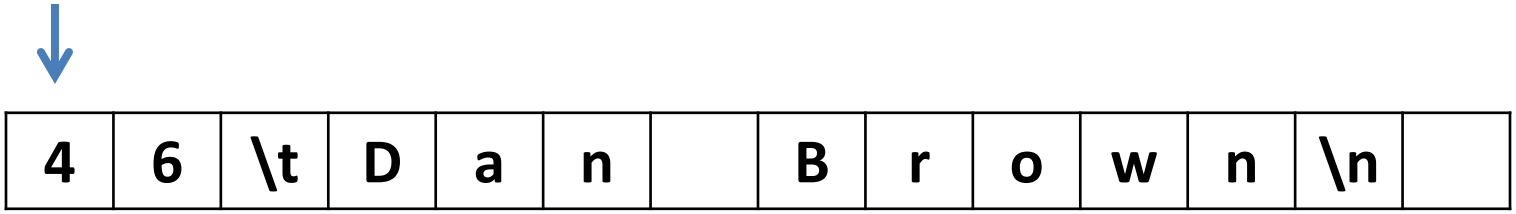
“Stream” input, strings, cstrings, arrays

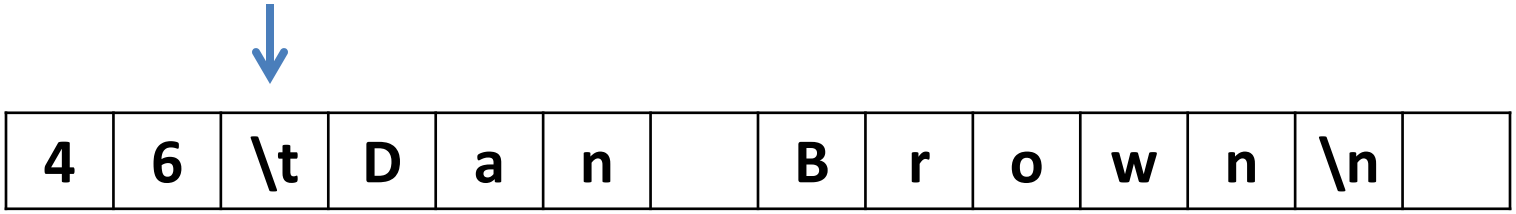


Recall that a “stream” is a type of C++ “object” – i.e., a construct that has member ***data*** and member ***functions***. Think of it as a sequence of characters that will flow to your program code in response to functions that you call.

In response to the above stream, with the arrow identifying the first character in the stream, our goal is to achieve:

intVar = 46;
stringVar = “Dan Brown”;

“Stream” input, strings, cstrings, arrays

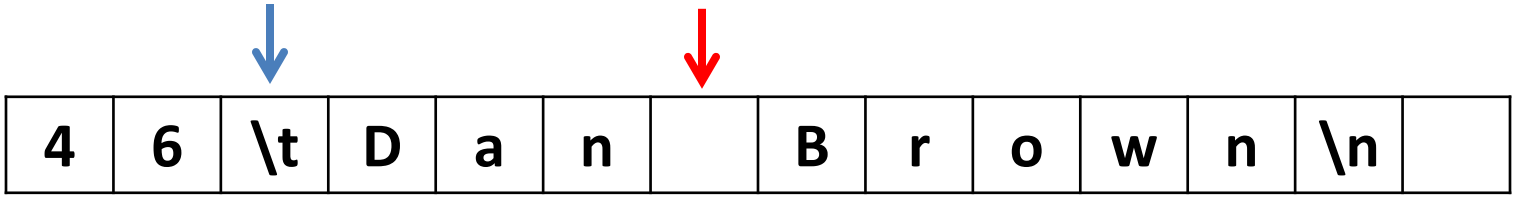


Tool: cin >> intVar;

Effect: As long as “digit” symbols emerge from the stream, they’re incorporated into “int” variable. New head of the stream (where arrow now points) is first non-digit symbol encountered. If UNABLE to produce an “int”, then (cin >> intVar) would evaluate to FALSE (*bottom p. 335*).

intVar = 46;

“Stream” input, strings, cstrings, arrays

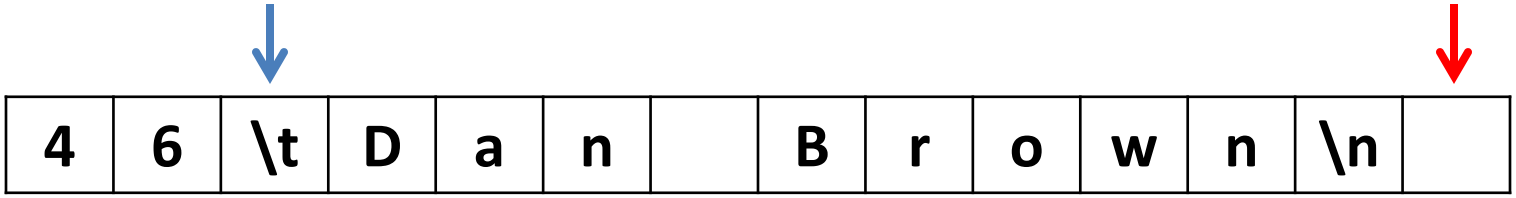


Tool: cin >> stringVar;

Effect: First, discards any leading “white space” characters (such as tab, space). Once non-white-space is encountered, builds string until NEXT white space is encountered. New head of the stream at red arrow. (*p. 475*)

stringVar = “Dan”;

“Stream” input, strings, cstrings, arrays

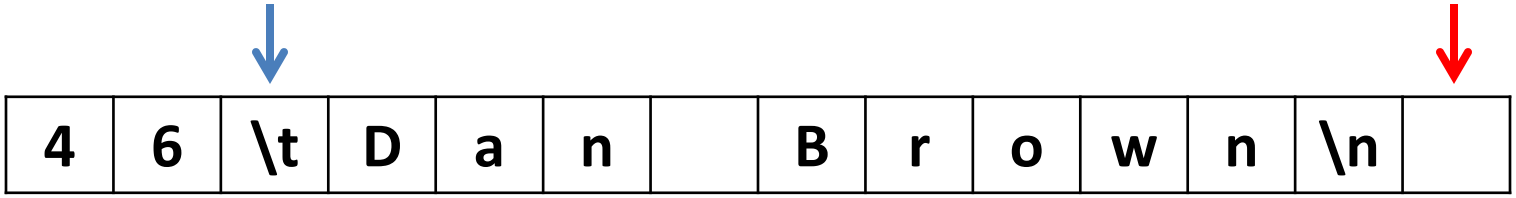


Tool: getline(cin, stringVar);

Effect: Builds string composed of ALL symbols that emerge, until a ‘\n’ is encountered. ‘\n’ is consumed, but left out of the string. New head of the stream at red arrow. (*p. 479*)

stringVar = “\tDan Brown”;

“Stream” input, strings, cstrings, arrays

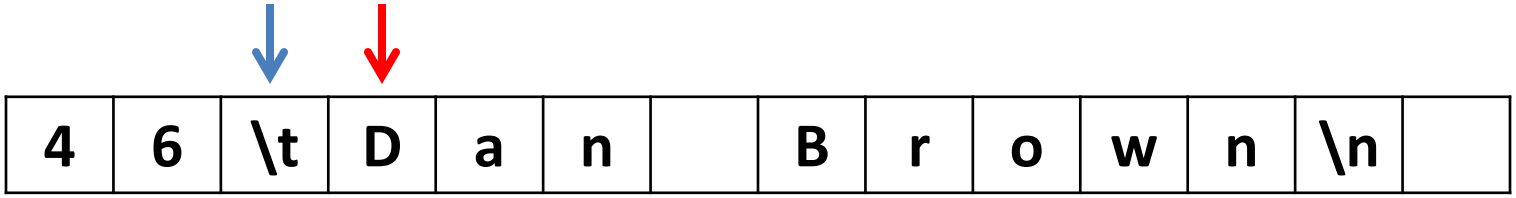


Tool: getline(cin, stringVar, ‘\n’);

Effect: Builds string composed of ALL symbols that emerge, until a ‘\n’ is encountered. ‘\n’ is consumed, but left out of the string. New head of the stream at red arrow.

stringVar = “\tDan Brown”;

“Stream” input, strings, cstrings, arrays



Tool: getline(cin, stringVar, ‘\t’);

Effect: Builds string composed of ALL symbols that emerge, until a ‘\t’ is encountered. ‘\t’ is consumed, but left out of the string. New head of the stream at red arrow.

stringVar = “”;

“Stream” input, strings, cstrings, arrays

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]				
\t	**D**	**a**	**n**		**B**	**r**	**o**	**w**	**n**				

If we had used getline(cin, stringVar), above is the content of stringVar, viewed as an array of chars. (*Unlike old cstrings, the new C++ version of a “string” does NOT include a ‘\0’ null terminating byte*)

Recall that bracket[index] access is available to work with individual chars in a string. E.g., the letter ‘a’ is stringVar[2], stringVar.size() returns 10 (the # of chars in the string). *See p. 483*

DISPLAY 8.7 Member Functions of the Standard Class `string` *(part 2 of 2)*

Accessors	
`str[i]`	Returns read/write reference to character in `str` at index `i`. Does not check for illegal index.
`str.at(i)`	Returns read/write reference to character in str at index `i`. Same as `str[i]`, but this version checks for illegal index.
`str.substr(position, length)`	Returns the substring of the calling object starting at `position` and having `length` characters.
`str.length( )`	Returns the length of `str`.
Assignment/Modifiers	
`str1 = str2;`	Initializes `str1` to `str2's` data.
`str1 += str2;`	Character data of `str2` is concatenated to the end of `str1`.
`str.empty( )`	Returns true if `str` is an empty string; false otherwise.
`str1 + str2`	Returns a string that has `str2's` data concatenated to the end of `str1's` data.
`str.insert(pos, str2);`	Inserts `str2` into `str` beginning at position `pos`.
`str.erase(pos, length);`	Removes substring of size `length`, starting at position `pos`.
Comparison	
`str1 == str2` `str1 != str2`	Compare for equality or inequality; returns a Boolean value.
`str1 < str2` `str1 > str2`	Four comparisons. All are lexicographical comparisons.
`str1 <= str2` `str1 >= str2`	
Finds	
`str.find(str1)`	Returns index of the first occurrence of `str1` in `str`. If `str1` is not found, then the special value `string::npos` is returned.
`str.find(str1, pos)`	Returns index of the first occurrence of string `str1` in `str`; the search starts at position `pos`.
`str.find_first_of(str1, pos)`	Returns the index of the first instance in `str` of any character in `str1`, starting the search at position `pos`.
`str.find_first_not_of (str1, pos)`	Returns the index of the first instance in `str` of any character not in `str1`, starting the search at position `pos`.

Mission: parse user input using strings/arrays

Declare 3 variables: age (int), lastN (string) and firstN (string)

Write a loop within main() that prompts the user to input an age, first name, and last name on a single line, with tab characters '\t' as the delimiters separating the three fields.

Within the loop, call a function named rdPerson() that uses call-by-reference to access the 3 variables.

Within the loop, after rdPerson() returns, call a second function named makeUC() that converts alphabet chars within a string to upper case.

Within the loop, after the UC activity, display the 3 variables

Then loop and prompt for NEXT input

Mission: parse user input using strings/arrays

Write the rdPerson() function. It should read the user's console input and dissect it, producing values for the 3 variables.

Write the makeUC() function. It should treat a string as an array, working through it and converting chars

Here are a couple of input samples. Note that characters within names might be NON-alpha, and that a name might consist of more than one word.

```
39      Ralph   Nader
57      Kathy   Yates-Stevenson
108     John    von Neumann
54      Henry   d'Artagnan
```