Programming in C

Based on the Original Slides from
Politehnica International- Computer Engineering
Lecture Slides
Outline

• Strings
  – Character Arrays/ Character Strings
  – Initializing Character Strings. The null string.
  – Escape Characters
  – Displaying Character Strings
  – Inputting Character Strings
  – String processing:
    • Testing Strings for Equality
    • Comparing Strings
    • Copying Strings
  – Functions in <string.h>
  – String to number conversion functions
  – Character Strings, Structures, and Arrays
  – Example: Simple dictionary program
    • Sorting the dictionary
    • A better search in sorted arrays
Arrays of characters

- char word[] = { 'H', 'e', 'l', 'l', 'o', '!' };
- To print out the contents of the array word, you run through each element in the array and display it using the %c format characters.
- To do processings of the word (copy, concatenate two words, etc) you need to have the actual length of the character array in a separate variable!
Character strings

- A method for dealing with character arrays without having to worry about precisely how many characters you have stored in them:
- **Placing a special character at the end of every character string.** In this manner, the function can then determine for itself when it has reached the end of a character string after it encounters this special character.
- In the C language, the special character that is used to signal the end of a string is known as the *null* character and is written as '\0'.
- char word[] = { 'H', 'e', 'l', 'l', 'o', '!', '\0' };
Example: string length

// Function to count the number of characters in a string
#include <stdio.h>

int stringLength (char string[]){
    int count = 0;
    while ( string[count] != '\0' )
        ++count;
    return count;
}

text

int main (void) {
    char word1[] = { 'a', 's', 't', 'e', 'r', '\0' };   
    char word2[] = { 'a', 't', '\0' };               
    char word3[] = { 'a', 'w', 'e', '\0' };         
    printf ("%i %i %i\n", stringLength (word1),      
            stringLength (word2), stringLength (word3));
    return 0;
}

Example: const strings

// Function to count the number of characters in a string
#include <stdio.h>

int stringLength (const char string[]){
    int count = 0;
    while ( string[count] != '\0' )
        ++count;
    return count;
}

int main (void) {
    const char word1[] = { 'a', 's', 't', 'e', 'r', '\0' };
    const char word2[] = { 'a', 't', '\0' };
    const char word3[] = { 'a', 'w', 'e', '\0' };
    printf("%i %i %i\n", stringLength (word1),
            stringLength (word2), stringLength (word3));
    return 0;
}
Initializing character strings

- Initializing a string:
  ```c
  char word[] = "Hello!";
  ```

- Is equivalent with:
  ```c
  char word[] = { 'H', 'e', 'l', 'l', 'o', '!', '\0' };  
  ```

- The null string: A character string that contains no characters other than the null character
  ```c
  char empty[] = ""; 
  char buf[100]= "";
  ```

- Initializing a very long string over several lines:
  ```c
  char letters[] = 
  { "abcdefghijklmnopqrstuvwxyz
  \n  ABCDEFGHIJKLMNOPQRSTUVWXYZ" };
  ```

- Adjacent strings are concatenated:
  ```c
  char letters[] = 
  { "abcdefghijklmnopqrstuvwxyz"  
  "ABCDEFGHIJKLMNOPQRSTUVWXYZ" };
  ```

  `printf ("Programming" " in C is fun\n");`
Strings vs Characters

- The string constant "x"
- The character constant 'x'

Differences:
1. 'x' is a basic type (char) but "x" is a derived type, an array of char
2. "x" really consists of two characters, 'x' and '\0', the null character
Escape characters

- \a Audible alert
- \b Backspace
- \f Form feed
- \n Newline
- \r Carriage return
- \t Horizontal tab
- \v Vertical tab
- \ Backslash
- \" Double quotation mark
- \' Single quotation mark
- \? Question mark
- \nnn Octal character value \textit{nnn}
- \unnnnn Universal character name
- \Uunnnnnnnnnn Universal character name
- \xnn Hexadecimal character value \textit{nn}

- the backslash character has a special significance
- other characters can be combined with the backslash character to perform special functions. These are referred to as escape characters.
Examples: Escape characters

```c
printf ("\aSYSTEM SHUT DOWN IN 5 MINUTES!!\n");
printf ("%i\t%i\t%i\n", a, b, c);
printf ("\t is the horizontal tab character.\n");
printf (""Hello,"" he said.\n");
c = '\'';
printf("\033"Hello"\n");
```
Displaying character strings

- Displaying a string: %s
- printf ("%s\n", word);
Inputting character strings

- `char string[81];`
- `scanf ("%s", string);`
- The `scanf` function can be used with the `%s` format characters to read in a string of characters up to a blank space, tab character, or the end of the line, whichever occurs first.
- Note that unlike previous `scanf` calls, in the case of reading strings, the `&` is *not* placed before the array name!
- If you type in more than 80 consecutive characters to the preceding program without pressing the spacebar, the tab key, or the Enter (or Return) key, `scanf` overflows the character array!
- `scanf` has no way of knowing how large its character array parameters are! When handed a `%s` format, it simply continues to read and store characters until one of the noted terminator characters is reached.
- If you place a number after the `%` in the `scanf` format string, this tells `scanf` the maximum number of characters to read.
- Correct use of `scanf` for reading strings:
  - `scanf ("%80s", string);`
Example: string processing

```c
#include <stdio.h>

void concat (char result[],
        const char str1[], const char str2[]);

int main (void)
{
    const char s1[] = "Test ";
    const char s2[] = "works." ;
    char s3[20];
    concat (s3, s1, s2);
    printf ("%s\n", s3);
    return 0;
}
```
Example: concatenate strings

// Function to concatenate two character strings
void concat (char result[], const char str1[], const char str2[])
{
    int i, j;
    // copy str1 to result
    for ( i = 0; str1[i] != '\0'; ++i )
        result[i] = str1[i];
    // copy str2 to result
    for ( j = 0; str2[j] != '\0'; ++j )
        result[i + j] = str2[j];
    // Terminate the concatenated string with a null character
    result[i + j] = '\0';
}
Testing strings for equality

```cpp
bool equalStrings (const char s1[], const char s2[]) {
    int i = 0;
    bool areEqual;
    while ( s1[i] == s2[i] && s1[i] != '\0' && s2[i] != '\0' )
        ++i;
    if ( s1[i] == '\0' && s2[i] == '\0' )
        areEqual = true;
    else
        areEqual = false;
    return areEqual;
}

!!! NOT:  s1==s2
```
Alphabetically comparing strings

// Function to compare two character strings
int compareStrings (const char s1[], const char s2[])
{
    int i = 0, answer;
    while ( s1[i] == s2[i] && s1[i] != '\0' && s2[i] != '\0' )
        ++i;
    if ( s1[i] < s2[i] )
        answer = -1; /* s1 < s2 */
    else if ( s1[i] == s2[i] )
        answer = 0; /* s1 == s2 */
    else
        answer = 1; /* s1 > s2 */
    return answer;
}

!!! NOT: s1 < s2
!!! NOT: s1 > s2
Copying strings

```c
void copyString(char dest[], char srs[]) {
    int i;
    i=0;
    while ((dest[i] = srs[i]) != '\0')
        i++;
}

!!! NOT: dest = srs
```
String functions

- The C library supplies several string-handling functions; you don’t have to re-write them from scratch!
- ANSI C uses the `<string.h>` header file to provide the prototypes.
- Most frequently used functions: `strlen()`, `strcat()`, `strncat()`, `strcmp()`, `strncmp()`, `strcpy()`, and `strncpy()`.

- `#include <string.h>`
- `strcat (s1, s2)`
  - Concatenates the character string `s2` to the end of `s1`, placing a null character at the end of the final string. The function also returns `s1`.
- `strcmp (s1, s2)`
  - Compares strings `s1` and `s2` and returns a value less than zero if `s1` is less than `s2`, equal to zero if `s1` is equal to `s2`, and greater than zero if `s1` is greater than `s2`.
- `strcpy (s1, s2)`
  - Copies the string `s2` to `s1`, also returning `s1`.
- `strlen (s)`
  - Returns the number of characters in `s`, excluding the null character.
String functions (cont.)

- `strncat (s1, s2, n)`
  - Copies `s2` to the end of `s1` until either the null character is reached or `n` characters have been copied, whichever occurs first. Returns `s1`.
- `strncmp (s1, s2, n)`
  - Performs the same function as `strcmp`, except that at most `n` characters from the strings are compared.
- `strncpy (s1, s2, n)`
  - Copies `s2` to `s1` until either the null character is reached or `n` characters have been copied, whichever occurs first. Returns `s1`.
- `strchr (s, c)`
  - Searches the string `s` for the last occurrence of the character `c`. If found, a pointer to the character in `s` is returned; otherwise, the null pointer is returned.
- `strstr (s1, s2)`
  - Searches the string `s1` for the first occurrence of the string `s2`. If found, a pointer to the start of where `s2` is located inside `s1` is returned; otherwise, if `s2` is not located inside `s1`, the null pointer is returned.
Example: String functions

```c
#include <stdio.h>

#define PRAISE "What a super marvelous name!"

#include <string.h> /* provides strlen() prototype */

#define PRAISE "What a super marvelous name!"

int main(void) {
    char name[40];
    printf("What's your name?\n");
    scanf("%39s", name);
    printf("Hello, %s. %s\n", name, PRAISE);
    printf("Your name of %d letters occupies %d memory\n", strlen(name), sizeof name);
    return 0;
}
```
#include <stdio.h>

#include <string.h>

int main(void) {
    char string1[] = "this is";
    char string2[] = "a test";
    char string3[20] = "Hello, ";
    char string4[] = "world!";
    printf("%s\n", string3);
    strcat(string3, string4);
    printf("%s\n", string3);
    if(strcmp(string1, string2) == 0)
        printf("strings are equal\n");
    else printf("strings are different\n");
    return 0;
}
String to number conversions

- Storing a number as a string means storing the digit characters.
- Example: the number 213 can be stored in a character string array as the digits '2', '1', '3', '\0'. Storing 213 in numeric form means storing it as an int.
- C requires numeric forms for numeric operations, such as addition and comparison, but displaying numbers on your screen requires a string form because a screen displays characters. The printf() and scanf() functions, through their %d and other specifiers, convert numeric forms to string forms, and vice versa.
- C also has functions whose sole purpose is to convert string forms to numeric forms.
String to number conversion functions

- `<stdlib.h>`
- `atoi()` converts string `s` to a type `int` value and returns it. The function converts characters until it encounters something that is not part of an integer.
- `atof()` converts a string to a type `double` value and returns it
- `atol()` converts a string to a type `long` value and returns it

```c
// Using atoi
#include <stdio.h>
#include <stdlib.h>
int main (void) {
    printf ("%i\n", atoi("245"));
    printf ("%i\n", atoi("100") + 25);
    printf ("%i\n", atoi("13x5"));
    return 0;
}
```
String to number conversion
Do-it-yourself exercise

// Function to convert a string to an integer – my atoi
#include <stdio.h>
int strToInt (const char string[]) {
    int i, intValue, result = 0;
    for ( i = 0; string[i] >= '0' && string[i] <= '9'; ++i )
    {
        intValue = string[i] - '0';
        result = result * 10 + intValue;
    }
    return result;
}

int main (void) {
    printf ("%i\n", strToInt("245"));
    printf ("%i\n", strToInt("100") + 25);
    printf ("%i\n", strToInt("13x5"));
    return 0;
}
Example: readLine

// Function to read a line of text from the terminal
void readLine (char buffer[])
{
    char character;
    int i = 0;
    do
    {
        character = getchar ();
        buffer[i] = character;
        ++i;
    }
    while ( character != '
' );
    buffer[i - 1] = '\0';
}
Example continued

```c
#include <stdio.h>

void readLine (char buffer[]);

int main (void)
{
    int i;
    char line[81];
    for ( i = 0; i < 3; ++i )
    {
        readLine (line);
        printf (%s

n
n, line);
    }
    return 0;
}
```
Example: array of structures
Example: a dictionary program

```c
struct entry
{
    char word[15];
    char definition[50];
};

struct entry dictionary[100] =
{
    { "aardvark", "a burrowing African mammal" },
    { "abyss", "a bottomless pit" },
    { "acumen", "mentally sharp; keen" },
    { "addle", "to become confused" },
    { "aerie", "a high nest" },
    { "affix", "to append; attach" },
    { "agar", "a jelly made from seaweed" },
    { "ahoy", "a nautical call of greeting" },
    { "aigrette", "an ornamental cluster of feathers" },
    { "ajar", "partially opened" } 
};
```
Example: dictionary continued

```c
int lookup (const struct entry dictionary[],
            const char search[], const int entries);

int main (void)
{
    char word[10];
    int entries = 10;
    int entry;
    printf ("Enter word: ");
    scanf ("%14s", word);
    entry = lookup (dictionary, word, entries);
    if ( entry != -1 )
        printf ("%s\n", dictionary[entry].definition);
    else
        printf ("The word %s is not in my dictionary.\n", word);
    return 0;
}
```
Searching in array

#include <string.h>

// function to look up a word inside a dictionary
int lookup (const struct entry dictionary[],
            const char search[],
            const int entries)
{
    int i;
    for (i = 0; i < entries; ++i)
        if (strcmp(search, dictionary[i].word))
            return i;
    return -1;
}
Binary search

- Binary Search Algorithm
- Search x in **SORTED** array M
- **Step 1:** Set low to 0, high to n – 1.
- **Step 2:** If low > high, x does not exist in M and the algorithm terminates.
- **Step 3:** Set mid to \((low + high) / 2.\)
- **Step 4:** If M[mid] < x, set low to mid + 1 and go to step 2.
- **Step 5:** If M[mid] > x, set high to mid – 1 and go to step 2.
- **Step 6:** M[mid] equals x and the algorithm terminates.
Binary search

// Function to look up a word inside a dictionary
int lookup (const struct entry dictionary[],
           const char search[], const int entries)
{
    int low = 0;
    int high = entries - 1;
    int mid, result;
    while ( low <= high )
    {
        mid = (low + high) / 2;
        result = strcmp (dictionary[mid].word, search);
        if ( result == -1 )
            low = mid + 1;
        else if ( result == 1 )
            high = mid - 1;
        else
            return mid; /* found it */
    }
    return -1; /* not found */
}