CS1010E: Programming Methodology

Tutorial 10: String

10 Mar 2017 - 15 Mar 2017

1. Discussion Questions

(a) [String] What is the *closest* function in `#define <string.h>` corresponding to the functions below?

i.
```
int foo(char *str) {
  int i;
  for(i=0; str[i]!='\0'; i++);
  return i;
}
```
i. finding the length (strlen)

ii.
```
int foo(char *strA, char *strB) {
  int i;
  for(i=0; strA[i]!='\0' && strB[i]!=0 && strA[i]==strb[i]; i++);
  return strA[i] - strB[i];
}
```
ii. lexicographical comparison (strcmp)

iii.
```
void foo(char *strA, char *strB) {
  for(; *strB!='\0'; strA++, strB++) *strA = *strB;
  *strA = '\0';
}
```
iii. copy (strcpy)

iv.
```
void foo(char *strA, char *strB) {
  for(; *strA!='\0'; strA++);
  for(; *strB!='\0'; strA++, strB++) *strA = *strB;
  *strA = '\0';
}
```
iv. concatenation (strcat)

v.
```
bool foo(char *strA, char c) {
  for(; *strA!='\0' && *strA!=c; strA++);
  return *strA==c; /* return strA */
}
```
v. finding character (modified strchr)

vi.
```
bool bar(char *strA, char *strB) {
  for(; *strA!='\0' && *strB!='\0' && *strA==*strB; strA++, strB++);
  return *strB=='\0';
}
bool foo(char *strA, char *strB) {
  for(; *strA!='\0'; strA++)
    if(bar(strA, strB)) return true; /* return strA */
  return false; /* return strA */
}
```
vi. finding string (modified strstr)

2. Program Analysis

(a) [String Reasoning] What is/are the output of code fragments below?

i.
```
bool is_alpha(char c) { return (c>='a'&&c<='z')||(c>='A'&&c<='Z'); }
int main(void) {
  char word[] = "Hello World", *ptr = word;
  while(*ptr != '\0') {
    if(is_alpha(*ptr) && !is_alpha(*(ptr+1)))
      *ptr = '\0';
    ptr++;
  }
  printf("%s", &word[6]);
}
```

i. Worl (search end-of-word)

ii.
```
char word[] = "cs1010e", *ptr = word; int S = 0, i;
for(i=0; word[i] != '\0'; i++) {
  if(word[i] >= '0' && word[i] <= '9') {
    printf("%c%c%c ", word[i], *(word+i), *(i+word));
    S += word[i] - '0';
  }
}
printf("%d", S);
```

ii. 111 000 111 000 2 (char to int conversion)

iii.
```
int foo(char *W) {
  int s=-1, e=strlen(W);
  while(e-->s++)
    if(W[s] != W[e])
      return false;
  return true;
}
int main() {
  char W1[] = "aaa", W2[] = "aba", W3[] = "abc";
  printf("%d %d %d", foo(W1), foo(W2), foo(W3));
  return 0;
}
```

iii. 1 1 0 (palindrome check)

(b) [Complex String Reasoning] What is/are the output of code fragments below?

i.
```
char W[4][6] = {"Brown", "Fox", "Quick", "The"}, R[20] = "";
int  I[4]    = {1, -1, 0, 2}, i = 3;
while(I[i]!=-1) {
  strcat(R, W[i]);
  i = I[i];
}
printf("%s", R);
```

i. TheQuickBrown

ii.
```c
int E(int n1[], int n2[]) {
  int i;
  for(i=0; i<26; i++)
    if(n1[i]!=n2[i])
      return 0;
  return 1;
}
int A(char *s1, char *s2) {
  int n1[26] = {0}, n2[26] = {0};
  while(*s1!='\0') {
    if(*s1!=' ') {
      n1[(*s1)-'a']++;
    }
    s1++;
  }
  while(*s2!='\0') {
    if(*s2!=' ') {
      n2[(*s2)-'a']++;
    }
    s2++;
  }
  return E(n1, n2);
}
int main() {
  char s11[] = "i am a weakish speller";
  char s12[] = "william shakespeare";
  char s21[] = "madam curie";
  char s22[] = "Radium came";
  printf("%d %d", A(s11,s12), A(s21,s22));
}
```

ii. `1 0 (anagram check)`

3. Designing a Solution

(a) [String] Email address is composed of **two (2)** sub-components: the username and domain. The format of an email address is of the form username@domain where domain usually consists of at least a single dot (.) but must not end with it. The summary of the valid email address is:

- Can be *decomposed* into **two (2)** components: username and domain
- username must satisfy the following criteria:
  - It must be at least 8 characters long
- domain must satisfy the following criteria:
  - It must contain at least **one (1)** dot (.) character
  - It must **NOT** end with a dot (.) character
  - It must **NOT** contain any at (@) character
  - It must be at least 5 characters long

In this problem, we will try to make a simple program to check for valid email address.

i. Write a code to split a potential email address into its username and domain where the domain is considered only from the **first** at (@) character. Assume that username and domain contains enough space to store all the characters.

```
void split(char *email, char *username, char *domain) {
  /* Split email into username and domain */
  int idx=0;
  while(email[idx]!='\0' && email[idx] != '@') idx++;
  if(email[idx] != '\0') {
    email[idx] = '\0';
    strcpy(email      ,username);
    strcpy(email+idx+1,domain  );
    email[idx] = '@';
  }

}
```

ii. Write a code to check if the potential domain contains at least **one (1)** dot (.) and it is not at the end.

```
bool check_dot(char *domain) {
  /* Check dot correctness */
  int idx=0, dot=0;
  while(domain[idx]!='\0') {
    if(domain[idx]=='.') dot++;
    idx++;
  }
  return idx>0 && domain[idx-1]!='.' && dot>0;

}
```

iii. Write a code to check if the potential domain does not contain at (@) character.

```
bool check_at(char *domain) {
  /* Check dot correctness */
  int idx=0, dot=0;
  while(domain[idx]!='\0') {
    if(domain[idx]=='@') return false;
    idx++;
  }
  return true;

}
```

iv. Write a code to check if the potential email address is a valid email address based on the above criteria. Assume that the maximum number of character in email is given as SIZE.

```
bool is_valid_email(char *email) {
  /* Check dot correctness */
  char username[SIZE+1]="", domain[SIZE+1]="";
  split(email, username, domain);
  return check_dot(domain) && check_at(domain)
      && strlen(username)>=8 && strlen(domain)>=5;

}
```

4. Challenge

(a) [String] A maze is described as 2D array of walls and empty space with **two (2)** such space designated as an entrance and an exit. In our representation, the walls are represented as `'X'`, space as `'_'`, entrance as `'S'`, and exit as `'E'`. The question is to find out whether there is a path from the entrance to the exit given that you can only move in the **four (4)** cardinal direction (up, down, left, and right). For example, the map below has the path to exit of length 21. The path is marked with `'.'`.

```
XXXXXXXXXXX  XXXXXXXXXXX
X_____X__XX  X.....X__XX
S_X_X_X_X_E  S.X_X.X_X.E
X_X_X_X_X_X  X_X_X.X_X.X
X_XX__X_X_X  X_XX..X_X.X
X__X_X____X  X__X.X....X
XX_X___X_XX  XX_X...X_XX
XXXXXXXXXXX  XXXXXXXXXXX
```

You are to implement a function to check for path from entrance to exit. The function accepts as input an array of **string** representing the map. The element of the array is a single row of the map.

```
bool path(char maze[][SIZE], int n) {
  int si, sj; /* index of entrance to be searched */
  /* given a maze and number of rows n, find there is a path from entrance to exit */
  int i, j, move=0, m=strlen(maze[0]);
  for(i=0; i<n; i++) /* search start and exit */
    for(j=0; j<m; j++)
      if(maze[i][j]=='S') { si=i; sj=j; }
  if(sj+1>=0 && sj+1<m && maze[si][sj+1]=='_') maze[si][sj+1]='.'; /* down  */
  if(sj-1>=0 && sj-1<m && maze[si][sj-1]=='_') maze[si][sj-1]='.'; /* up    */
  if(si+1>=0 && si+1<n && maze[si-1][sj]=='_') maze[si-1][sj]='.'; /* left  */
  if(si-1>=0 && si-1<n && maze[si+1][sj]=='_') maze[si+1][sj]='.'; /* right */
  do {
    move = 0;
    for(i=0; i<n; i++) { /* marking */
      for(j=0; maze[i][j]!='0'; j++) {
        if(maze[i][j]=='.') {
          if(j+1>=0 && j+1<m && maze[i][j+1]=='_') maze[i][j+1]='.'; /* down  */
          if(j-1>=0 && j-1<m && maze[i][j-1]=='_') maze[i][j-1]='.'; /* up    */
          if(i+1>=0 && i+1<n && maze[i-1][j]=='_') maze[i-1][j]='.'; /* left  */
          if(i-1>=0 && i-1<n && maze[i+1][j]=='_') maze[i+1][j]='.'; /* right */
          if(j+1>=0 && j+1<m && maze[i][j+1]=='E') return true;
          if(j-1>=0 && j-1<m && maze[i][j-1]=='E') return true;
          if(i+1>=0 && i+1<n && maze[i-1][j]=='E') return true;
          if(i-1>=0 && i-1<n && maze[i+1][j]=='E') return true;
          maze[i][j]='+'; move=1; /* done marking */
        }
      }
    }
  } while(move==1);
  return false;
}
```