Tutorial –5B Solution

Objectives: To practise with `C – Strings`

1. Given the declaration

```
char partOne[40]="BT137";
char partTwo[40]="WT105";
```

a. Write an `if` statement that compares two strings and outputs content of the string with the greatest value.

```
if( strcmp(partOne, partTwo) > 0 )
    printf("%s", partOne);
else
    printf("%s", partTwo);
```

b. Copies content of `partOne` into `partTwo`.

```
strcpy( partTwo, partOne );
```

c. Checks that the `partTwo` array size is sufficient to store the content of `partOne` String.

```
if( sizeof(partTwo) > strlen(partOne) )   . . . ;
```

Recall the function strlen does not include the null character in its count. The header <string.h> should be declared to use C-string Built-in functions

2. Write a function that pads a variable-length string with blanks to its maximum size. For example, if s10 is a ten-character array currently holding the string "screen", `blank_pad` would add three blanks (one of which would overwrite the null character) and finish the string with the null character. Be sure your function would work if no blank padding were necessary.

Answer:

```
/*
 *  Adds blanks to the end of s until s is its full declared
    size.
 */


void blank_pad(char s[],int   size)
```

```
    {
       int t;
       for (t = strlen(s);  t < size - 1;  ++t)
          s[t] = ' ';

       s[size-1] = '\0';
    }
```

3. Write a program that takes a word less than 25 characters long and prints a statement like this:

`fractal starts with the letter f`

Have the program process words continually until it encounters a "word" beginning with the character `'9'`.

```
#include <stdio.h>
int main(void)
{
   char in[25];
   scanf("%24s", in);
   while(in[0]!='9'){
        printf("%s starts with the letter %c\n", in, in[0]);
       scanf("%24s", in);
   }
   return (0);
}
```

OR using for loop

```
#include <stdio.h>

int main(void)
{
   char in[25];

   for (scanf("%24s", in);in[0] != '9';scanf("%24s", in))
     printf("%s starts with the letter %c\n", in, in[0]);

   return (0);
}
```

4. Given these declarations,

```
char socsec[12] = "123-45-6789";
char ssnshort[7], ssn1[4], ssn2[3], ssn3[5];
```

write statements to accomplish the following:
a. Store in `ssnshort` as much of `socsec` as will fit.
b. Store in `ssn1` the first three characters of `socsec`.
c. Store in `ssn2` the middle two-digit portion of `socsec`.

d. Store in `ssn3` the final four digits of `socsec`.

Be sure your statements store valid strings in each variable.

```
a.  strncpy(ssnshort, socsec, 6);
    ssnshort[6] = '\0';
```

we should not use strcpy as ssnshort is shorter than socsec

```
b.  strncpy(ssn1, socsec, 3);
    ssn1[3] = '\0';

c.  strncpy(ssn2, &socsec[4], 2);
    ssn2[2] = '\0';
```

That's because **"123-45-6789"; are at relative address offset of 4 and 5.**

We could also have written

```
strncpy(ssn2, socsec+4, 2);
    ssn2[2] = '\0';
```

```
d.  strcpy(ssn3, &socsec[7]);
```

The function strcpy automatically add a null pointer

5. **Given the string `pres` (value is "Adams, John Quincy"). There is an error in the last line of the following code fragment. What is the error? Why is it wrong? How would you correctly achieve the intent of this call?**

```
strcpy(tmp1, &pres[12]);
strcat(tmp1, " ");
strcat(tmp1, pres[7]);
```

Answer:

The second argument to strcat (pres[7]) is a character rather than a string.

```
strncat(tmp1, &pres[7], 1);

Value of tmp1:
QUINCY J
```

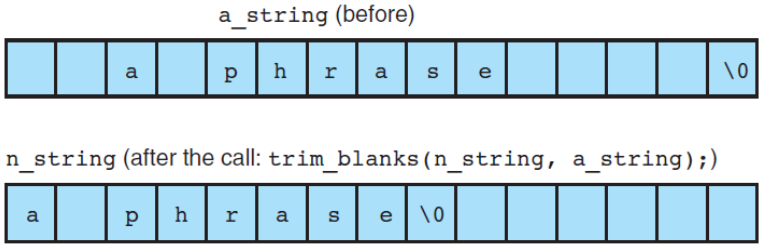
6. **Write a function `bracket_by_len` that takes a word as an input argument and returns the word bracketed to indicate implicitly its length. Words less than five characters long are bracketed with << >> , words five to ten letters long are bracketed with (* *) , and words over ten characters long are bracketed with /+ +/ . Your function should require the calling function to provide as the first argument, space for the result, and as the third argument, the amount of space available.**

```
/*
 * Brackets a string according to its length.
```

```
 *          Length          Brackets used
 *          < 5 chars          <<   >>
 *          5 - 10 chars       (*   *)
 *          > 10 chars         /+   +/
 */
void
bracket_by_len(char       result[],/* output */
               const char word[], /* input - string to bracket */
               int        size)
    {
       int len;
       len = strlen(word);
       if (len + 5 > size) {   // 5 to count for 2 brackets+null chars
          printf("Insufficient space\n");
       } else if (len < 5) {
          strcpy(result, "<<");
          strcat(result, word);
          strcat(result, ">>");
       } else if (len < 11) {
          strcpy(result, "(*");
          strcat(result, word);
          strcat(result, "*)");
       } else {
          strcpy(result, "/+");
          strcat(result, word);
          strcat(result, "+/");
       }
       return (result);
    }
```

Check lecture 5b for how strcpy,strcat process the null character

7. **Complete function `trim_blanks` whose purpose is to take a single string input parameter (`to_trim`) and return a copy of the string with leading and trailing blanks removed. Use `strncpy` in `trim_blanks` .**



Answer:

Design solution:

```
void trim_blanks(char trimmed [], const char to_trim[])
{
/* Find subscript/index of first nonblank in to_trim */
/* Find subscript/index of last nonblank in to_trim */
/* Use strncpy to store the nonblank chars in trimmed */
}
```

Code:

```
void trim_blanks (char trimmed [], const char to_trim [])
{
    int first, last;
    // leading blanks
    first=0;
    while(to_trim[first] == ' ')
    first++;
     // trailing blanks
    last = strlen(to_trim)- 1; // given array first index is 0
    while(to_trim[last] == ' ')
    last--;
    // copy last-first+1 chars
    strncpy(trimmed, &to_trim[first], last - first + 1);
  // we need to add a null since source above is longer than
  // num see notes
    trimmed[last - first + 1] = '\0';
     printf("%s", trimmed);
}
```

The same algorithm using for loop is as follows:

```
void trim_blanks (char trimmed [], const char to_trim [])
{
 int first, last;
for (first = 0; to_trim[first] == ' '; ++first) {}
for (last = strlen(to_trim)- 1; to_trim[last] == ' ';--last) {}
strncpy(trimmed, &to_trim[first], last - first + 1);
trimmed[last - first + 1] = '\0'; // must be added
 printf("%s", trimmed);
}
```