Michael Campbell

X52.9008 – 01

Tues 6:30-9:00PM

July 20, 2010

--Final—

1. Write a program containing a concatenate function, **char\* s1 string\_cat(char \*s1, char \* s2),** that appends **string2** to **string1**. **(40%)**

* **string1** should be declared as **char s1[80]** and **string2** should be declared as **s2[40].**
* The function needs to check that the **sum of the lengths** of both strings is less than 80 characters before appending s2 to s1.
* In the event that the sum of the two strings is greater than 80, the function should return NULL otherwise it returns the &s1[0].
* Within the function call the user should print the sum of the total length of the final concatenated string. When appending s2 to s1, make sure to remove the NULL termination character in s1 before proceeding to append s2.
* The input strings should be collected sequentially from the keyboard using **fgets(&s1[0], len1 + 1, stdin) and fgets(&s2[0], len2 + 1, stdin)** where, **len1** and **len2** are the maximum size of the respective input strings**.**
* The program should query the user for input strings and print the initial strings(s1 and s2) and the final concatenated string.
* The **queering loop** should continue until the user says **No(0).** For this problem you can use **strlen(char \*)** function defined in <**string.h**> but not any other function in that libarary!!!
* Run the program several times to demonstrate that it works for all cases.

/\* Program: concatenate.c

Author: Michael Campbell

Date: 7/20/10

Synopsis: concatenate function that appends string2 to string1.

- it checks that the sum of the lengths of both strings

is less than 80 characters before appending

- prints the sum of the total length of the final

concatenated string

- strings are collected sequentially from keyboard -->

continue gathering strings until user says no

\*/

#include <stdio.h>

#include <string.h>

#define No 0

int main (void) {

char s1[80];

char s2[40] = "";

char loop [4];

int len1 = 80;

int len2 = 40;

fflush(stdin);

//first string gathered from keyboard

printf("\nInput First String\n");

\*s1 = fgets(&s1[0] + 1, len1, stdin);

//second string gathered from keyboard

printf("\nInput Second String\n");

\*s2 = fgets(&s2[0] + 1, len2, stdin);

printf("\nstring 1 = %s\n", s1);

printf("string 2 = %s\n", s2);

char string\_cat(char \* s1, char \* s2);

string\_cat(s1, s2);

printf("New String = %s\n", s1);

fflush(stdin);

//starts loop again

printf("Start loop again?\n");

scanf("%s",loop);

if (strlen(loop) == 3 && loop == "yes")

goto start;

else if (strlen(loop) == 2 && loop == "no")

return 0;

// printf("string length = %i", strlen(loop));

// printf("%s", loop);

start:

return 0;

}

//Concatenatation function

char string\_cat(char \* s1, char \* s2)

{

int i;

int s1length = 0;

int s2length = 0;

for (s1length = 0; s1[s1length]; ++s1length);

for (s2length = 0; s2[s2length]; ++s2length);

//if statements to determine if strings can be joined

if ((s1length + s2length) < 80)

for (i = 0; i < s2length; i++){

s1[s1length - 1] = ' ';

s2[0] = ' ';

s1[(s1length - 1) + i] = s2[i];

}

else if ((s1length + s2length) > 80)

;

else

return &s1[0];

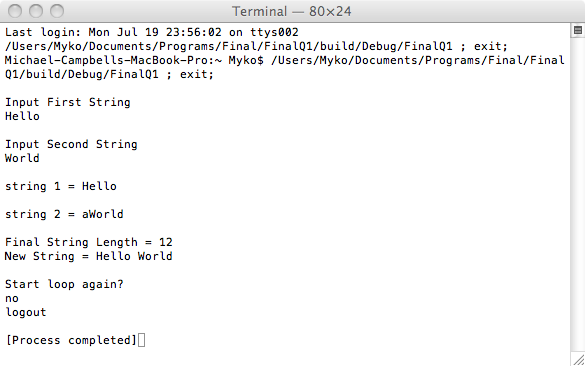
s1[(s1length + i) - 1] = '\0';

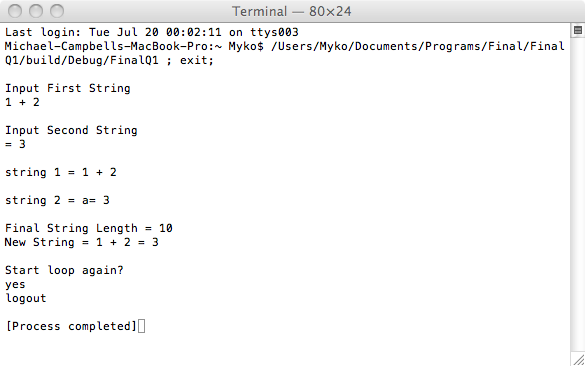
//prints the sum of the total length of the final concatenated string

printf("Final String Length = %i\n", ((s1length + i) - 2));

}

Output:





1. Write a program that **(40%):**

* Collects sequentially lines of text (phrases) from the keyboard using f**gets(&buffer[0], len1 + 1, stdin).**
* The program should iteratively ask the user for a new text line.
* Control out of the query mode should be possible through the input of the **# character** or after **exceeding 40 lines** of text.
* The lines of text should be stored on the **heap**, through the use of the **malloc**( #bytes) function call.
* The correct number of bytes on the heap required for each line should be obtained through a **string length** function call ( **strlen(char \*)** ).
* The pointer returned from malloc( ) should be stored sequentially in a ***character pointer*** array, **line\_array[40].**
* After finishing collecting all the lines of text, the program should print all the input text lines
* Delete the 5th and 6th line of text and move them to the end of the document
* After moving the 5th and 6th line, all lines of text should be printed again
* Finally, the lines of text should be sorted alphabetically and printed again to demonstrate that the lines have been correctly sorted (use the ***sort function*** that was discussed in class o accomplish this step)

/\*

Program: linesOfText.c

Author: Michael Campbell

Date: 7/20/10

Synopsis: this program sequentially collects lines of text from the keyboard using the fgets function, asking the

user for a new text line.

- the query can be exited through the input of the # character or after exceeding 40 lines of text

- lines of text are stored in the heap through malloc

- the number of bytes needed for each line should be obtained through the string length function

- the pointer returned from malloc should be stored in a character pointer array

- after collecting all the lines, the program should print them all

- delete the 5th and 6th lines of text and move them to the end of the document,

after moving the 5th and 6th lines all lines of text should be printed again.

- the lines of text should be sorted alphabetically and printed again.

\*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

int main (void) {

char buffer[80];

int len = 80;

int i;

int \*line\_array[80];

char strings[80];

int stringlength = 0;

// int strlencount = 0;

stringlength = strlen(&strings[i]);

// do {

// printf ("\nPlease enter the next line of text\n");

// fgets(&buffer[0] + 1, len, stdin);

// }

// while (i < 40 && buffer[i] != '#'); {

// i++;

// }

for (i = 0; i < 40 || buffer[i] == '#'; i++){

printf ("\nPlease enter the next line of text\n");

strings[i] = fgets(&buffer[0] + stringlength +i, len, stdin);

// printf("string was: %s", &buffer[i]);

line\_array[i] = (int \*)malloc (stringlength \* sizeof(strings[i]));

if (buffer[i] == '#')

// for (i; i = 0; i--) {

// strlencount++;

goto exit\_loop;

}

// }

exit\_loop:

line\_array[i] = (int \*)malloc (stringlength \* sizeof(char));

//print all lines of text

printf("The strings were: %s\n\n", &buffer[0]);

// move 5th and 6th lines of text to the end of document

line\_array[4] = line\_array[i + 1];

line\_array[4] = '\0';

line\_array[5] = line\_array[i + 1];

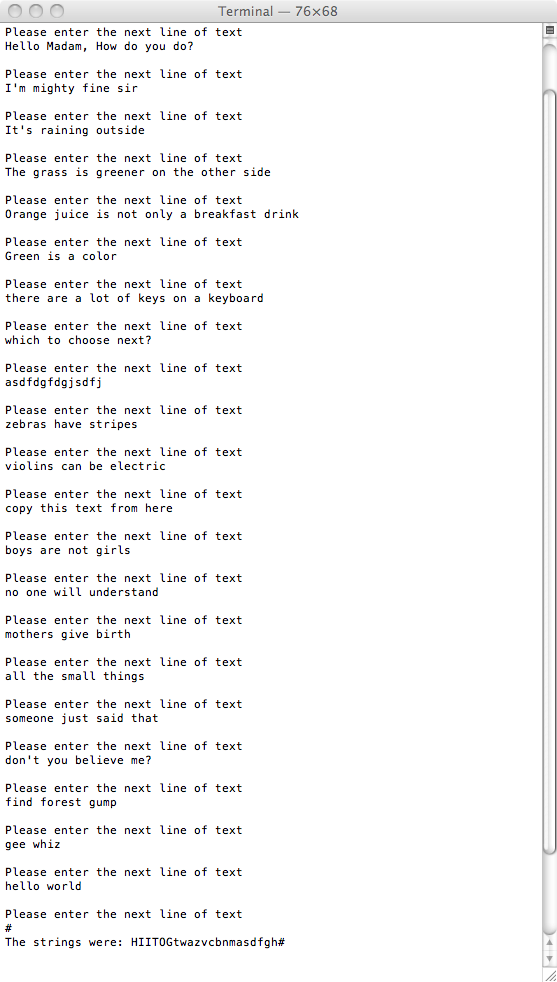
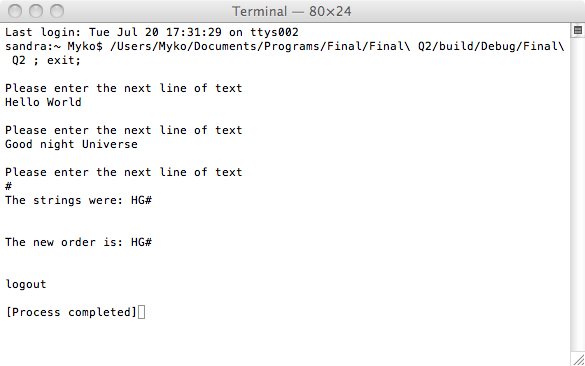
line\_array[5] = '\0';

printf("The new order is: %s\n\n", &buffer[0]);

return 0;

}

Output:

3. Write a program **(20%),** **Currency**, that accepts arguments from the command line (argc, \*argv[ ])as follows:

* **Currency EU 100**
* **Currency BP 50**
* **Currency CD 1000**
* **Currency CY 125**
* The above program will convert the input ***currency into US dollars***.
* For instance the first entry will take 100 euros and convert it into 1.4\*100 = $140
* The second entry will convert 50 British Pounds and convert into 1.5\*50 = $75
* The third entry will convert the 75 Canadian dollars into 0.99\*1000 = $999
* The fourth entry will convert 125 Chinese Yuan into 8\*125 = $1000

**The program should provide error checking for wrong currencies.**

/\* Program: currency.c

Author: Michael Campbell

Date: 7/20/10

Synopsis: program accepts arguments from the command line and

converts foreign currency into US dollars.

\*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

int main (int argc, const char \* argv[])

{

int result = 0;

int value = 0;

// char c[2];

value = atoi(argv[2]);

printf("value = %i\n", value);

if (argc != 3){

fprintf(stderr, "Need country code and amount\n");

return 1;

}

/\*

if (argv[1] == "EU")

c[0] = 'E';

else if (argv[1] == "BP")

c[0] = 'B';

else if (argv[1] == "CD")

c[0] = 'D';

else if (argv[1] == "CY")

c[0] = 'Y';

\*/

// printf("c[0] = %s", c[0]);

printf("argv[1] = %s", argv[1]);

switch (\*argv[1]) {

case 'E':

result = value \* 1.4;

break;

case 'B':

result = value \* 1.5;

break;

case 'D':

result = value \* 0.99;

break;

case 'Y':

result = value \* 8;

default:

break;

}

printf("argv = %s\n", argv[2]);

printf("\nResult = %i\n", result);

return 0;

}