Strings (part 1)

CS2023 Winter 2004

Outcomes: Strings (part 1)

- “C for Java Programmers”, Chapter 9
- Other textbooks on C on reserve
- After the conclusion of this section you should be able to
  - Use functions that process single characters
  - Define strings and string constants
  - Perform formatted and line-oriented string I/0

Character Processing Functions

In **`ctype.h`** header

To classify:

- **`int isalnum(int c)`** is **`c`** an alphanumeric
- **`int isalpha(int c)`** is **`c`** an alphabetic letter
- **`int islower(int c)`** is **`c`** a lower case letter
- **`int isupper(int c)`** is **`c`** an upper case letter
- **`int isdigit(int c)`** is **`c`** a digit
- **`int isxdigit(int c)`** is **`c`** a hexadecimal digit
- **`int isodigit(int c)`** is **`c`** an octal digit

Character Processing Functions

To classify (continued):

- `int isprint(int c)` is `c` printable (not a control character)
- `int isgraph(int c)` is `c` printable (not a space)
- `int ispunct(int c)` is `c` printable (not space or alphanumeric)
- `int isspace(int c)` is `c` whitespace

To convert:

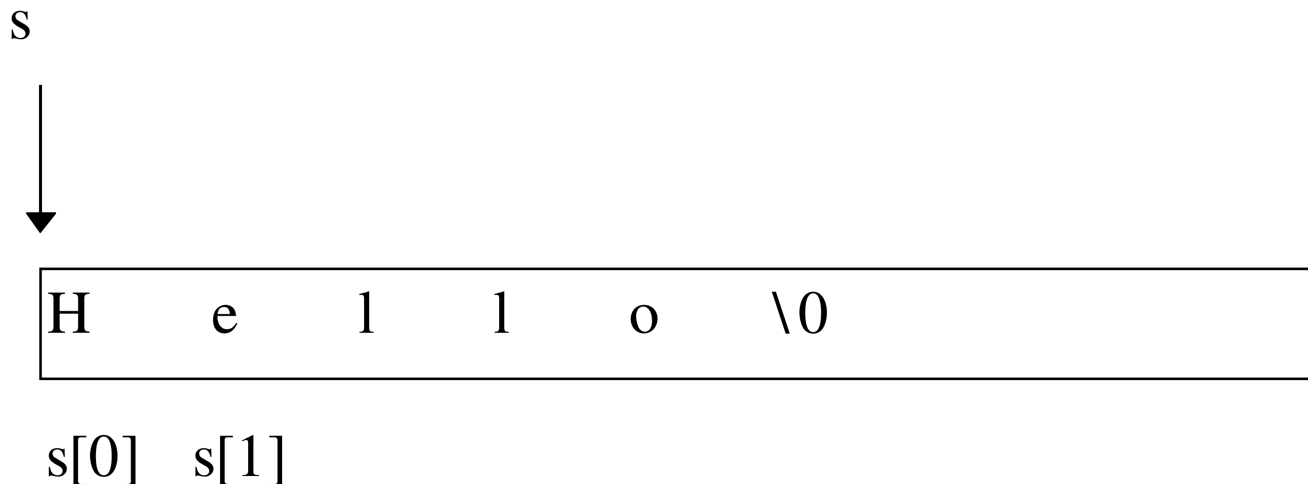
- `int tolower(int c)`
- `int toupper(int c)`

What does this program do?

```
#include <stdio.h>
#include <ctype.h>

int main(){

  int c, v;
  v=0;
  while(isdigit(c = getchar()))
    v = v * 10 + (c - '0');

  printf("%d\n", v);

  return 1;
}
```

Strings in C

- C stores a string in a block of memory.
- The string is terminated by the **\0** character:



Definitions of Strings

Strings are defined as pointers to characters:

```
char *s;
```

To allocate memory for a string that can hold up to 10 characters:

```
#define SIZE 10
if((s = malloc((SIZE+1)*sizeof(char)))
  == NULL) …
s[0] = '\0';
```

"Memory allocation" *Idiom*

Definitions of Strings

`s[0] = '\0‘` makes s a null string.

Safer to use **`calloc`**, which initializes the block to null:

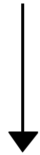
```
if((s = calloc(n+1, sizeof(char)))
  == NULL) …
```

Definitions of Strings

- i-th character of a string
  - To refer to the **`i`**-th character in the string **`s`**, use **`s[i]`**, where **`0 <= i <`** length of **`s`**.
- Careful: Initialized pointers are not necessarily initialized strings.
  - If an initialized pointer points to a memory block that does not contain the null character, the string is not initialized.

Definitions of Strings

s



H	e	l	l	o	\0

s[0] s[1]

- The string **`s`** above has the length 5; **`"hello"`**
- **`s+1`** (a suffix of **`s`**) has length 4; **`"ello"`**
- **`s+2`** (a suffix of **`s`**) has length 3; **`"llo"`**
- **`s+5`** (a suffix of **`s`**) has a length 0; (it is a *null* string) **`""`**
- However, **`s+6`** is not well defined.

String Constants

```
char *name = "Kasia";
char *p;
p = “abc”
```

- Also known as *string literals*
- **“a”** represented as a pointer to memory location that contains character **a** (followed by null character)
- Character constant **‘a’** is represented by an integer

String Constants

- The block of memory for a string constant should not be modified, as this may cause programs to behave erratically

```
char *p = “abc”, *q = “abc”
```

  - Some compilers will store **“abc”** just once, making both **p** and **q** point to it. If **“abc”** changed through **p** , then string that **q** points to also affected.

- Therefore, do not reset any of the characters in the constant string:

```
char *name = "Kasia";
name[0] = 'B';
```

Character Arrays vs. Character Pointers

`char date[] = "June 14";`

- Declares **`date`** to be an array of characters
- Characters can be modified, like the elements of any array
- **`date`** is an array name

`char *date = "June 14";`

- Declares **`date`** to be a pointer to a string constant
- Characters shouldn't be modified
- **`date`** is a pointer variable that can be made to point to other strings during program execution.

Initialization of Strings

- To create a string to be modified, it is programmer's responsibility to either set up an array of characters

```
char s[SIZE+1], *p;
p = s;
```

or to allocate memory for the string:

```
s = calloc((SIZE+1)*sizeof(char))
```

Initialization of Strings

```
char *p;
p[0] = 'a';
p[1] = 'b';
p[2] = 'c';
p[3] = '\0';
```

We don't know where **p** is pointing!

Initialization of Strings

```
char date1[8] = “June 14”
char date[] = “June 14”
```

J	u	n	e		1	4	\0

```
char date2[9] = “June 14”
```

J	u	n	e		1	4	\0	\0

```
char date3[7] = “June 14”
```

J	u	n	e		1	4

String Parameters

C strings can be used as parameters as *any other pointers*

```
void modify(char *s) {
    s[0] = toupper(s[0]);
}
```

"Memory allocation" *Idiom*

```
char *p;  /* modify(p); */
if((p = calloc(10, sizeof(char))) == NULL)
  error
p[0] = 'h'; p[1] = 'o'; /* p[2] == '\0' */
modify(p);
modify(p+1);
char *q = "hello";
modify(q);
```

"String suffix" *Idiom*

String Parameters

```
/* Same as strlen() */
int length(const char *s) {
    char *p;

    for(p = s; *p; p++) /* *p != '\0'  */
        ;
    return p - s;
}
```

Traversing a String

```
for(p = s; *p; p++)
    use *p
```

```
char *strdup(const char *s) { /* return copy of s
*/
  char *kopy;        /* copy of s */
  char *ps;          /* used for copying */
  char *pkopy;       /* for copying */

if((kopy
=calloc((length(s)+1),sizeof(char)))==NULL)
      return NULL;


  /* memory allocated, now copy */
  for (ps = s, pkopy = kopy; *ps; ps++, pkopy++)
     *pkopy = *ps;
  *pkopy = *ps;

  return kopy;
}
```

"String Traversal" *Idiom*

```c
char *modify(const char *s) {
/* return a copy of s modified */
    char *news;

    if((news = strdup(s)) == NULL)
        return NULL;

    news[0] = toupper(news[0]);

    return news;
}
char *q = modify("c for java");
char *s = modify("c for java" + 6);
```

"i-th character" *Idiom*

(the last one returns “Java”)

"String Suffix" *Idiom*

String Parameters & Return Values

```
void modify1(const char *s, char
  **news) {
/* return through parameter a copy of s
  modified*/
    if(s == NULL)
        return;

    *news = strdup(s);
    (*news)[0] = toupper((*news)[0]);
}
char *p;
modify1("hello", &p);
```

Formatted String I/O

- The formal control string **`%s`** is used for string I/O.
- Leading whitespace characters are skipped in a search for the first
- non-whitespace character, and input stops when a *word* is read
- (a word is a sequence of characters not containing any whitespace).
- Therefore, **`scanf()`** can read at most one word.

Formatted String I/O

- To input a string use:

```
scanf("%s", s)
```

- rather than:

```
scanf("%s", &s)
```

- make sure that **s** is initialized; i.e. there is some memory allocated for **s** (for example, using **calloc()**)
- make sure that there is *enough* memory allocated for **s**, and consider using the *field width* to avoid overflow.

```
if(scanf("%10s", s) != 1)
    error
```

```
int lower(char *s) { /* return number of l.c. letters */
    int i;
    char *q;
    for(i = 0, q = s; *q; q++)
       if(islower(*q))
         i++;
    return i;
}
int main() {
  const int M = 10;
  char *p;
  if((p = calloc(M + 1, sizeof(char)) == NULL)
     return EXIT_FAILURE;
  if(scanf("%10s", p) != 1)
     return EXIT_FAILURE;
  printf("%d lower case letters in %s\n", lower(p), p);
  return EXIT_SUCCESS;
}
```

Formatted String I/O

There are two formatted string I/O operations:

```
int sscanf(s, "format", arguments)
int sprintf(s, "format", arguments)
```

```
#define N 100
int i; double d; char *s;

if((s = calloc(N+1,sizeof(char))) ==
 NULL)
    return EXIT_FAILURE;
sprintf(s, "%s %d %f", "test", 1, 1.5);
if(sscanf(s+4, "%d%f", &i, &d) != 2)
  ...
```

Line-Oriented String I/O

`char* fgets(char *buf, int n, FILE *in);`

- reads a line from the file **`in`**, and stores it in the block pointed to by **`buf`**. Stops reading when:
  - **`n-1`** characters have been read
  - end-of-line has been encountered; (**`\n`** is stored at the end of **`buf`**)
  - end-of-file has been encountered.
- In any case, **`buf`** is always properly terminated (**`\0`** is stored).
- The function returns **`buf`** if successful and **`NULL`** if no characters have been read or there has been a reading error.

Line-Oriented String I/O

Read a line at most n-1 characters from a file

```
if(fgets(buffer, n, f) == NULL)
    error
```

```
/* find the length of the longest line; at most max */
long longest(const char *fname, const int max) {
   char *line;
   FILE *f;
   long i = 0;
   if((f = fopen(fname, "r")) == NULL)
       return -1;
   if((line = calloc(max + 1, sizeof(char))) == NULL) {
      fclose(f); return -1;
   }
   while(fgets(line, max, f) != NULL)
      if(strlen(line) > i)
         i = strlen(line);
   free(line);
   if(fclose(f) == EOF)
      return -1;
   return i - 1;
}
```