//Derek D Kim

//Jan. 21, 2016

//CS 241

//HW0

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <string.h>

int main(){

//Chapter 1

//Hello World (System call style)

printf("-Hello World (System call style)\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

write(1, "Hi! My name is \n", 16);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\n");

//Hello Standard Error Stream

printf("-Hello standard Error Stream\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int count;

for(count = 0; count < 4; count++){

write(1, "\*\*\*", count);

printf("\n");

}

printf("\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\n");

//Writing to files

printf("-Writing to files\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n");

mode\_t mode = S\_IRUSR | S\_IWUSR;

int file = open("test1.txt", O\_CREAT | O\_RDWR | O\_TRUNC, mode);

write(file, "Hi! My name is \n", 16);

close(file);

printf("need to check by cat test1.txt\n");

printf("\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\n");

//Not everything is a system call

printf("-Not everything is a system call\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n");

// mode\_t mode = S\_IRUSR | S\_IWUSR;

printf("need to check by cat test2.txt\n");

printf("\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\n");

close(1);

int file1 = open("test2.txt", O\_CREAT | O\_RDWR | O\_TRUNC, mode);

printf("Hi! My name is \n");

close(file1);

//Difference between write() and printf()

//write could be controlled on how long or where to write

//but printf only prints to std output

//Chapter 2

return 0;

}

//Derek D Kim

//Jan. 21, 2016

//CS 241

//HW0

#include <stdio.h>

#include <limits.h>

int main(){

//Chapter 2

//how many bits are there in a byte

printf("char is a byte and there are %d bits\n", CHAR\_BIT);

printf("char is 1 byte\n");

printf("Number of bytes for int, double, float, long, long long \n %d, %d, %d, %d, %d \n", sizeof(int), sizeof(double), sizeof(float), sizeof(long), sizeof(long long));

//Follow the int pointer

//if the address of data is 0x7fbd9d40

// data+2 would be 0x7fbd9d56

//also data[3] == 3[data]

//sizeof character arrays, incrementing pointers

//char \*ptr = "hello";

//\*ptr = 'J';

//above code will seg fault because \*ptr is read only

printf("sizeof(\"Hello/0World\") will return %d\n", sizeof("Hello\0World"));

printf("strlen(\"Hello/0World\") will return %d\n", strlen("Hello\0World"));

printf("example of sizeof(x) = 3 would be sizeof(\"ab\") = %d\n" , sizeof("ab"));

printf("example of sizeof(y) = 4 or 8 depending on machine would be sizeof(int) = %d\n", sizeof(int));

//this would depend on if your machine is 32 or 64 bits

return 0;

}

//Derek D Kim

//Jan. 21, 2016

//CS 241

//HW0

#include <stdio.h>

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

//Chapter 3

//Program arguments argc argv

//two ways to find the length of argv is to return the value of

//argc or loop around argv until hitting a NULL ptr

//also argv[0] is the program it self

//Environment Variables

//ptrs to environment variables are stored in environ

//String searching (Strings are just char arrays)

//on a machine where pointers are 8 bytes

//sizeof(ptr) would be 8 because that is the size of the pointer

//sizeof(array) would be 6 where each char would be 1 byte with 0 at the end

//Lifetime of automatic variables

//Datastructure that is managing the lifetime of automatic variables are stack

}

//Derek D Kim

//Jan. 21, 2016

//CS 241

//HW0

#include <stdio.h>

#include <time.h>

#include <string.h>

#include <stdlib.h>

//Chapter 4

//Memory allocation using malloc, heap and time

//if you want to use data after the lifetime of the function it was created in

//then you could use a static data out side of the function or create memory using malloc

//For every malloc there is a "free"

//Heap allocation Gotchas

//malloc would fail if we used up all the memory

//time would return a time\_t and ctime would return a char\* where most people can understand

//free(ptr); free(ptr); is wrong because it is freeing the memory twice

//free(ptr);

//printf("%s\n", ptr);

//above code is incorrect because it is trying to access part of the memory that have been freed

//above 2 mistakes could be avoided by only releasing the memory once your done with the memory

//also you could set the ptr to NULL once it has been freed as Dangling Pointer

//struct, typedefs and a linked list

struct data{

char\* name;

int age;

struct data\* friends;

};

typedef struct data Person;

Person\* create\_person(char\*, int);

void destroy\_person(Person\*);

int main(){

Person\* person1 = (Person\*) malloc(sizeof(Person));

Person\* person2 = (Person\*) malloc(sizeof(Person));

person1->name = "Agent Smith";

person2->name = "Sonny Moore";

person1->age = 128;

person2->age = 256;

person1->friends = person2;

person2->friends = person1;

printf("%s %s %d %d %s %s \n", person1->name, person2->name, person1->age, person2->age, person1->friends->name, person2->friends->name);

free(person1);

free(person2);

//Duplicating strings, memory allocation and deallocation of structures

Person\* person3 = create\_person("Agent Smith", 128);

Person\* person4 = create\_person("Sonny Moore", 256);

printf("%s %s %d %d \n", person3->name, person4->name, person3->age, person4->age);

destroy\_person(person3);

destroy\_person(person4);

return 0;

}

Person\* create\_person(char\* p\_name, int p\_age){

Person\* result = (Person\*) malloc(sizeof(Person));

result->name = strdup(p\_name);

result->age = p\_age;

result->friends = NULL;

return result;

}

void destroy\_person(Person\* p){

free(p->name);

p->name = NULL;

free(p);

p = NULL;

}

//Derek D Kim

//Jan. 21, 2016

//CS 241

//HW0

//Chapter 5

//Reading characters, Trouble with gets

//Inorder to get stdin to stdout we could use putchar() function

//gets() function would have problems because it could have overflow

//Introducing sscanf and friends

#define \_GNU\_SOURCE

#include <stdio.h>

#include <stdlib.h>

int main(){

char \* data = "Hello 5 World";

//do i need to use strcpy?

char buffer1[6];

char buffer2[6];

int num = 99;

int result = sscanf(data, "%s %d %s", buffer1, &num, buffer2);

printf("Result: %d %s %d : %s\n", result, buffer1, num, buffer2);

//getline is useful

//you would need to #define \_GNU\_SOURCE

char \* buffer = NULL;

size\_t capacity = 0;

ssize\_t getline\_result = getline(&buffer, &capacity, stdin);

if(result > 0 && buffer[getline\_result -1] == '\n'){

buffer[getline\_result-1] = 0;

}

printf("%d : %s\n", getline\_result, buffer);

free(buffer);

return 0;

}

//Derek D Kim

//Jan. 21, 2016

//CS 241

//HW0

//C Development

//complier flag to generate debug build

//-g -O

//modifying makefile and using make command would not do anything because there were no changes to the program files

//yes correct tab and spacing is importent in makefiles

//one of the example on difference between heap and stack memory would be

//heap would be used in globle var and stack would be used in func var

//yes, there are other parts of memory other than stack and heap