SYSTEM SKILL MIDTERM QUIZ

Pattapon Songpetchmongkol 6380830

---

1. LINUX SCRIPTING

a.

git clone -b pattapon https://github.com/rausavar/MUIC-s2021-t3-syskill

To clone a repository from git, we need to use the command git clone <url>. If we wanted to fork a branch from the masters folder and name it, we would use -b command in order to signify a branch and then add a name afterwards.

b.

grep -rnw ./assignmentX -e myMalloc

-r means recursive so it goes through every subdirectory as well
-n is a function of grep that will display the line number of the query inputted for a given file or folder.
-w is used to match the whole word
-e is the pattern of characters that we are searching for

c.

1. git revert adbe182fe
2. git push -u origin [branch]

git commit adbe182fe: will revert back to the commit with the code ‘adbe182fe’
git push -u origin master: will push the correct new commit to a specified branch

d.

a.b.*

This will match with anything that has the letter a (case sensitive), followed by any character, followed by the letter b (case sensitive), followed by any character, and * represents the previous character that could be repeated 0 times or an unlimited number of times afterwards.

e.

^[1-9]\d*$ // excluding 0 if 0 is not considered a positive integer

^ signifies the start of the regex line
[1-9] indicates that we want a positive integer; one without a negative sign in front, this excludes 0 if we are assuming 0 is not a positive integer
\d is any digit from [1-9]
* matches \d from 0 to unlimited number of times

f.

1. pwd

2. cd ../.. OR cd /
3. find /home -type f -name “content” 2>/dev/null
4. cd ./home/u6XXXXXX/hi
5. cat content
6. repeat 4 and 5 (4 more times) for the other “content” files in other users directory

pwd: to find the path of the directory I am in, in the server
cd ../..: to go back to the home directory
find /home -type f -name “content” 2>/dev/null:

- find: search function
- /home: folder that we are searching through
- -type f: we are searching for a file type and want the output to only return file types
- -name “content”: search specifically for the files that contain the name “content”
- 2>/dev/null: to make sure that we don’t return unsuccessful attempts; only successful attempts

cd ./home/u6XXXXXX/hi: take us to the directory with the file “content”
cat content: to allow us to see what is inside the content file

2. BASIC C

a. The actual isFib.c file is in the zipped folder.

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

int isFib(int n){
   int i=0;
   int first = 1;
   int second = 2;
   int third;
   while (i<n){
       third = first + second;
       first = second;
       second = third;
       if (n == first || n == second || n == third || n == 1){
           return 1;
           break;
       } else if (third > n){
           return 0;
       }
       i++;
   }
```

```
    return 0;
}

int main(){
    printf("%d\n", isFib(1));
    printf("%d\n", isFib(2));
    printf("%d\n", isFib(3));
    printf("%d\n", isFib(4));
    printf("%d\n", isFib(5));
    printf("%d\n", isFib(20));
    printf("%d\n", isFib(99));
}
```

b. The matrixVector.c file is in the zipped folder.

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

void matrixVector(int **inMatrix, int *inVector, int * result, int
sizeX, int sizeY){
    for (int i=0;i<sizeX;i++){
        for (int j=0;j<sizeY;j++){
            result[i]+=inMatrix[i][j]*inVector[j];
        }
    }
}

int main(){
    int ** a;
    int * b;
    int * c;
    a = malloc(sizeof(int*)*3);
    b = malloc(sizeof(int)*3);
    c = malloc(sizeof(int)*3);
    for(int i=0;i<3;i++)
    {
        *(a+i) = malloc(sizeof(int)*3);
        for(int j=0;j<3;j++)
        {
            *(*(a+i)+j) = i+j;
```

```
        }
        *(b+i) = i;
    }
    printf("a: \n{\n");
    for(int i = 0;i<3;i++)
    {
        for(int j=0;j<3;j++)
            printf("%d, ", a[i][j]);
        printf("\n");
    }
    printf("}\n");
    printf("b: \n{ ");
    for(int i = 0;i<3;i++)
    {
        printf("%d, ", b[i]);
    }
    printf("}\n");
    matrixVector(a,b,c,3,3);
    printf("c: \n{ ");
    for(int i = 0;i<3;i++)
    {
        printf("%d, ", c[i]);
    }
    printf("}\n");
}
```

3. LINKED LIST STRIKES BACK

Called question3.c in the zipped folder.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define DEBUG 0

struct my_node{
   char* i; // Your data that you want to store
   struct my_node * next; // The pointer to the next node
};

typedef struct my_node Node;

Node * first; // point to the first element in my linked list
```

```
void deleteAt(int index)
{
   int counter = 0;
   Node *temp, *temp2=NULL;
   if(first==NULL) return;
   for(temp=first; (temp!=NULL) && (counter<index); temp = temp->next)
   {
       temp2=temp;
       counter++;
   }
   if(temp2==NULL)
   {
       first = temp->next;
       free(temp);
   }
   else if(temp!=NULL)
   {
       temp2->next = temp->next;
       free(temp);
   }
   return;
}

void insertAt(int index,char *data)
{
   int counter = 0;
   if(DEBUG) printf("Inserting at index %d, data = %s, first is at %p\n",
index, data, first);
   Node * temp2=NULL;
   Node * temp=first;
   for(; (temp!=NULL) && (counter<index); temp = temp->next)
   {
       if(DEBUG) printf("Looping through our list, at index %d, data is %s\n",
counter, temp->i);
       temp2 = temp;
       counter++;
   }
   // Example, if index is 2
   // A -> (temp2 points here) B -> (insert at index 2, temp points to C) C ->
D
```

```
   // Now that temp2 points to B, temp points to C. We want to insert after B
   Node * temp3 = malloc(sizeof(Node));
   temp3->i = data;
   if(temp2 == NULL)
   {
       temp3->next = first;
       first = temp3;
   }
   else
   {
       temp3->next = temp; // The new node->next points to C
       temp2->next = temp3; // B-> next point to the new node
   }
}

void print()
{
   for(Node * temp = first; temp!=NULL ; temp = temp->next)
   {
       printf("%s, ", temp->i);
   }
   printf("\n");
}

int getSize(Node * list)
{
   int size=0;
   for(Node * temp = first; temp!=NULL ; temp = temp->next)
       size++;
   return size;
}

int compFunc(char* a, char * b){
   int sum_a = 0, sum_b = 0;
   for (int i=0;a[i];i++){         // get the sum of the ascii values of the
string
       sum_a += (int)(a[i]);
       sum_b += (int)(b[i]);
   }
   if (sum_a > sum_b){
       return 1;
```

```
    } else if (sum_a == sum_b){
        return 0;
    } else {
        return -1;
    }
}

int countPop(char * input){
    int counter=0;
    Node *temp;
    if(first==NULL) return 0;
    for(temp=first; temp!=NULL; temp = temp->next){
        if (compFunc(temp->i, input) == 1){             // compare input with
temp->i
            counter++;
        }
    }
    return counter;
    free(temp);
}

int main(int argc, char* argv[])
{
    // Test case for linked list
    insertAt(0, "aa");
    insertAt(1, "bb");
    insertAt(1, "cc");
    insertAt(4, "hello");
    insertAt(3, "world");
    insertAt(7, "gcc");
    insertAt(6, "midterm");
    print();
    printf("Size of linkedlist is %d\n",getSize(first));
    deleteAt(1);
    print();
    printf("Size of linkedlist is %d\n",getSize(first));
    deleteAt(0);
    print();
    printf("Size of linkedlist is %d\n",getSize(first));
    deleteAt(1000);
    print();
```

```
    printf("Size of linkedlist is %d\n",getSize(first));
    deleteAt(0);
    print();
    printf("Size of linkedlist is %d\n",getSize(first));


    // Test case for compFunc
    int (*compPtr)(char*, char*) = &compFunc;   // results: uses the function
pointer compPtr
    printf("%d\n", (*compPtr)("aca", "aba"));   // 1
    printf("%d\n", (*compPtr)("a", "b"));       // -1
    printf("%d\n", (*compPtr)("aaa", "aaa"));   // 0


    // Test case for countPop                   results: look correct
    printf("%d\n", countPop("aa"));             // 4
    printf("%d\n", countPop("zzzzzz"));         // 1
    printf("%d\n", countPop("aeiou"));          // 3
    printf("%d\n", countPop("ggwp"));           // 3

    return 0;
}
```

4. DATA IN EACH BYTE

base address	base+0	base+1	base+2	base+3	base+4	base+5	base+6	base+7
0x10000	0	1	2	3	4	5	6	7
0x10008	0	1	2	3	4	5	6	7
0x10010	0	1	XX	XX	XX	XX	XX	XX
0x10018	XX	XX	XX	XX	XX	XX	XX	XX
0x10020	XX	XX	XX	XX	YY	YY	YY	YY
0x10028	YY	YY	YY	YY	YY	YY	YY	YY
0x10030	YY	YY	YY	YY	YY	YY	YY	YY
0x10038	YY	YY	YY	YY	YY	YY	YY	YY
0x10040	YY	YY	YY	YY	YY	YY	YY	YY

0x10048	YY	YY	YY	YY	YY	YY	YY	YY