Strings

Strings are Character Arrays

- Strings in C are simply arrays of characters.
  - Example: `char s [10];`
- This is a ten (10) element array that can hold a character string consisting of ≤ 9 characters.
- This is because C does not know where the end of an array is at run time.
  - By convention, C uses a NULL character '\0' to terminate all strings in its library functions
- For example:

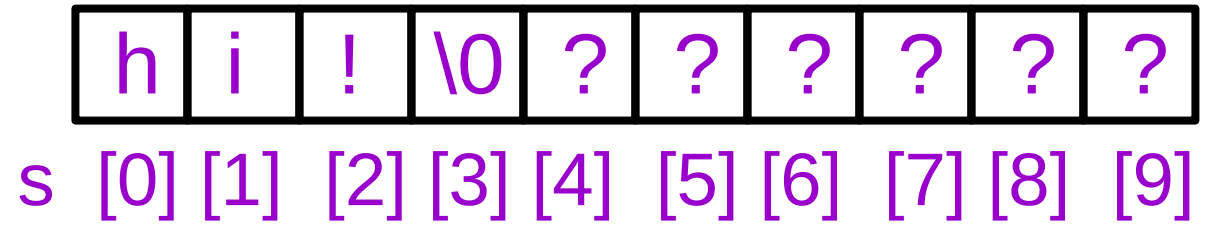
  ```
  char str [10] = {'u', 'n', 'l', 'x', '\0'};
  ```

- It's the string terminator (not the size of the array) that determines the length of the string.

Accessing Individual Characters

- The first element of any array in C is at index 0. The second is at index 1, and so on ...

```
char s[10];
s[0] = 'h';
s[1] = 'i';
s[2] = '!';
s[3] = '\0';
```



- This notation can be used in all kinds of statements and expressions in C:
- For example:

```
c = s[1];
if (s[0] == '-') …
switch (s[1]) ...
```

String Literals

- String literals are given as a string quoted by double quotes.
  - printf("Long long ago.");
- Initializing char array ...
  - char s[10]="unix"; /* s[4] is '\0'; */
  - char s[ ]="unix"; /* s has five elements */

Printing with printf ( )

- Example:

```
char str[ ] = "A message to display";
printf ("%s\n", str);
```

- printf expects to receive a string as an additional parameter when it sees %s in the format string
  - Can be from a character array.
  - Can be another literal string.
  - Can be from a character pointer (more on this later).

- printf knows how much to print out because of the NULL character at the end of all strings.
  - When it finds a \0, it knows to stop.

Example

```
char str[10]="unix and c";

 printf("%s", str);
 printf("\n");
 str[6]='\0';
 printf("%s", str);
 printf("\n");

 printf("\n");
 printf(str);
 printf("\n");
 str[2]='%';
 printf(str);
 printf("\n");
```

Printing with puts( )

- The puts function is a much simpler output function than printf for string printing.
- Prototype of puts is defined in stdio.h

  int puts(const char * str)

  – This is more efficient than printf

    - Because your program doesn't need to analyze the format string at run-time.

- For example:

```
char sentence[] = "The quick brown fox\n";
puts(sentence);
```

- Prints out:

```
The quick brown fox
```

Inputting Strings with gets( )

- gets( ) gets a line from the standard input.
- The prototype is defined in stdio.h
  char *gets(char *str)
  - str is a pointer to the space where gets will store the line to, or a character array.
  - Returns NULL upon failure. Otherwise, it returns str.

```
char your_line[100];
printf("Enter a line:\n");
gets(your_line);
puts("Your input follows:\n");
puts(your_line);
```

  - You can overflow your string buffer, so be careful!

Inputting Strings with scanf ( )

- To read a string include:
  - %s scans up to but not including the “next” white space character
  - %***n***s scans the next ***n*** characters or up to the next white space character, whichever comes first
- Example:

```
scanf ("%s%s%s", s1, s2, s3);
scanf ("%2s%2s%2s", s1, s2, s3);
```

  - Note: No ampersand(&) when inputting strings into character arrays! (We’ll explain why later …)
- Difference between gets
  - gets( ) read a line
  - scanf("%s") read up to the next space

An Example

```
#include <stdio.h>
int main () {
    char lname[81], fname[81];
    int  count, id_num;
    puts ("Enter the last name, firstname, ID number
    separated");
    puts ("by spaces, then press Enter \n");
    count = scanf ("%s%s%d", lname, fname,&id_num);
    printf ("%d items entered: %s %s %d\n",
                    count,fname,lname,id_num);
    return 0;
}
```

The C String Library

- String functions are provided in an ANSI standard string library.
  - Access this through the include file:

    #include <string.h>
  - Includes functions such as:
    - Computing length of string
    - Copying strings
    - Concatenating strings
  - This library is guaranteed to be there in any ANSI standard implementation of C.

strlen

- strlen returns the length of a NULL terminated character string:

```
size_t  strlen (char * str) ;
```

- Defined in string.h
- size_t
  - A type defined in string.h that is equivalent to an unsigned int
- char *str
  - Points to a series of characters or is a character array ending with '\0'
  - The following code has a problem!

```
char a[5]={‘a’, ’b’, ’c’, ’d’, ’e’};
strlen(a);
```

strcpy

- Copying a string comes in the form:

  char *strcpy (char * destination, char * source);

- A copy of source is made at destination
  - source should be NULL terminated
  - destination should have enough room (its length should be at least the size of source)
- The return value also points at the destination.

strcat

- Included in string.h and comes in the form:

  char * strcat (char * str1, char * str2);

  - Appends a copy of str2 to the end of str1
  - A pointer equal to str1 is returned

- Ensure that str1 has sufficient space for the concatenated string!
  - Array index out of range will be the most popular bug in your C programming career.

Example

```
#include <string.h>
#include <stdio.h>
int main() {
    char str1[27] = "abc";
    char str2[100];
    printf("%d\n",strlen(str1));
    strcpy(str2,str1);
    puts(str2);
    puts("\n");
    strcat(str2,str1);
    puts(str2);
}
```

Comparing Strings

- C strings can be compared for equality or inequality
- If they are equal - they are ASCII identical
- If they are unequal the comparison function will return an int that is interpreted as:

  < 0 : str1 is less than str2

  0 : str1 is equal to str2

  > 0 : str1 is greater than str2

strcmp

- Four basic comparison functions:

  int strcmp (char *str1, char *str2) ;

  - Does an ASCII comparison one char at a time until a difference is found between two chars
    - Return value is as stated before
  - If both strings reach a '\0' at the same time, they are considered equal.

  int strncmp (char *str1, char * str2, size_t n);

  - Compares n chars of str1 and str2
    - Continues until n chars are compared or
    - The end of str1or str2 is encountered

  – Also have strcasecmp() and strncasecmp() which do the same as above, but ignore case in letters.

Example

- An Example - strncmp

```
int main() {
   char str1[] = "The first string.";
   char str2[] = "The second string.";
   size_t  n, x;
   printf("%d\n", strncmp(str1,str2,4) );
   printf("%d\n", strncmp(str1,str2,5) );
}
```

Searching Strings (1)

- There are a number of searching functions:
  - char * strchr (char * str, int ch) ;
    - strchr search str until ch is found or NULL character is found instead.
    - If found, a (non-NULL) pointer to ch is returned
      - Otherwise, NULL is returned instead.
  - You can determine its location (index) in the string by:
    - Subtracting the value returned from the address of the start of the string
      - More pointer arithmetic … more on this later!

Example

Example use of strchr:

```
#include<stdio.h>
#include<string.h>
int main() {
   char ch='b', buf[80];
   strcpy(buf, "The quick brown fox");
   if (strchr(buf,ch) == NULL)
       printf ("The character %c was not found.\n",ch);
   else
       printf ("The character %c was found at position
                %d\n", ch, strchr(buf,ch)-buf+1);
}
```

Searching Strings (2)

- Another string searching function:

  char * strstr (char * str, char * query) ;

  - strstr searches str until query is found or a NULL character is found instead.
  - If found, a (non-NULL) pointer to str is returned.
    - Otherwise, NULL is returned instead.

sprintf

```
#include <stdio.h>

int sprintf( char *s, const char *format, …);
```

- Instead of printing to the stdin with printf(…), sprintf prints to a string.
- Very useful for formatting a string, or when one needs to convert integers or floating point numbers to strings.
- There is also a sscanf for formatted input from a string in the same way scanf works.

Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
      char result[100];
      sprintf(result, "%f", (float)17/37 );
      if (strstr(result, "45") != NULL)
        printf("The digit sequence 45 is in 17
                    divided by 37. \n");
      return 0;
}
```

Converting Strings to Numbers (1)

- Contained in <stdlib.h> and are often used

  int atoi (char *ptr);

  - Takes a character string and converts it to an integer.
  - White space and + or - are OK.
  - Starts at beginning and continues until something non-convertible is encountered.

- Some examples:

String	Value returned
"157"	157
"-1.6"	-1
"+50x"	50
"twelve"	0
"x506"	0

Converting Strings to Numbers (2)

long atol (char *ptr) ;

- Same as atoi except it returns a long.

double atof (char * str);

- Handles digits 0-9.
- A decimal point.
- An exponent indicator (e or E).
- If no characters are convertible a 0 is returned.

◆ Examples:

String	Value returned
"12"	12.000000
"-0.123"	-0.123000
"123E+3"	123000.000000
"123.1e-5"	0.001231