Characters, Strings, Arrays, Files

Characters

- *char* is a type that holds a <u>single</u> character.
- It is stored numerically as its ASCII code.
- for example: *char letter = 'A';* holds the value 65.
- You can compare characters and do arithmetic operations on them.
    - For example:
        - if(letter < 'B') // true.
        - letter += 1; // letter is now equal to 66 or 'B'
- Character are denoted with single quotes. Double quotes are for strings.
    - 'A' ≠ "A"
    - Note: many other languages do not handle quotes this way, this is very "C thing," as are most C rules for strings and characters.

Arrays

- Arrays store groups of values of the same type.
- int values[ ]; //this is an uninitialized array of size 0.
- int values[5]; //this is an uninitialized array of size 5. It will hold 5 integers
- int values[5] = {23, 4, 12, 68, 1}; //initialized array of size 5
- Array elements are accessed by specifying the index starting with 0
  - values[0] == 23
  - values[1] == 4
  - values[4] == 1
- Variables can be used to access arrays
  - int i = 3;
  - values[i] == 68

Strings and Character Arrays

- C/C++ does not have a built-in native data type for Strings (like most languages).
- Strings are created in one of two ways:
  - C style: Strings are arrays of characters with a null terminator.
  - C++ style: String objects can be created using the String class.
- *char name[15];* // creates an empty character array of size 15. Note: this is **NOT** a string yet. It is only a character array.
- *char name[15] = "Hello World";* //creates a character array of size 15 and initializes is with a string of size 11 ← Note: takes up 24 bytes, not 22. Extra 2 bytes for '\0'.
- Character arrays are strings ***<u>if and only if</u>*** they contain a null terminator '\0'
- *name = "new string";* // WILL NOT WORK IN C (will work in C++ if you use the string object instead of char)
- Character arrays used for strings generally require string functions. ***Strings are not a native data type*.**
  - Example: *strcpy(name, "new string");* // this assigns a string to name in C

Strings - C++ Class

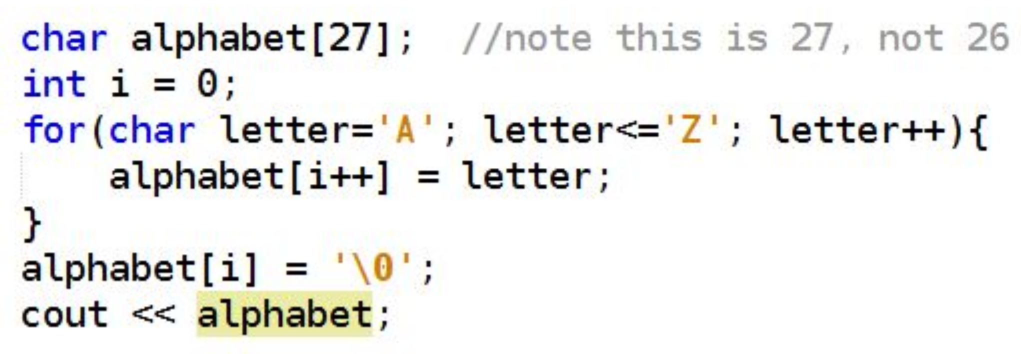
- The C++ string class allows strings to function similar to native data types.
- Must include the string library #include<string>;
- *string name;* // makes an empty string object
- name = “bob”; // assigns “bob” to name
- cout << name; // outputs *bob*
- Remember, the *string* class is not a native data type. It allows you to create a string object which has many extended capabilities that make strings work in C++.

Operations on Strings

- String Copy
  - C style: ***strcpy(str, "string");*** // must #include<string.h>
  - C++ style: ***str1 = str2;*** // must #include<string>
- String Length
  - C style: int size = strlen(str); // uses the C library function, #include<string.h>
  - C++ style: int size = str.size();; // uses the C++ string member function, #include<string>
- String conversion
  - *int number = stoi(str);* // takes a string with digit characters and converts it to an integer
- C and C++ allow you to do the same thing many different ways.
- There are many string functions (C) and string member functions (C++)
- For example: https://www.geeksforgeeks.org/5-different-methods-find-length-string-c/

Operations on Character Arrays

- Remember, a character array without a null terminator is not a string.
- A string consisting of only “\0” is still a string, it is a string of size 0.
- Loops are a very common way of interacting with character arrays.



```
char alphabet[27];  //note this is 27, not 26
int i = 0;
for(char letter='A'; letter<='Z'; letter++){
    alphabet[i++] = letter;
}
alphabet[i] = '\0';
cout << alphabet;
```

Strings Gotchas

- Character arrays can be overrun! (like all arrays in C)
  - If you attempt to enter a character past the end of the array, it will “work.”
  - You will cause a memory corruption.
- C character arrays DO NOT need a null terminator ***IF AND ONLY IF*** you use them as character arrays. But be careful, you cannot use any string functions without the ‘\0’.
- C character arrays DO need a null terminator if you use them as strings.
- Don’t forget to allow space for the null terminator. String size is always +1 in memory.
- White space can cause string operations like cin to operate unexpectedly.

Inputting Strings and Characters

- cin does not work well with strings or characters. cin uses white space as a delineator.
  - cin >> name; //assume name is a string, input “bob” and it works. Input “Bob Smith” and only “bob” goes into name, and “Smith” will still be in the input buffer.
  - cin >> letter; //assume letter is a char, input space or a string and it doesn’t work.
- getline is a C++ function that is better for inputting strings.

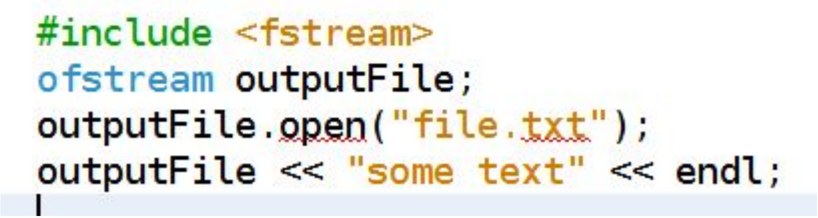
  string name; // create a C++ string object

  getline(cin, name); // use getline and pass it the cin and string objects

- cin.get is a member function of cin that allows you to get a single character including whitespace. Note: You can use *cin.get();* alone to pause a program for input (e.g. “press any key to continue”)

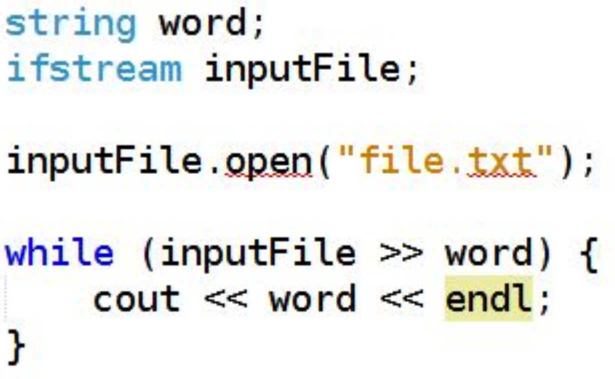
File Access Methods

- Writing to a file



```
#include <fstream>
ofstream outputFile;
outputFile.open("file.txt");
outputFile << "some text" << endl;
```

- Reading from a file



```
string word;
ifstream inputFile;

inputFile.open("file.txt");

while (inputFile >> word) {
    cout << word << endl;
}
```

File Access Methods (continued)

- inputFile.close( ); // Don’t forget to close your files when done!
- Read Position
  - When a file is opened it’s read position is set to 0 (the start of a file)
  - Each “read” from a file takes in all characters up until the next white space.
  - The read position is then placed at the start of the next “word.”
  - White space is “thrown away.”
- Detecting EOF
  - There are several ways, but at this point the easiest is with the >> operator.
  - See previous example.
- Think of a file as a character array.
  - Each byte is a place that the read position can point.
  - Each “word” is all the bytes between white space.
- Testing for valid file
  - Simplest way is to test the object *after* you attempt to open it; if(inputFile) { //valid }