CSC 270 – Survey of Programming Languages

C Lecture 4 – Strings

C-String Values

- The most basic way to represent a string of characters in C++ is using an array of characters that ends with a null byte.
- Example
  ```
  printf("Hello");
  ```
C-String Variables

• A C-string variable is an array of characters that is expected to be terminated with a null byte (\0).

• Example
  ```c
  char s[10];
  strcpy(s, "Hello");
  ```

Initializing C Strings

• Writing:
  ```c
  char yourString[11] = "Do Be Do";
  char myString[] = "Do Be Do";
  ```
  will initialize both strings with “Do Be Do” with a null byte immediately after “Do Be Do”.

• Writing
  ```c
  char s[] = "abc";
  ```
  and
  ```c
  char s[] = {'a', 'b', 'c', '\0'};
  ```
  produce the same string
Pitfall: Using =

• C-strings and C-string variables are different from other data types.
• While you can declare a string by writing
  ```c
  char s[] = "Hello";
  ```
  You cannot write
  ```c
  char s[10];
  s = "Hello";
  ```

Pitfall: Using ==

• C-strings and C-string variables cannot be compared correctly using ==.
• If you write
  ```c
  if (s1 == s2)
    printf("The same");
  else
    printf("Different");
  ```
  you are comparing their starting addresses.
• Instead, use
  ```c
  if (strcmp(s1, s2) == 0)
    ... ...
  ```
Predefined Functions in `<string.h>`

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Caution</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strcpy(s, t)</code></td>
<td>Copies t into s</td>
<td>No bounds checking</td>
</tr>
<tr>
<td><code>strncpy(s, t, n)</code></td>
<td>Copies t into s but no more than n characters are copied</td>
<td>Not implemented in older versions of c</td>
</tr>
<tr>
<td><code>strcat(s, t)</code></td>
<td>Concatenates t to the end of s</td>
<td>No bounds checking</td>
</tr>
<tr>
<td><code>strncat(s, t, n)</code></td>
<td>Concatenates t to the end of s but no more than n characters</td>
<td>Not implemented in older versions of c</td>
</tr>
<tr>
<td><code>strlen(s)</code></td>
<td>Returns the length of s (not counting '0')</td>
<td></td>
</tr>
<tr>
<td><code>strcmp(s, t)</code></td>
<td>Returns 0 if s == t, &lt; 0 if s &lt; t, &gt; 0 if s &gt; t</td>
<td>No bounds checking</td>
</tr>
<tr>
<td><code>strncmp(s, t, n)</code></td>
<td>Same as strcmp but compares no more than n characters</td>
<td>Not implemented in older versions of c</td>
</tr>
</tbody>
</table>

**strcpy()**

- `strcpy(s, t)` copies the contents of `t` into `s`.
- This is standard way of assigning a value to a string, i.e., we copy it character by character instead of copying over the address at which it starts.
**strcpy() – A Example**

```c
#include <stdio.h>

int main(void)
{
    char s[80];

    strcpy(s, "Let's learn strings");
    printf("%s", s);
    return(0);
}
```

**Output**
Let's learn strings

---

**strncpy()**

- `strncpy(s, t, n)` copies the contents of `t` into `s` but no more than `n-1` characters.
- This is standard way of assigning a value to a string, i.e., we copy it character by character instead of copying over the address at which it starts.
`strncpy()` – An Example

```c
#include <stdio.h>
#include <string.h>

int main(void)
{
    char s[80], t[80];

    strcpy(s, "The quick brown fox jumped "
          "over the lazy dogs");
    strncpy(t, s, 10);
    /*
     * You need this in case s was more
     * than 10 characters
     */
    t[10] = '\0';

    printf("s = \"%s\"\n", s);
    printf("t = \"%s\"\n", t);

    return(0);
}
```
**strcat()**

- `strcat(s, t)` concatenates `s` and `t`, saving the new string in `s`.
- Example
  ```c
  strcpy(s, "Robert");
  strcpy(t, "Michael");
  strcat(s, t);
  printf("s = \"%s\"\n", s);
  ``
  Output
  ```
  s = "RobertMichael"
  ```

**strncat()**

- `strncat(s, t, n)` concatenates `s` and `t`, saving the new string in `s`. At most it will copy `n` characters.
- Example
  ```c
  strcpy(s, "Robert");
  strcpy(t, "Michael");
  strncat(s, t, 5);
  printf("s = \"%s\"\n", s);
  ```
  Output
  ```
  s = "RobertMicha"
  ```
**strlen()**

- **strlen(s)** returns the number of characters in the string `s`.
- Example
  ```c
  strcpy(s, "The quick brown fox");
  len = strlen(s);
  printf("\"%s\" has %d characters.\n", s, len);
  ```
- **Output**
  "The quick brown fox" has 19 characters.

**strcmp()**

- **strcmp(s, t)** compares `s` and `t` returns an integer:
  - If `s` precedes `t` in collating sequence, `strcmp(s, t) < 0`
  - If `s` and `t` are the same, `strcmp(s, t) = 0`
  - If `s` follows `t`, `strcmp(s, t) > 0`
**strcmp()** – An example

```c
#include<stdio.h>

char r[] = "The quick brown fox",
    s[] = "His quick brown fox",
    t[] = "Your quick brown fox";

int main(void){
    printf("strcmp(r, s) = %d\n", strcmp(r, s));
    printf("strcmp(r, t) = %d\n", strcmp(r, t));
    printf("strcmp(r, r) = %d\n", strcmp(r, r));

    return(0);
}
```

Output

```
strcmp(r, s) = 1
strcmp(r, t) = -1
strcmp(r, r) = 0
```

---

**strncmp()**

- `strncmp(s, t)` compares up to the first n characters of `s` and `t` returns an integer:
- If `s` precedes `t` in collating sequence, `strncmp(s, t) < 0`
- If `s` and `t` are the same, `strncmp(s, t) = 0`
- If `s` follows `t`, `strncmp(s, t) > 0`
```c
#include <stdio.h>

char r[] = "The quick brown fox",
    s[] = "His quick brown fox",
    t[] = "Your quick brown fox";

int main(void) {
    printf("strncmp(r, s, 8) = %d\n",
            strncmp(r, s, 8));
    printf("strncmp(r, t, 8) = %d\n",
            strncmp(r, t, 8));
    printf("strncmp(r, r, 8) = %d\n",
            strncmp(r, r, 8));

    return(0);
}
```

**Output**

```
strncmp(r, s, 8) = 1
strncmp(r, t, 8) = -1
strncmp(r, r, 8) = 0
```
Example: Command-Line Arguments

- Command line parameters are entered when invoking the program using a command-line interface.
- Example:
  
  myProg This is a test

```c
#include  <stdio.h>

int main(int argc, char *argv[]) {
    int   i;

    for (i = 0;  i < argc;  i++)
        printf("Argument #\d is \s\n", i, argv[i]);

    return(0);
}
```
Output for `args.c`

C:\Users\Robert M. Siegfried\Documents\Visual Studio 2008\Projects\args\Debug>args This is a test
Argument #0 is args
Argument #1 is This
Argument #2 is is
Argument #3 is a
Argument #4 is test

C:\Users\Robert M. Siegfried\Documents\Visual Studio 2008\Projects\args\Debug>

String: Input and Output

- In addition to `scanf()` and `printf()`, there are other input methods available when working with characters and strings:
  - `getchar()`
  - `gets()`
  - `gets_s()`
  - `putchar()`
  - `puts()`
**getchar()**

- `getchar()` returns a character from the input stream.
- The program will read input until the Enter (or Return) key is pressed. It will then return the input from the line character by character.
- Older versions of `getchar()` define 0 as the End of File marker (`EOF`); newer versions use -1 instead; therefore, it is accepted practice to use `getchar()` to assign the character to an integer variable.

**getchar() – An Example**

```c
#include <stdio.h>
/* main() – Copy input to output */
int main(void) {
    int c;

    c = getchar();
    while (c != EOF) {
        printf("%c", c);
        c = getchar();
    }
    return(0);
}
```
Output From Our Example

This is the start of something big
This is the start of something big

gets ()

- `gets()` can be used to read an entire string up until the next newline character.
- Because `gets()` does not include bounds checking, it was deprecated in the 1999 C standard and made obsolete in the 2011 standard.
gets_s()

- **gets_s()** is an improved version of `gets` that does bounds checking.

- Example
  ```c
  int main(void) {
      char s[81];

      gets_s(s, 30);
      printf("\"%s\"\n", s);
      return(0);
  }
  ```

putchar()

- **putchar()** displays the next character that it is given as a parameter.

- Example
  ```c
  int main(void) {
      int c;

      while ((c = getchar()) != EOF)
          putchar(c);

      return(0);
  }
  ```
puts()

- **puts()** will print a string passed as a parameter.
- Example
  ```c
  int main(void) {
      int s[41];
      gets_s(s, 40);
      puts(s);
      return(0);
  }
  ```

Functions in `<ctype.h>`

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>toupper(c)</td>
<td>Returns the upper case version of the character</td>
<td>c = toupper('a');</td>
</tr>
<tr>
<td>tolower(c)</td>
<td>Returns the lower case version of the character</td>
<td>c = tolower('A');</td>
</tr>
<tr>
<td>isupper(c)</td>
<td>Returns true if c is an upper case letter</td>
<td>if (isupper(c)) printf(&quot;upper case&quot;);</td>
</tr>
<tr>
<td>islower(c)</td>
<td>Returns true if c is a lower case letter</td>
<td>if (islower(c)) printf(&quot;lower case&quot;);</td>
</tr>
<tr>
<td>isalpha(c)</td>
<td>Returns true if c is a letter</td>
<td>if (isalpha(c)) printf(&quot;it’s a letter&quot;);</td>
</tr>
<tr>
<td>isdigit(c)</td>
<td>Returns true if c is a digit (0 through 9)</td>
<td>if (isdigit(c)) printf(&quot;it’s a number&quot;);</td>
</tr>
</tbody>
</table>
Functions in `<ctype.h>` (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isalnum(c)</code></td>
<td>Returns true if c is alphanumeric</td>
<td><code>if (isalnum('3')) printf(&quot;alphanumeric&quot;);</code></td>
</tr>
<tr>
<td><code>isspace(c)</code></td>
<td>Returns true if c is a white space character</td>
<td><code>while (isspace(c)) c = getchar();</code></td>
</tr>
<tr>
<td><code>ispunct(c)</code></td>
<td>Returns true if c is a printable character other than number, letter or white space</td>
<td><code>if (ispunct(c)) printf(&quot;punctuation&quot;);</code></td>
</tr>
<tr>
<td><code>isprint(c)</code></td>
<td>Returns true if c is a printable character</td>
<td></td>
</tr>
<tr>
<td><code>isgraph(c)</code></td>
<td>Returns true if c is a printable character other an white space</td>
<td></td>
</tr>
<tr>
<td><code>isctrl(c)</code></td>
<td>Returns true if c is a control character</td>
<td></td>
</tr>
</tbody>
</table>

In C, Characters are Integers

- In many ways, C considers characters to be 8-bit unsigned integers. For this reason, many string functions return an `int` value.
- Using `printf` to write `toupper('a')` will not only write 'A' but the numeric code that represents 'A' depending on the field specifier that you use.
**toupper() – An Example**

```c
printf("%d	 %c\n", toupper('a'), toupper('a'));
```

**Output**

```
65   A
```

**String Processing in C**

- Despite C's lack of direct support of strings, it can be used easily to handle all kinds of string processing.
```c
#include
#define YES 1
#define NO 0
/
* main() - Count the lines, words and
* characters in the input
*/
int main(void) {
    int c, nl, nw, nc, inword;
    inword = NO; /* We aren't in a word yet */
    nl = nw = nc = 0;
    while ((c = getchar()) != EOF) {
        nc++;
        if (c == 'n')
            nl++;
        if (c == ' ' || c == 'n' || c == 't')
            inword = NO; /* Our word ended */
        else if (inword == NO) {
            /* New word started */
            inword = YES;
            nw++;
        }
    }
```
```c
#include <stdio.h>

int main(void) {
    int c, i, nwhite, nother;
    int ndigit[10];
    nwhite = nother = 0;

    for (i = 0; i < 10; i++)
        ndigit[i] = 0;

    printf("Lines = %d	word = %d	"
            "characters = %d\n", nl, nw, nc);
    return(0);
}
```

CountDigits.c
while ((c = getchar()) != EOF)
    /* c - '0' is the value of the digit */
    if ('0' <= c && c <= '9')
        ++ndigit[c-'0'];
    else if (c == ' ' || c == '	'
                || c == '
')
        nwhite++;
    else nother++;

    printf("digits = ");
    for (i = 0; i < 10; i++)
        printf(" %d ", ndigit[i]);

    printf("\nwhite space = %d\t" 
           "other = %d\n", nwhite, nother);
    return(0);
}

squeeze.c

#include <stdio.h>

char line[] =
{"This is the start of a very big deal of a line"};

void squeeze(char s[], char c);

int main(void) {
    printf("My line is:\n%s\n", line);
    squeeze(line, 'i');
    printf("My line is:\n%s\n", line);
    return(0);
}
/*
 * squeeze() - delete all c's from s
 */
void squeeze(char s[], char c) {
    int i, j;
    for (i = j = 0; s[i] != '\0'; i++)
        if (s[i] != c)
            s[j++] = s[i];
    s[j] = '\0';
}

int strlen(char s[]) {
    int i;

    /* Advance to the null byte at the end */
    for (i = 0; s[i] != '\0'; i++)
        ;

    return(i);
}
```c
strcpy()

/*
 * strcpy() – Copy t into s one character at a time
 */
void strcpy(char s[], char t[]) {
  int i;

  /* Until we reach the final null byte */
  for (i = 0; t[i] != '\0'; i++)
    /* Copy the current character */
    s[i] = t[i];
  s[i] = '\0';
}

strcat()

/*
 * strcat() – Concatenate t to s
 */
void strcat(char s[], char t[]) {
  int i, j;

  /* Skip to the end of s */
  for (i = 0; s[i] != '\0'; i++)
    ;

  /* Copy t into the end of s */
  for (j = 0; t[j] != '\0'; i++, j++)
    s[i] = t[j];
  s[i] = '\0';
}
```
atoi()

/*
 * atoi() - The function accepts a character string
 * and translates it into the integer that it
 * represents. It assumes that the first
 * nondigit is the end of the number.
 */
int atoi(char s[]) {
    int i, n = 0;

    for (i = 0; '0' <= s[i] && s[i] <= '9'; i++)
        n = 10*n + s[i] - '0';

    return(n);
}

lower()

/*
 * lower() - Convert a character to lower case
 */
int lower(int c) {
    if ('A' <= c && c <= 'a')
        return(c + 'a' - 'A');
    else
        return(c);
}
char text[] = "Fourscore and seven years ago";
char dest[20];

char *s, *t;

s

s

t

t


\texttt{strncpy(dest, "abcd", 10);}

\texttt{memset(dest, '\0', sizeof(dest));}
\[ t = \text{strstr(text, "score");} \]

```
F o u r \text{score and seven}

\text{years ago \0}
```

\[ \text{strncpy(dest, t, 10);} \]

```
\text{score and seven}

\text{\0 \0 \0 \0 \0 \0}
```

```
\text{\0 \0 \0 \0 \0}
```
```c
#include <string.h>
#include <stdio.h>

char    text[] = "Fourscore and seven years ago";

int main(void) {
    char    dest[20];
    char    *s, *t;
    int     i;

    strncpy(dest, "abcd", 10);
    printf("%s\n", dest);
}```
memset(dest, '\0', sizeof(dest));
t = strstr(text, "score");
strncpy(dest, t, 10);
dest[10] = '\0';

for (i = 0; i < 20; i++)
    putchar(dest[i]);
putchar('\n');
for (i = 0; i < 20; i++)
    printf("%d ", (int)dest[i]);
putchar('\n');

for (i = 0; i < 20; i++)
    printf("%d ", (int)t[i]);
putchar('\n');

printf("%s\n", t);
printf("%s\n", dest);
return(0);
The output from `substring.c`

```
abcd
score and
115  99  111  114  101  32  97  110  100
32  0  0  0  0  0  0  0  0  0
115  99  111  114  101  32  97  110  100
32  115  101  118  101  110  32  121  101
97  114
score and seven years ago
score and
```

`strstr()` Example

```
/* strstr example */
#include <stdio.h>
#include <string.h>

int main () {
    char str[] = "This is a simple string";
    char * pch;
    int i;
```
pch = strstr (str,"simple");
puts(pch);
i = pch-str;
printf("i = %d\n", i);
strncpy (pch,"sample",6);
puts (str);
return 0;
}