Characters and Strings

Constants

- Characters are the fundamental building blocks of source programs
- Character constants
  - One character surrounded by *single quotes*
  - ‘A’ or ‘?’
  - Actually an *int* value represented as a character in single quotes
- Special characters and non-graphic characters
  - Denoted by preceding other characters with a backslash \
    \n    \t horizontal tab
    \v vertical tab
    \b backspace
    \r carriage return
    \f form feed
    \\ backslash
    \' single quote
    \a alert
  - Another form \ddd where each d is an octal digit
    * ddd specifies the desired internal value of a character
  - **NULL** character
    * Indicated by the escape sequence \0
    * All bits corresponding to the character are zero
    * Not the same as the ASCII character 0 which is represented by hexadecimal sequence 30
- String constants
  - Also known as *literals*
  - Sequence of characters surrounded by double quotes
  - Backslash can be used for special characters
  - Double quotes within the string written as "
  - "ABC" or "%d\n"
  - A null character (\0) is added immediately after the final character of a string
    * "ABC" is stored in four bytes as "ABC\0"
    * String constant "A" is different from character constant ‘A’
  - String constant "ABC\nDEF" is a two-line string
  - *Important*: A character constant is enclosed in single quotes while string constants are enclosed in double quotes

Fundamentals of strings and characters
- String is accessed via a pointer to its first character
- String is also viewed as an array of characters, with ‘\0’ being used to terminate the string
- A string variable is declared as a pointer to character (or array)

```c
char color[] = "blue";
char * color_ptr = color;
```

- What is the number of bytes reserved for the string in the above cases?
- You must declare enough space for the string (especially if you intend to increase the size of the string later on)
- Never forget to account for the NULL character when allocating space for strings

- Assigning a string to another variable
  - Since strings are implemented by a pointer to the first element of the character array, they cannot be copied by a simple assignment
  - `color_ptr = color` does not copy the string into `color_ptr` but merely copies the pointer value
  - You may have to use a function such as `strcpy` to actually achieve the copy operation
  - Assuming that `color_ptr` has enough memory allocated, the following function can also achieve the string copy

```c
void copy_string ( char * color_ptr, char * const color )
{
    char * in = color;
    char * out = color_ptr;
    while ( *out++ = *in++ );
}
```

- A string can be read by using `scanf` and the `%s` format specifier, but we will resort to using `fgets` to read strings from `stdio` and files, and `sscanf` to read them from within memory
- Be careful about mixing characters and strings, especially when passing them as parameters to functions
- Initializing strings
  - Can be done using either array or pointer notation
  - `char color[] = "blue";`
    * Compiler allocates the space and copies literal into that space
    * This string can be modified
  - `char * color = "blue";`
    * Compiler just creates a pointer to the string literal
    * This string cannot be modified

**Character handling library**

- Standard library in C to work with characters and strings
• You must include the header file `ctype.h` to use these functions

• In the following table, the type of variable `c` is an `int`, with its value restricted to that in an `unsigned char` (or that of the predefined constant `EOF`)

<table>
<thead>
<tr>
<th>Character classification macros or Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isalpha(c)</code></td>
</tr>
<tr>
<td><code>isupper(c)</code></td>
</tr>
<tr>
<td><code>islower(c)</code></td>
</tr>
<tr>
<td><code>isdigit(c)</code></td>
</tr>
<tr>
<td><code>isxdigit(c)</code></td>
</tr>
<tr>
<td><code>isalnum(c)</code></td>
</tr>
</tbody>
</table>
| `isspace(c)` | `c` is a whitespace character, `' ', '	', '', '
', ''` or vertical tab |
| `ispunct(c)` | `c` is a punctuation character (neither control nor alphanumeric) |
| `isprint(c)` | `c` is a printing character |
| `iscntrl(c)` | `c` is a control character or `' \b'` |
| `isascii(c)` | `c` is an ASCII character, code less than `0200` |
| `isgraph(c)` | `c` is a visible graphic character |

<table>
<thead>
<tr>
<th>Character conversion macros</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>toupper(c)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Character conversion functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tolower(c)</code></td>
</tr>
</tbody>
</table>

• Program to illustrate the character functions

```c
#include <stdio.h>

int main()
{
    int foo;

    printf ( "Please type in a hex integer: ");
    scanf ( "%x", &foo );

    printf ( "The decimal value of %x is %d\n", foo, foo );

    if ( isalpha ( foo ) )
    {
        printf ( "%x is a letter\n", foo );
        printf ( "The character equivalent of %x is %c\n", foo, foo );
        printf ( "%x is an %s letter\n", foo, isupper(foo) ? "uppercase" : "lowercase" );
    }

    if ( isdigit ( foo ) )
        printf ( "%x is a digit\n", foo );

    if ( isxdigit ( foo ) )
        printf ( "%x is a hexadecimal digit\n", foo );

    if ( isalnum ( foo ) )
        printf ( "%x is an alphanumeric character\n", foo );

    if ( isspace ( foo ) )
        printf ( "%x is a whitespace character\n", foo );
```
Strings

if ( ispunct ( foo ) )
    printf ( "%x is a punctuation character\n", foo );
if ( iscntrl ( foo ) )
    printf ( "%x is a control character\n", foo );
if ( isascii ( foo ) )
    printf ( "%x is an ASCII character %c\n", foo, foo );
else
{
    printf ( "The ASCII equivalent of %x is %x", foo, toascii(foo) );
    printf ( isprint(toascii(foo)) ? ", or character %c\n" : "\n", foo );
}
if ( isgraph ( foo ) )
    printf ( "%x is a visible graphic character %c\n", foo, foo );

String conversion functions

• Available in general utilities library (stdlib)
• Useful to convert a string of digits to integer or floating point values
• In the following table, str represents a string (array) and ptr represents a pointer to a character

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>atof ( str )</td>
<td>Convert string to double precision number</td>
</tr>
<tr>
<td>strtod ( str, ptr )</td>
<td>Convert string to double precision number</td>
</tr>
<tr>
<td>atoi ( str )</td>
<td>Convert string to integer</td>
</tr>
<tr>
<td>atol ( str )</td>
<td>Convert string to integer</td>
</tr>
<tr>
<td>strtol ( str, ptr, base )</td>
<td>Convert string to integer</td>
</tr>
</tbody>
</table>

• A word about the strxto? functions
  – If the value of ptr is not (char **)NULL, a pointer to character terminating the scan is returned to the location pointed to by ptr
  – If no number can be formed, *ptr is set to str and 0.0 is returned
  – base is of type int and, if its value is between 0 and 36, is used as the base for conversion
    * 0x or 0X are ignored if base is 16

Standard I/O library functions

• Require the file <stdio.h> to be included
• These functions are: getchar, gets, putchar, puts, printf, scanf
• Writing strings using printf and puts
  – Use %s conversion in printf to write strings
  – If the NULL character is missing, printf continues printing until it finds a NULL somewhere in memory
  – Use the conversion %.ps to print a part of the string

    char str[] = "Hello world";
    printf ( "First five characters are: %.5s\n", str );
– Elimination of . will print the string in full, if \( p \) is less than the string length
– If string is smaller than \( p \) characters, it is right justified
– String can be left justified by using \(-\), as in \%-ps

```c
#include <stdio.h>

int main()
{
    char str[] = "Hello world";
    printf ( "First five characters are: |%.5s|\n", str );
    printf ( "Trying to print in five character field gives: |%5s|\n", str );
    printf ( "Printing in 20 character field gives: |%20s|\n", str );
    printf ( "Left justification is achieved by: |%-20s|\n", str );
    return ( 0 );
}
```

String manipulation functions

- Require the inclusion of the standard string library `<string.h>`
- These functions operate on null-terminated strings
- These functions do not check for overflow of any receiving strings
- In the following table, \( s_1 \) and \( s_2 \) represent pointers to character type (or strings)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{strcat} ( s_1, s_2 )</td>
<td>Appends a copy of strings ( s_2 ) to the end of string ( s_1 )</td>
</tr>
<tr>
<td>\texttt{strncat} ( s_1, s_2, n )</td>
<td>Appends at most ( n ) characters from ( s_2 ) to ( s_1 )</td>
</tr>
<tr>
<td>\texttt{strcpy} ( s_1, s_2 )</td>
<td>Copies ( s_2 ) to ( s_1 ) until the null character</td>
</tr>
<tr>
<td>\texttt{strncpy} ( s_1, s_2, n )</td>
<td>Copies ( s_2 ) to ( s_1 ) until the null character, or ( n ) characters</td>
</tr>
<tr>
<td>\texttt{strdup} ( s )</td>
<td>Duplicates string and returns pointer to the new string</td>
</tr>
<tr>
<td>\texttt{strlen} ( s )</td>
<td>Number of characters in ( s ), not including the NULL character</td>
</tr>
</tbody>
</table>

- \texttt{strcat}, \texttt{strncat}, \texttt{strcpy}, and \texttt{strncpy} return a pointer to the null-terminated string \( s_1 \)
- In \texttt{strcpy} and \texttt{strncpy}, if the length of target string \( s_1 \) is more than the source string \( s_2 \), \( s_1 \) is padded with NULL characters
- \texttt{strdup} automatically allocates space for the new string
- \texttt{strdup} returns a NULL if it cannot allocate space for the duplicate string

String comparison functions

- These functions return an integer which is
  
  \[
  \begin{align*}
  0 & \quad \text{if the two strings are equal} \\
  > 0 & \quad \text{if first string is greater than second string (in alphabetic order)} \\
  < 0 & \quad \text{if first string is smaller than second string (in alphabetic order)}
  \end{align*}
  \]

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{strcmp} ( s_1, s_2 )</td>
<td>Lexicographically compare strings ( s_1 ) and ( s_2 )</td>
</tr>
<tr>
<td>\texttt{strncmp} ( s_1, s_2, n )</td>
<td>Lexicographically compare first ( n ) characters of strings ( s_1 ) and ( s_2 )</td>
</tr>
<tr>
<td>\texttt{strcasecmp} ( s_1, s_2 )</td>
<td>Same as \texttt{strcmp} but ignore case differences</td>
</tr>
<tr>
<td>\texttt{strncasecmp} ( s_1, s_2, n )</td>
<td>Same as \texttt{strncmp} but ignore case differences</td>
</tr>
</tbody>
</table>
• The functions do not compare characters following the NULL character in the strings
• Collating sequences are different in ASCII and EBCDIC

String search functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strchr (s, c)</td>
<td>Returns a pointer to the first occurrence of character c in string s</td>
</tr>
<tr>
<td>strrchr (s, c)</td>
<td>Returns a pointer to the last occurrence of character c in string s</td>
</tr>
<tr>
<td>strpbrk (s1, s2)</td>
<td>Returns a pointer to the first occurrence of character c in string s1</td>
</tr>
<tr>
<td>strspn (s1, s2)</td>
<td>Returns a pointer to the first occurrence of characters in string s1</td>
</tr>
<tr>
<td>strcspn (s1, s2)</td>
<td>Returns a pointer to the first occurrence of characters in string s1</td>
</tr>
<tr>
<td>strstr (s1, s2)</td>
<td>Returns a pointer to the first occurrence of string s2 in string s1</td>
</tr>
<tr>
<td>strtok (s1, s2)</td>
<td>Returns a pointer to the first occurrence of string s2 in string s1</td>
</tr>
</tbody>
</table>

• strchr and strrchr return a pointer to NULL if the character c does not appear in the string s

Memory functions

• Declared in <memory.h> file

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memcpy (s1, s2, n)</td>
<td></td>
</tr>
<tr>
<td>memccpy (s1, s2, c, n)</td>
<td></td>
</tr>
<tr>
<td>memchr (s, c, n)</td>
<td></td>
</tr>
<tr>
<td>memcmp (s1, s2, n)</td>
<td></td>
</tr>
<tr>
<td>memset (s, c, n)</td>
<td></td>
</tr>
</tbody>
</table>