CS 201
String

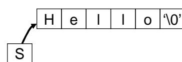
Debzani Deb

Characters

- Characters are small integers (0-255)
- Character constants are integers that represent corresponding characters.
  - '0' – 48
  - 'A' – 65
  - '\0' (NULL) – 0
  - '\t' (TAB) – 9
  - '\n' (newline) – 10
  - SPACE – 32
- ASCII code maps integers to characters.

Strings(1)

- **String –** a group of characters
- C implements the string data structure using arrays of type char.
- Two things to remember
  1. All strings have a "hidden" extra character at the end – '\0', the null character.
     - "Hello" is a string of length 6
  2. Strings are managed by pointers
     - `char *s = "Hello";`
     - Creates a pointer s such that



Strings(2)

- Two interpretations of String
  - Arrays whose elements are characters.
  - Pointer pointing to characters.
- Strings are always terminated with a NULL characters('\0'). C needs this to be present in every string so it knows where the string ends.

```
char a[] = "hello\n";
char* b = "hello\n";
```

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
*b	*(b+1)	*(b+2)	*(b+3)	*(b+4)	*(b+5)	*(b+6)
h	e	l	l	o	\n	null
104	101	108	108	111	10	0

String Constants

- We have already used **string constants** extensively in our earlier work:

  `printf("Hello There");`

  Hello There is a string constant
- In C, string constants are identified by surrounding **"**
  - Note that this is different from characters – they use single quotes – **'**
  - Spaces are just treated like any other character
  - A null character is automatically inserted after the constant string.
- You can also use #define directive to associate a symbolic name with a string constant.
  - `#define ERR "***ERROR***"`

Initializing and Declaring String

```
char s[] = "hello";
char* s_ptr = "hello";
char s[] = {'h', 'e', 'l', 'l', 'o', '\0'};
```

- We can leave out the dimension of the array, the compiler can compute it automatically based on the size of the string (including the terminating \0).
- String Declaration

  `char s[30];` // declared as an array of 30 char

  `char *s;` // declared as a pointer to char
  - Note that the array s actually holds 30 characters, but the string can only contain up to 29. Because the 30th character would be '\0'.

Input/Output with printf and scanf

- Both printf and scanf can handle string arguments as long as the placeholder %s is used in the format string:

```
char s[] = "hello";
printf("%s\n", s);
```

- The printf function, like other standard library functions that take string arguments, depends on finding a null character in the character array to mark the end of the string.
- If printf were passed a character array that contained no '\0', the function would continue to display as characters the content of memory locations following the array argument until it encountered a null character or until it attempted to access a memory cell that was not assigned to the program, causing a **run-time error**.

Input/Output Cont.

- The approach scanf takes to string input is very similar to its processing of numeric input.
  - When it scans a string, scanf skips leading whitespace characters such as blanks, newlines, and tabs.
  - Starting with the first non-whitespace character, scanf copies the characters it encounters into successive memory cells of its character array argument.
  - When it comes across a whitespace character, scanning stops, and scanf **places the null character** at the end of the string in its array argument.

```
char s[10];
scanf("%s", s);    // bad practice
scanf("%9s", s);   // Good practice, prevents overflowing s
```

  - Notice that there is no &, this is because strings are arrays, and arrays are already pointers.

String Library Functions

- C provides functions to work with strings, these are located in the string.h file.
- Put #include <string.h> at the top of the program to use these.
- Look at Table 9.1 on page 441 to see what functions are provided.
- Note that all of these function expect the strings to be **null terminated**.

String Copy (1)

- We typically use the assignment operator = to copy data into a variable.

```
char c, s[10];
c = 'a';
s = "Hello";  // Does not work, Why?
```

- Exception: we can use the assignment symbol in a declaration of a string variable with initialization.

```
char s[] = "hello";
```

- In all other cases, you must do

```
s[0] = 'H'
s[1] = 'e'
s[2] = 'l' // etc
```

- Or use string copy function.

String copy (2)

- Function strcpy copies the string that is its second argument into its first argument.

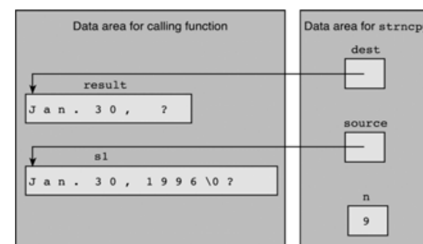
  char *strcpy(char *dest, const char *source)

  Example: strcpy(s, "Test String");
- Overflowing is possible if destination is not long enough to hold source.
- Function strncpy copies upto n characters of the source string to the destination.

  strncpy(s, "Test String", 4); //copies 4 characters of "Test String"
- If the first n characters do not contain the null character, then it is not copied into variable s and you must do a s[n] = '\0';
- Good Practice:

```
strncpy(dest, source, dest_len -1);
dest[dest_len-1] = '\0';
```


Figure 9.5 Execution of strncpy



char *strcpy(char *dest, const char *source)

Both strcpy and strncpy return pointer to the old value of destination.

String Length

int strlen(char *s)

- Counts the number of characters before the null terminator.
- If the null terminator is the first character (empty string), it returns 0.

Substring Extraction

- We frequently need to reference a substring of a longer string.
- For example we might want to examine the "Cl" in the string "NaCl".
- We can use strncpy to copy a middle or an ending portion of a string.
  strncpy(dest, &source[2], 2);
- This goes to the 3rd spot in the array source, gets the memory location, and then copies 2 elements starting at this new memory location.
- Function strstr finds the first occurrence of one string inside another.
  char* strstr(char *s_in, char *s_for)
  - Returns a pointer to the first location in s_in where s_for appears. Returns NULL if not found.
  - Case sensitive.

String Comparison

- Will string_1 < string_2 work? Wrong ! they are only pointers

int strcmp(char *s1, char *s2)
  - Compares two strings lexicographically (based on ASCII code).
  - Returns 0 if they are same
  - Returns negative if s1< s2
  - Returns positive if s1> s2

int strncmp(char *s1, char *s2, int n)
  - Compares two strings upto n characters.

- How to compare and swap two strings:

```
if (strcmp(s1, s2) > 0){
    strcpy(tmp, s1);
    strcpy(s1, s2);
    strcpy(s2, tmp);
}
```


String Concatenation

- Concatenation is taking two strings and then make them become one string.
  char* strcat(char *dest, char *src)
  - Puts src at the end of dest, including src's null terminator.
  - Puts the first character of src in place of dest's null terminator.
  - Reurns pointer to the old value of dest.
- strncat does the same as strcat, but it only copies n characters from the second string
  - strncat(s1, s2,3); would copy only 3 character from s2.
  - Puts a null terminator at the end of dest when it is done copying.
- In both cases, overflowing is possible and as a result we may overwrite other variables or get a run-time error.

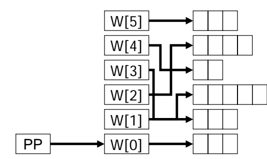
Array of Pointers (1)

- Arrays can be of any type, including pointers.
  char *suit[4] = {"Hearts", "Diamonds", "Clubs", "Spade"};
- Each element of array suit is an address of a character string and there are 4 such elements in it.
- Suit[0][0] refers to 'H'
- Suit[3][4] refers to 'e' in "Spade"
- Suit[2][20] is an illegal access.
- Suit[0] refers to the address of "Hearts"

Array of Pointers (2)

- Arrays of pointers are more flexible and faster to move around than arrays of data.
- More efficient use of memory
- No need to allocate memory in advance
- Applications
  - Alphabetical sorting of words
  - Memory management by operating system.

- Example:
  int *W[6];
  int **PP = W;



W and PP are pointer to pointer to int

W[i] and *PP are pointer to int

*W[i] and **PP are int

Ragged Arrays (1)

- An array of pointers whose elements point to arrays of different sizes is called a ragged array.
- A group of words may be stored as
  - A two dimensional array of type char (array of strings), or
  - A one dimensional array of pointers to char.

  Example:

```
#include <stdio.h>
int main(void){
   char a[2][9] = {"abc:", "an apple"};
   char *p[2] = {"abc:", "an apple"};
   printf("%c%c%c %s %s\n", a[0][0], a[0][1], a[0][2], a[0], a[1]);
   printf("%c%c%c %s %s\n", p[0][0], p[0][1], p[0][2], p[0], p[1]);
   return 0;
}
```


Ragged Arrays (2)

`char a[2][9] = {"abc:", "an apple"};`

a is a 2D array, and its declaration allocates space for 2x9 = 18 chars, only the first five elements 'a', 'b', 'c', ':' and'\0' are used in a[0]; the rest are initialized to zero.

Note that a[0][8] is a valid Reference, but p[0][8] is not

a	b	c	:	\0				
a	n		a	p	p	l	e	\0

`char *p[2] = {"abc:", "an apple"};`

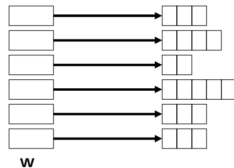
p is a 1D array of pointers to char, When it is declared, space for two pointers are allocated. p[0] is initialized to point at "abc:", a string requiring space for 5 chars. p[1] is initialized to point at "an apple" which requires space for 9 chars. Thus p does its job in less space than a; it is also faster, as no mapping function is generated

→ a | b | c | : | \0

→ a | n | | a | p | p | l | e | \0

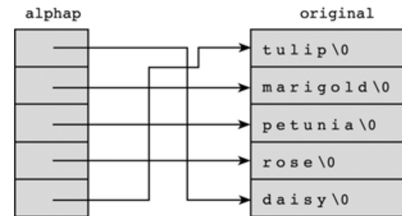
Example: sortwords(1)

Storage in memory



- The idea of the sorting algorithm is that, instead of copying and swapping the arrays of words, just the pointers are swapped in case words are out of order. We send the function sort_words the array w of pointers, and its size n.

Example: sortwords in your textbook



Example: sortwords(2)

```
void sort_words (char *w[], int n) {
/* n elements (words) to be sorted */
int i, j;
for(i = 0; i < n; ++i)
    for(j = i+1; j < n; ++j)
        if ( strcmp(w[i], w[j]) > 0)
            swap ( &w[i], &w[j] );
}
void swap (char **p, char **q) {
    char * tmp;
    tmp = *p;
    *p =  *q;
    *q = tmp;
}
```

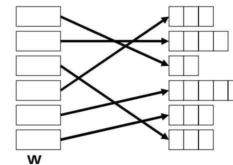
Compares each pair of string pointed to by addresses.

If they are out of order swap the pointers.

The arguments passed to the swap function are addresses of pointers to char or equivalently pointers to pointers to char. Hence they are declared as output parameters in the function swap.

Example: sortwords(3)

Pointer assignment after sorting



- Reading through the words pointed to by the elements of w, we obtain the sorted list of words.
- Advantages
  - Pointer requires less space
  - Faster copy involving only pointers
  - Original list is preserved.

Summary

- The two most important questions dealing with strings are the following:
  1. Is there enough room to perform the given operation?
  2. Does the created string end in '\0'?
- When we write our own string-building functions, we must be sure that a null character is inserted at the end of every string. This inclusion of the null character is automatic for constant string.
- The caller of a string function must provide space into which the called function can store its result.
- Proper behavior is entirely dependent on the availability of adequate space in the output parameter even though the called function does not require the user to provide this size as an input argument.

Remember

- I will be out of town in the coming week.
  - On 2nd and 4th April classes will be taken by Shahriar.
  - You can go to the lab, but there will be no grading and no TA will be there. I will ask the student consultants to be there to help you.
- There will be a quiz in either 2nd or 4th april.
  - Will cover upto String.
- Do not get too comfortable with the LOOOOG lab deadlines.
  - 48 hours deadlines are coming soon.
- I will discuss array (specially multidimensional) a little bit more once I get back.