

Output: Hello

STRCPY(String copy) function

We can also chain together a series of strcpy calls

```
#include <stdio.h>
#include <string.h>
int main() {
char str1[10] = "Hello";
char str2[10];
char str3[10];
strcpy(str3,strcpy(str2,str1));
printf("%s %s", str2,str3);
return 0;
}
```

STRCPY(String copy) function

- In the call to strcpy(str1, str2), there is no way the strcpy will check whether the string pointed by str2 will fit in str1
- If the length of the string pointed by str2 is greater than the length of the character array str1 then it will be an undefined behavior.

- To avoid this we can call strncpy function.

STRCPY(String copy) function

strncpy(destination, source, sizeof(destination));
Example:

```
#include <stdio.h>
#include <string.h>
int main() {
char str1[6] = "Hello";
char str2[4];
strncpy(str2, str1, sizeof(str2));
printf("%s", str2);
return 0;
}
```

STRCPY(String copy) function

But if we try to copy the string using strcpy, it causes undefined behaviour.
Example:

```
#include <stdio.h>
#include <string.h>
int main() {
char str1[6] = "Hello";
char str2[4];
strcpy(str2, str1);
printf("%s", str2);
return 0;
}
```

STRNCPY(String copy) function

strncpy will leave the string in str2 (destination) without a terminating NULL character, If the size of str1 (source) is equal to or greater than the size of str2 (destination).

```
#include <stdio.h>
#include <string.h>
int main() {
char str1[6] = "Hello";
char str2[4];
strncpy(str2, str1, sizeof(str2));
str2[sizeof(str2)-1]='\0';
printf("%s", str2);
return 0;
}
```

STRCAT (String concatenate function

Prototype: char* strcat(char* str1, const char* str2);

Strcat function appends the content of string str2 at the end of the string str1.

It returns the pointer to the resulting string str1.

STRCAT (String concatenate) function

```
#include <stdio.h>
#include <string.h>
int main() {
char str1[100], str2[100];
strcpy(str1, “Learning C ");
strcpy(str2, “Programing");
strcat(str1, str2);
printf("%s", str1);
return 0;
}
```

STRCAT (String concatenate) function

An undefined behaviour can be observed if size of str1 isn't long enough to accommodate the additional characters of str2.

```
#include <stdio.h>
#include <string.h>
int main() {
char str1[15], str2[100];
strcpy(str1, "Welcome to ");
strcpy(str2, "our Academy");
strcat(str1, str2);
printf("%s", str1);
}
```

STRNCAT (String concatenate) function

strncat is the safer version of strcat.It appends the limited number of characters specified by the third argument passed to it.

strncat automatically adds NULL character at the end of the resultant string.

Because we are trying to add more characters than the memory allocated.

STRNCAT (String concatenate) function

```
#include <stdio.h>
#include <string.h>
int main() {
char str1[5], str2[100];
strcpy(str1, "He");
strcpy(str2, "llo!");
strncat(str1, str2, sizeof(str1)-strlen(str1)-1);
printf("%s", str1);
return 0;
}
```