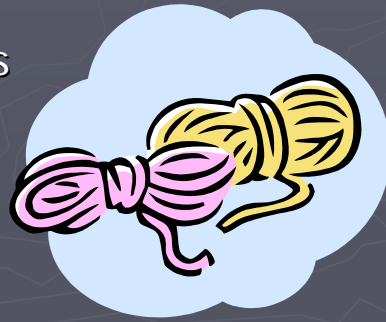


# Lecture 12

## Strings



We have already used strings

```
cout << "Hello Class" << endl;
```

"Hello Class" is a string.

# Strings

- ▶ String of characters
- ▶ can be represented by an array with base type of char with “\0” (null character) to end the string
- ▶ “abc” represented by

a	b	c	\0	
---	---	---	----	--

## declaring a string as an array

```
char s[10];
```

will declare an array capable of holding a character string of 9 characters plus null:

h	i		c	l	a	s	s	!	\0
---	---	--	---	---	---	---	---	---	----

Unlike an array holding characters, a c-string does not use a type int to keep track of how much of the array is used.

Instead, the null character marks the last character in the string.

Read the string starting from s[0] and end at the element “\0”

## Initializing a C-string

- To initialize a c-string you can do:

```
char my_message[size] = "Hello Class!"
```

- Note:

```
char my_message = "a";
```

is different than

```
char my_message = 'a';
```

- WARNING arrays are not a basic data type and do not work with the basic operators (`=`, `==`, `+`).

Although you can use the `"="` to initialize a c-string, you cannot use `"="` or `"=="` anywhere else with c-strings.

## String Functions

- Instead of `"="` use the function **strcpy**
- Instead of `"=="` use the function **strcmp**


```
strcpy(a_string, "Hello");
```

```
strcmp(a_string, other_string);
```

Pitfall: `strcpy` does not check to make sure that the string size is not exceeded!!!

## Safer String Functions

strncpy and strncmp work the same as strcpy and strcmp, but with an added argument for the size of the string.

 strncpy (my\_message, "Hello!", n);  
where n is the size of the string.

## String Functions

- |                              |                         |
|------------------------------|-------------------------|
| ▶ strcpy(target, source)     | copy a string           |
| ▶ strncpy(tar, src, limit)   | copy a string w/check   |
| ▶ strcat(target, source)     | concatenate two strings |
| ▶ strncat(tar, src, limit)   | concatenate w/check     |
| ▶ strcmp(string1, string2)   | compare two strings     |
| ▶ strncmp(str1, str2, limit) | compare w/check         |
| ▶ strlen(string)             | find length of a string |

## Input and Output with strings

```
char a[80], b[80];  
cout << "Enter some text:\n";  
cin >> a >> b;  
cout << a << b << "DONE" << endl;
```

Enter some input:  
Hello world it's me  
HelloworldDONE

## getline

To read an entire line, use the function getline:

```
char a[80], b[80];  
cout << "Enter some text:\n";  
cin.getline( a,80);  
cout << a << "DONE" << endl;
```

Enter some input:  
Hello world it's me Lynne  
Hello world it's me LynneDONE

## String to Number Conversions

- ▶ 1234 is not the same as "1234"
- ▶ The first is a number, the second is a character string. (See ASCII handout)

To use C-string to number conversion functions, you must include the following directive:

```
#include <cstdlib>
```

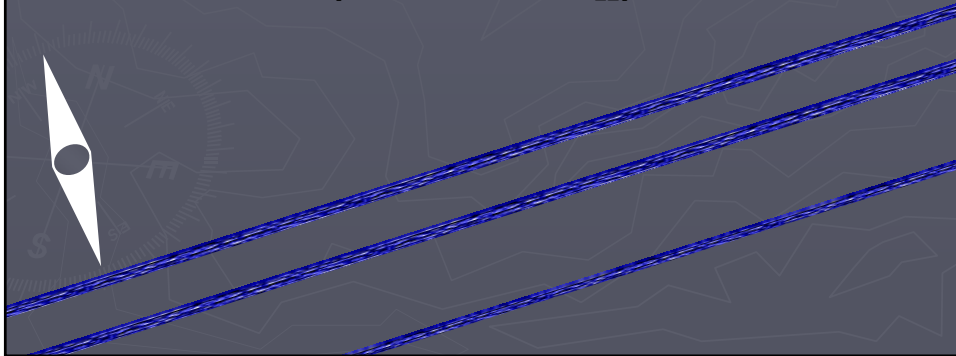
## Converting c-strings to numbers

The function `atoi()` will convert a string to an integer:

```
int num, char num_string[10];  
cin >> num_string;  
num = atoi(num_string);
```

## String Conversion Functions

- ▶ **int** `atoi(const char a[])`      *int*
- ▶ **long** `atol(const char a[])`      *long int*
- ▶ **double** `atof(const char a[])`      *double*

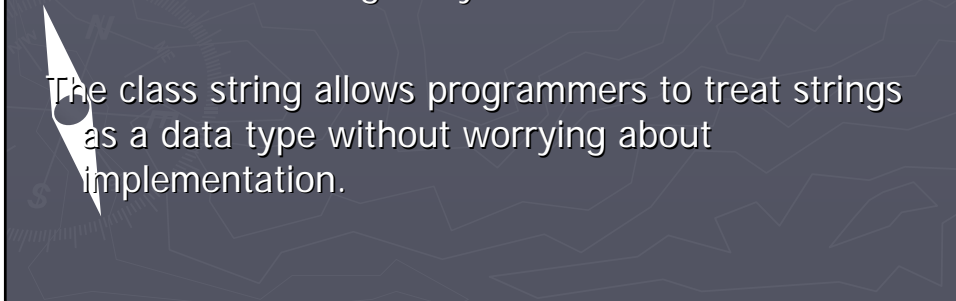


## New data types

The object oriented features of C++ allow new data types (classes) to be added to the language.

C-strings require programmers to keep track of details in handling arrays.

The class `string` allows programmers to treat strings as a data type without worrying about implementation.



## Standard Class String

- ▶ The class string is defined in the library `<string>`
- ▶ The string class allows you to treat strings very much like other data types
- ▶ You will learn about classes and objects next semester in CSII (or see section 11.2 in the book)

## Take a Break!





## 2-dimensional arrays

- Sometimes it is useful to have an array with more than one index (for example, an array of strings):

```
char page[30] [100];
```

```
page[0] [0], page [0] [1] ... page [0] [99]
```

```
page[1] [0], page [1] [1] ... page [1] [99]
```

```
page [99] [0], page [99] [1] ... page [99] [99]
```

```
char page[30] [100];
```

is declaring a one-dimensional array of size 30  
whose base type is a one-dimensional array of size  
100 (basically, an array of arrays).

Syntax:

```
type array_name [size_1] [size_2]...[size_last];
```

To hold 100 strings of 20 characters:  
`char names [100] [20];`

`cout << names [52] [5];`  
will display the 5<sup>th</sup> character in the 52<sup>nd</sup> string

`cout << names[52];`  
will display the entire 52<sup>nd</sup> string

## multi-dimensional arrays in functions

when a multi-dimensional array is used in a function heading or definition, the first dimension is not given:

`void get_page [] [100], int size_dimension1);`

The first dimension is the index of the array, the second is part of the description of the base type.

## Do not mix high and low level IO

```
▶ char ans;  
▶ char name[20];  
  
▶ do  
▶ {  
▶     cout << "Enter name: ";  
▶     cin.getline(name,20);  
▶     cout << "Name = " << name << endl;  
▶  
▶     cout << "Do again? ";  
▶     cin >> ans;  
▶ }  
▶ while (ans == 'y');
```



## Reading

[www.graphicsmagician.com](http://www.graphicsmagician.com)

