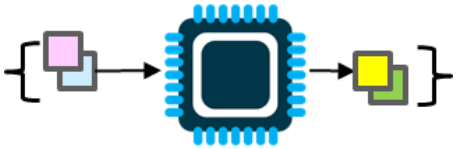
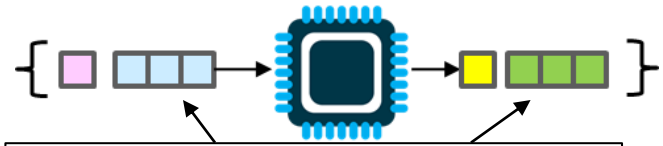
Unit 2d – Strings

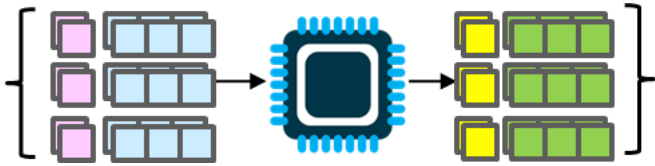
Mark Redekopp

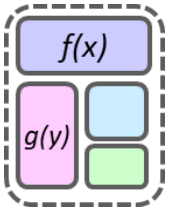
Unit 2

- **Unit 1**: Scalar processing
  - aka IPO=Input-Process-Output Programs
- **Unit 2**: Linear (1D) Processing
- **Unit 3**: Multidimensional Processing
- **Unit 4**: Divide & Conquer (Functional Decomposition)





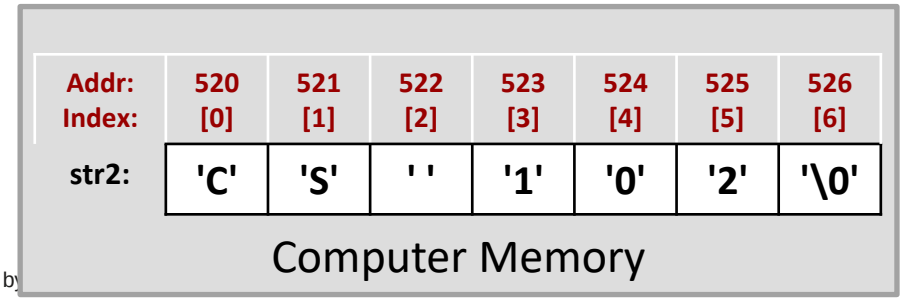




Character Arrays and Strings (1)

- Recall that in C/C++ string constants (the text in between " ") are just character arrays
  - Each character consumes 1 element in the array
  - Ends with the null character (e.g. 0 decimal or '\0' ASCII)
- This approach of using an array of `char`'s to store a string is referred to as a **C-String** because there was no `string` type in C (i.e. before C++)

```
#include <string>
using namespace std;
int main()
{
  char str1[3] = {'C', 'S', '\0'};
  // For char arrays easier to use ""
  char str2[7] = "CS 102"
  /* Initializes the array to "CS 102"*/

  cout << str1 << endl;    // prints "CS"
  cout << str2 << endl;    // prints "CS 102"

  str2[5] = '3';
  cout << str2 << endl;    // prints "CS 103"

  cin >> str2; // get a new string from
               // the user (suppose user
               // types "hello"
  cout << str2;
}
```

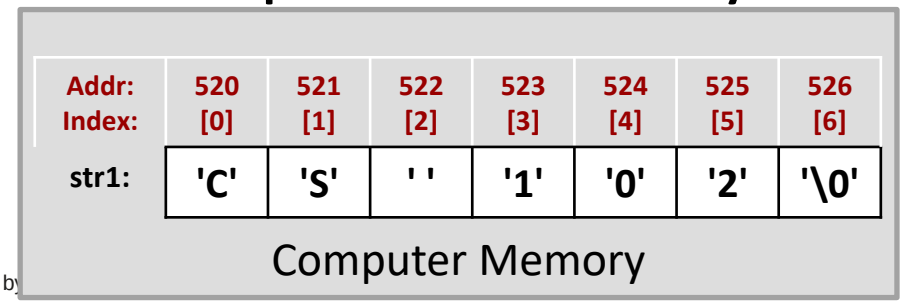


Program Output:

```
CS
CS 102
CS 103
hello
```

Character Arrays and Loops

- How many things can a computer do at a time?
- To printout a string/character array, we'd have to print one character at a time!
- But C/C++ treats character arrays specially. `cout` has a loop inside its code to print strings/character arrays.
- Though not shown, `cin` also has a loop inside to input a string.
- We say `cout` and `cin` have a **special relationship with character arrays.**



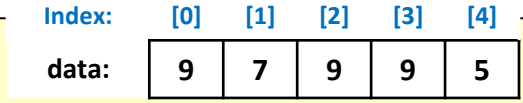

```
#include <string>
using namespace std;
int main()
{
  char str1[7] = "CS 102"
  /* Initializes the array to "CS 102"*/

  // Usually in C/C++ we must use a loop to do
  // many operations
  for(int i=0; str[i] != '\0'; i++) {
    cout << str[i];
  }
  cout << endl;

  // but cout has its own loop so you don't
  // have to write the loop above but just
  // what you see below.
  cout << str1 << endl;    // prints "CS 102"
}
```

Program Output:

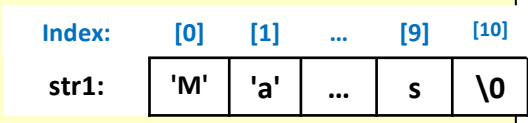
```
CS 102
CS 102
```

cout's Special Relationship with Character Arrays

- To print out all elements of **any array type *OTHER than a character array*** (i.e. `int`, `double`, `bool`, etc.) you must **write your OWN loop** (i.e. because computers can only do 1 thing at a time)
- But for **character arrays**, you can just give `cout` the name of the array and it will use its **own INTERNAL loop** to print out all characters for you
  - So, internally it is actually looping over the characters so you don't have to
  - It just assumes when you give it a character array that you WANT it to print out all the characters in the array
- Thus, we say **`cout`** treats character arrays specially




```
int main()
{
  int data[5] = {9, 7, 8, 9, 5};
  char str1[] = "Many chars";
  // right way to print int array contents
  for(int i=0; i < 5; i++){
    cout << data[i] << " ";
  }
  cout << endl;

  // doesn't work for an int, double
  // or any other type of array
  cout << data << endl;

  // cout treats char. arrays specially
  cout << str1 << endl;
}
```



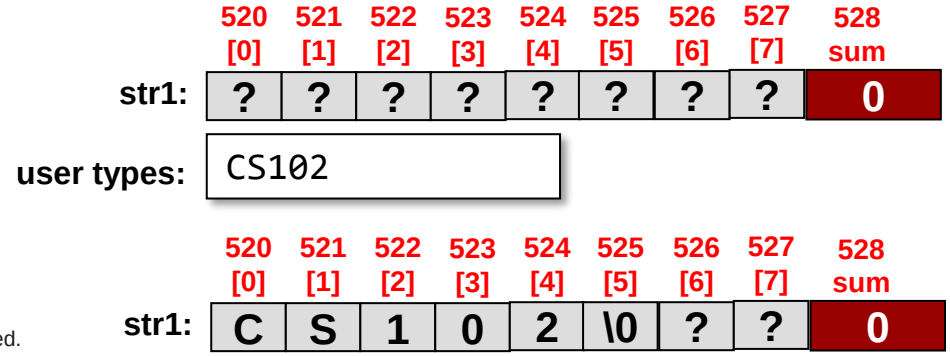
Program Output:

```
9 7 8 9 5
Many chars
0x7fffce40
```

cin's Special Relationship with Character Arrays

- To get input for all elements of an array type *OTHER than character arrays* (i.e. `int`, `double`, etc.) you must **write your OWN loop**
- But for character arrays, you can just give `cin` the name of the array and it will use its **own INTERNAL loop** to receive all characters the user types and store them sequentially in the array
  - So, internally it is actually looping over the characters so you don't have to
  - It just assumes when you give it a character array that you WANT it to get a full string (stopping at the next space)
- **`cin`** treats character arrays specially

```
int main()
{
  int data[5]; //5 garbage values to start
  char str1[8];//8 garbage values to start
  int sum = 0;
  // doesn't work for an int, double
  // or any other type of array
  cin >> data;  // won't even compile

  // right way to get int array contents
  for(int i=0; i < 5; i++){
    cin >> data[i];
  }

  // cin treats char. arrays specially
  cin >> str1;
}
```

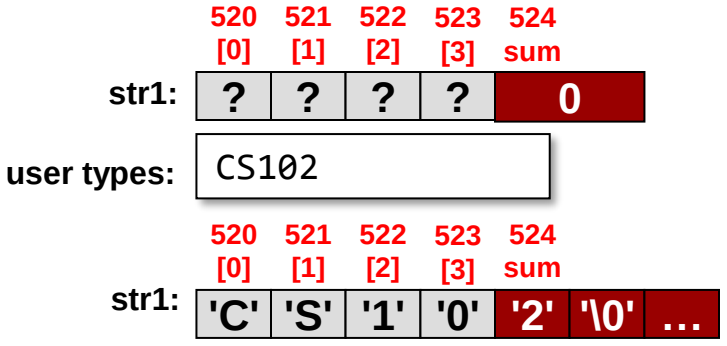


A Problem with `cin` and Character Arrays

- What if the user types in **TOO** much (*more characters than our array has room to store*)?
- `cin` will not stop! It will keep storing the characters the user types, overwriting whatever data and variables came after the array
- **Warning**: `cin` does not CHECK that the string typed by the user will fit in the array; instead it simply **overwrites memory leading to undefined (bad) behavior!**

- **C++ strings fix this issue, allocating more space based on what is typed.**

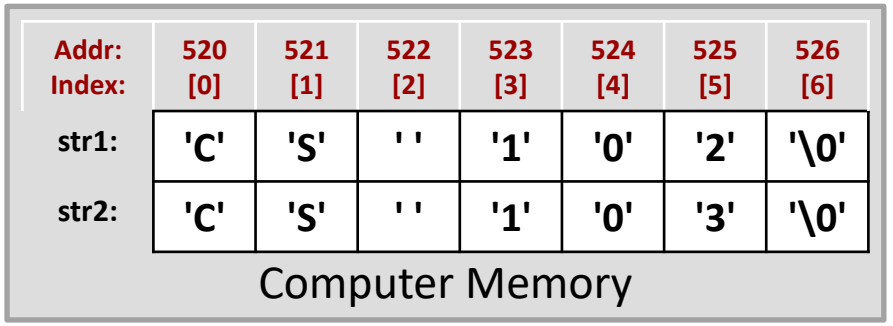
```
int main()
{
  char str1[4];
  int sum = 0;
  // What if user types in "CS102"
  cin >> str1;

  cout << sum << endl;
  // won't see 0 because sum was modified
  // when cin received the string that was
  // too long!

  string s2;
  cin >> s2;
  // works regardless of user input length
}
```



What About Other Operations

- How would you check whether two strings (character arrays) are equal (i.e. have the same character sequence).
- Since we can only do 1 thing at a time, we'd have to use a loop
- Does '==' have a special relationship with character arrays? NO!!!
  - Most operations on strings require a loop since we can only do 1 thing at a time.
  - cin and cout are exceptions. Every other operation requires the programmer to write a loop!
- So when C++ came along they said, let's fix this. Let's provide code to deal with strings. Enter the C++ `string` type

```
#include <string>
using namespace std;
int main()
{
  char str1[7] = "CS 102"
  /* Initializes the array to "CS 102"*/
  char str2[7] = "CS 103";

  if(str1 == str2) { … } // Doesn't work

  // Instead you'd need some kind of loop
  bool same = true;
  for(int i=0; /* some condition */; i++) {
    if(str1[i] != str2[i]) {
       same = false;
    }
  }
  cout << endl;
  return 0;
}
```



C-Strings != C++ strings

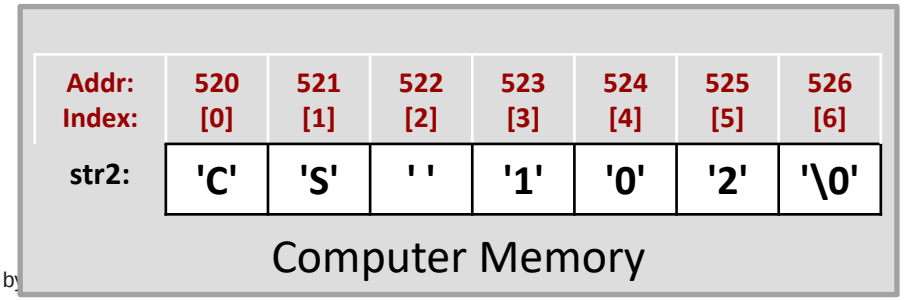
	C-String	C++ String
As constants **(same)**	`"hi"`	`"hi"`
As variables **(different)**	`char str1[3] = "hi";` `char str2[4] = "bye";`	`string str1 = "hi";` `string str2 = "bye";`
To use:	No special #include	`#include <string>`
Works with cout	Yes!	Yes!
Works with cin	Yes, but potentially dangerous	Yes!
Other ops	None	Reassignment Comparison (==, <, >, etc.) Substrings

Why are strings messy ? Because they are *variable* length, where as other variable types are a fixed size! Any int can fit in the memory of another int variable. But for strings what if we want to store a new, longer string in the memory of a shorter string? We don't have room?

Character Arrays and Strings (2)

- C++ strings can do all that character arrays can do

```
int main()
{
  char str2[7] = "CS 102"
  string str3 = "CS 102";

  cout << str2 << endl;    // prints "CS 102"
  cout << str3 << endl;    // prints "CS 102"

  str2[5] = '3';
  str3[5] = '3';
  cout << str2 << endl;    // prints "CS 103"
  cout << str3 << endl;    // prints "CS 103"
  cin >> str2; // get a new string from
               // the user (suppose user
               // types "hello"
  cin >> str3;
  cout << str2;
  cout << str3;
  return 0;
}
```



Program Output:

```
CS 102
CS 102
CS 103
CS 103
hello
hello
```

C++ Strings

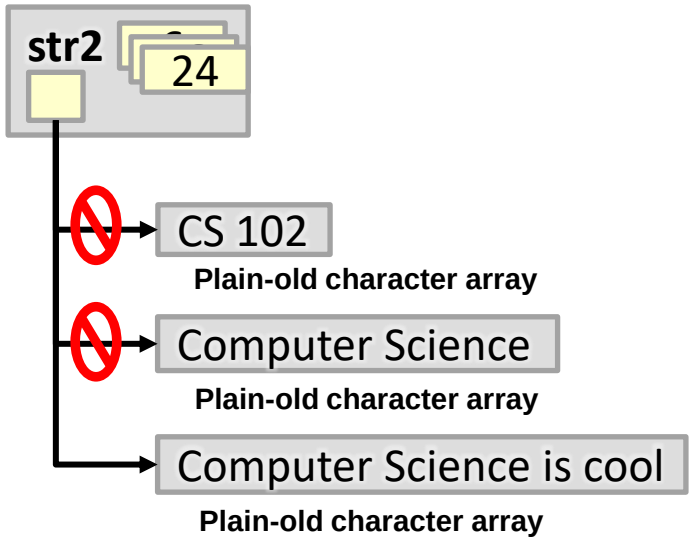
- In C++, the library adds a new object type named **string (C++)** and provides an easier alternative to working with plain-old **character arrays (C-language)**
- Do's and Don'ts
  - **Do** `#include <string>`
  - **Don't** need to declare the size (i.e. `[7]`), just assign
  - **Do** still use it like an array by using `[index]` to get individual characters
  - **Do** still use `cin/cout` with strings
  - **Don't** worry about how many characters the user types when inputting to a C++ string

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  char str1[7] = "CS 102";
  /* Initializes the array to "CS 102"*/
  string str2 = "CS 102";
  /* Initializes str2 to "CS 102"*/

  str1[5] = '3'; // now str1 = "CS 103"
  str2[5] = '4'; // now str2 = "CS 104"

  cout << str1 << endl;
                 // prints "CS 103"
  cout << str2 << endl;
                 // prints "CS 104"

  cin >> str1; // If the user types more
                 // than 6 chars..uh oh!
  cin >> str2; // str2 will adjust to
                 // hold whatever the user
                 // types
}
```

What Do Strings Do

- Strings simply abstract character arrays
- Behind the scenes strings are just creating and manipulating character arrays but giving you a simplified set of operators and functions
- Can concatenate (append) to a string with the + operator

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string str2 = "CS 102";
  // str2 stores 6 chars. = "CS 102"

  str2 = "Computer Science";
  // now str2 stores 16 characters

  // Can append using '+' or '+=' operator
  str2 = str2 + " is cool";
  // now str2 stores 24 characters
}
```



String Size

- Strings track how many characters they are storing
- Call the `<stringname>.size()` function get the string's size
  - Returns the actual number of real characters (and does not count overhead like the null character)

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string str2 = "CS 102";
  cout << str2.size() << endl; // 6

  str2 = "Computer Science";
  cout << str2.size() << endl; // 16

  str2 = str2 + " is cool";
  cout << str2.size() << endl; // 24
}
```

String Comparison

- Comparison operators **do not work** with plain old character arrays (C-Strings)
- C++ strings **do** perform lexicographic (alphabetical/dictionary-order) comparison when comparison operators (<, >, ==, etc.) are applied
  - "a" < "z" ? _____
  - "a" > "aa" ? _____
  - "ab" < "ba" ? _____
  - "aab" < "aac" ? _____

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  char str1[4] = "abc";
  string str2 = "abc";

  if( str1 == "abc" ) // doesn't work
    {...}
  if( str2 == "abc" ) // works..true
    {...}

  if( str1 < "aac" )  // doesn't work
    {...}
  if( str2 < "aac" )  // works..false
    {...}

  string str3 = "acb";

  if( str3 > str2 )   // works..true
    {...}
}
```

Substrings

- C++ strings allow you to produce a new string from a **substring of a current string**
- Call either of the 2 versions:
  `.substr(start_index)` **or**
  `.substr(start_index, length)` function on the string
  - 1st version generates substring from starting index location all the way to the end of the string
  - 2nd version generates substring from the starting index and includes the next 'length' characters
  - Note: when a function has the same name but different options for parameters we say the function is overloaded
- Returns a new string
  - Even if length is 1 (i.e. if length is 1 you might think you just get a char, but you still get a string)

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string str1 = "CS102";

  string str2 = str1.substr(2);
     // str2 = "102"

  str1 = "Hello World";
  str2 = str1.substr(6,2);
     // str2 = "Wo"

  str2 = str1.substr(0,1);
     // str2 = "H"

}
```

SOLUTIONS

String Comparison

- Comparison operators **do not work** with plain old character arrays (C-Strings)
- C++ strings **do** perform lexicographic (alphabetical/dictionary-order) comparison when comparison operators (<, >, ==, etc.) are applied
  - "a" < "z" ? TRUE
  - "a" > "aa" ? FALSE
  - "ab" < "ba" ? TRUE
  - "aab" < "aac" ? TRUE

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  char str1[4] = "abc";
  string str2 = "abc";

  if( str1 == "abc" ) // doesn't work
    {...}
  if( str2 == "abc" ) // works..true
    {...}

  if( str1 < "aac" )  // doesn't work
    {...}
  if( str2 < "aac" )  // works..false
    {...}

  string str3 = "acb";

  if( str3 > str2 )   // works..true
    {...}
}
```