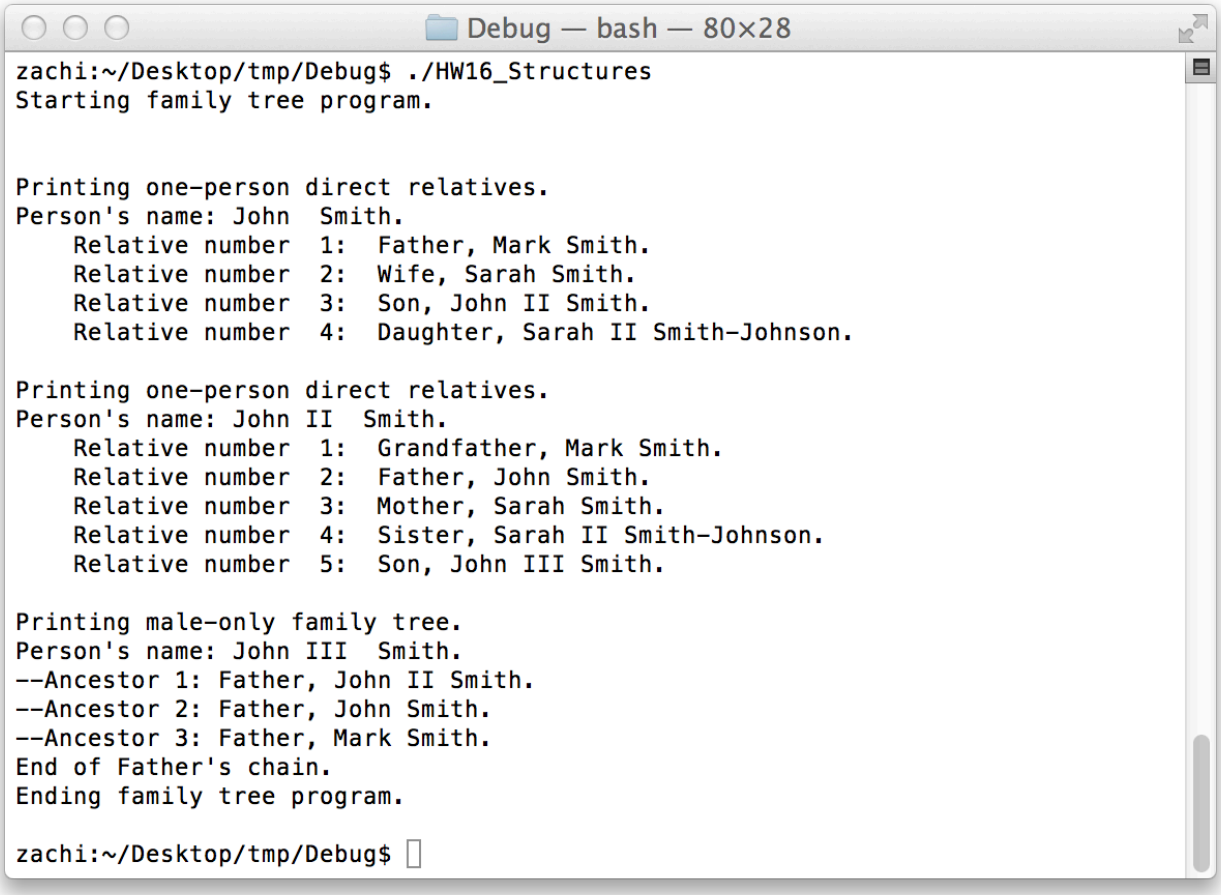
Assignment HW17
Due date (on or before): Announced in class.

Question 1: Family tree

In class we wrote a simple family program. Now, we are going to make it a little fancier. Below is the skeleton for the program (very similar to the one we used in class, WITH modifications).

Please fill in the places indicated by **// YOUR CODE HERE //**
Your goal is to get the output as described in the screenshot below.



Code Skeleton. Read the whole thing, but pay special attention to the parts:
// YOUR CODE HERE //

```
//
//  main.c
```

```c
//  HW16_Structures
//

#include <stdio.h>
#include <string.h>


////////////////////
// Declarations
////////////////////

#define CLAN_SIZE 120

enum relationsFamily {wife, husband, spouse, son, daughter,
father, mother, brother,sister, grandfather,grandchild};
const char* relationsFamilyStrings[] = { "Wife",
"Husband","Spouse","Son","Daughter","Father",
"Mother","Brother","Sister","Grandfather","GrandChild"};

struct Relatives{
    int relation;
    struct Person* ptr;
};

struct Person{
    char first[80];
    char last[80];
    int  numRelatives;
    struct Relatives relatives[50];
};

// NOTE: These ones are now Global variables!

struct Person familia[CLAN_SIZE];
int familiaIdx = 0;

////////////////////
// Functions
////////////////////

void createPerson(struct Person* p, char* s1, char* s2)
{
    strcpy(p->first, s1);
    strcpy(p->last, s2);
    p->numRelatives = 0;
}
```

```c
struct Person* findName(char* s1a, char* s1b)
{
    struct Person* ptr = NULL;

     // YOUR CODE HERE //
     //
     //  You need to search through the array of structures
     // and use the library function 'strcmp()' to check for the
     // right person record.

    return (ptr);
}

// In class we wrote this one:
// void setRaltive(struct Person* p1, struct Person* p2, int r)
// The setFancyRelative is VERY similar.

void setFancyRaltive(char* s1a, char* s1b, int r1, char* s2a,
char* s2b, int r2)
{
     // YOUR CODE HERE //
     //
     //
     //  VERY similar to the regular setRelatice we did in class,
     // but you need to do two things:
     // 1. Call 'findName()' to get the pointers to Persons.
     // 2. Update BOTH records.


}



void printFancyTree(char *s1, char *s2)
{



     // YOUR CODE HERE //
     //
     //  VERY similar to the regular printTree we did in class,
     // but you need to do two things:
     // 1. Call 'findName()' to get the pointer to the Person.
     // 2. Make sure to print the enum-relation in a nice form.
```

```
}

void printFathers(char *s1, char *s2, int idx)
{
     // YOUR CODE HERE //
     //
     // Think about doing something similar to printTree, but not
     // printing everyone. Then, you will also need recursion!!

}


/// NO need to touch from here on !!!

void initFamily()
{
    createPerson(&familia[familiaIdx++], "Mark", "Smith");  //
grandfather
    createPerson(&familia[familiaIdx++], "John", "Smith");  //
dad
    createPerson(&familia[familiaIdx++], "Sarah", "Smith"); //
mom
    createPerson(&familia[familiaIdx++], "John II", "Smith");
// 1 out of two kids
    createPerson(&familia[familiaIdx++], "Sarah II", "Smith-
Johnson");  // 2 out of two kids
    createPerson(&familia[familiaIdx++], "John III", "Smith");


    //setRaltive(&familia[0], &familia[2], son);
    //setRaltive(&familia[0], &familia[1], wife);

    setFancyRaltive("Mark","Smith",father,"John","Smith",son);
    setFancyRaltive("Mark","Smith",grandfather,"John
II","Smith",grandchild);
    setFancyRaltive("Mark","Smith",grandfather,"Sarah
II","Smith-Johnson",grandchild);


setFancyRaltive("John","Smith",husband,"Sarah","Smith",wife);
    setFancyRaltive("John","Smith",father,"John
II","Smith",son);
    setFancyRaltive("Sarah","Smith",mother,"John
II","Smith",son);
```

```
    setFancyRaltive("John","Smith",father,"Sarah II","Smith-
Johnson",daughter);
    setFancyRaltive("Sarah","Smith",mother,"Sarah II","Smith-
Johnson",daughter);

    setFancyRaltive("John II","Smith", brother,"Sarah
II","Smith-Johnson",sister);

    setFancyRaltive("John II","Smith",father,"John
III","Smith",son);

}

//////////////////
// main()
//////////////////

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

    printf("Starting family tree program.\n\n");

    initFamily();


    printf("\nPrinting one-person direct relatives.\n");
    printFancyTree("John","Smith");

    printf("\nPrinting one-person direct relatives.\n");
    printFancyTree("John II","Smith");

    printf("\nPrinting Father-only family tree.\n");
    printf("Person's name: %s  %s.\n","John III", "Smith");
    printFathers("John III","Smith",1);



    printf("Ending family tree program.\n\n");

    return(0);
}
```

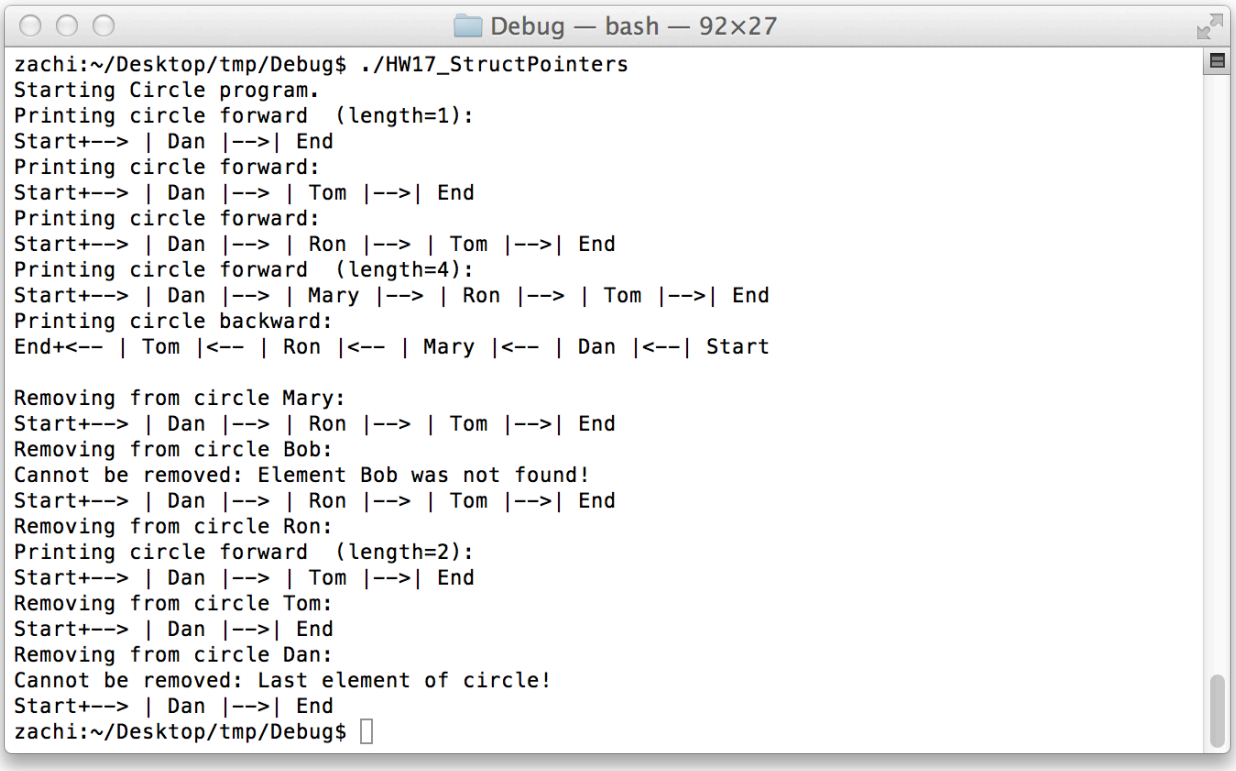
********** Start of code + Screen shot**********

********** End of code + Screen shot**********

Question 2: The Circle

In class we wrote a simple circle program. Now, we are going to add some functionality: removing people, length of circle, etc.
Below is the skeleton for the program (very similar to the one we used in class, WITH modifications).

Please fill in the places indicated by **// YOUR CODE HERE //**
Your goal is to get the output as described in the screenshot below the code.



Code Skeleton. Read the whole thing, but pay special attention to the parts:
// YOUR CODE HERE //

```
//
//  main.c
//  HW17_StructPointers
//
//
#include <stdio.h>
#include <string.h>
```

```c
struct Data{
    char name[80];
    struct Data* ptrPrev;
    struct Data  *ptrNxt;
};


struct Data players[80];
int playersIdx = 0;

void printCircleFwd(struct Data* ptr0)
{
    struct Data* ptr = ptr0;

    printf("Start+-->");
    do {
        printf(" | %s |-->", ptr->name);
        ptr = ptr->ptrNxt ;
    } while (ptr != ptr0);

    printf("| End");

}

void printCircleBack(struct Data* ptr0)
{
    // YOUR CODE HERE
    // Very similar to the fwd version.


}


void insertToCircleAlpha(struct Data *ptrCirc, struct Data
*ptrNew)
{
    struct Data* ptr = ptrCirc;

    do
    {
        if ( strcmp(ptr->name, ptrNew->name)>0 ) break;
        ptr = ptr->ptrNxt;
    }
    while ( ptr != ptrCirc);
```

```
    (ptr->ptrPrev) ->ptrNxt = ptrNew;
    ptrNew->ptrPrev = (ptr->ptrPrev) ;

    ptr->ptrPrev = ptrNew;
    ptrNew->ptrNxt = ptr;


}

int lengthOfCircle(struct Data *ptr0)
{
     int l=0;
     // YOUR CODE HERE



    return(l);
}


void removeFromCircle(char *s, struct Data *ptr0)
{
    // YOUR CODE HERE
    // Three things you need to do:
    // 1. find the right element to take out.
    //   1.a. If element not found, print a notice and exit
    // 2. Make sure this is NOT the last element.
    // 3. do the removal process.




}
////////////
// main
////////////

int main(void)
{
    struct Data *circPtr;

     printf("Starting Circle program.\n");

    strcpy(players[playersIdx].name,"Dan");  //0
    players[playersIdx].ptrPrev = &players[playersIdx];
    players[playersIdx].ptrNxt = &players[playersIdx];
```

```
    playersIdx++;

    strcpy(players[playersIdx].name,"Ron");  //1
    players[playersIdx].ptrPrev = &players[playersIdx];
    players[playersIdx].ptrNxt = &players[playersIdx];
    playersIdx++;

    strcpy(players[playersIdx].name,"Tom");  //2
    players[playersIdx].ptrPrev = &players[playersIdx];
    players[playersIdx].ptrNxt = &players[playersIdx];
    playersIdx++;

    strcpy(players[playersIdx].name,"Mary");  //3
    players[playersIdx].ptrPrev = &players[playersIdx];
    players[playersIdx].ptrNxt = &players[playersIdx];
    playersIdx++;

    // first element of circle
    circPtr = &players[0];
    printf("Printing circle forward (length=%d):\n",
lengthOfCircle(circPtr));
    printCircleFwd( circPtr);


    // Adding to circle
    insertToCircleAlpha(circPtr, &players[2]);  // insert Tom
    printf("\nPrinting circle forward:\n");
    printCircleFwd( circPtr);

    insertToCircleAlpha(circPtr, &players[1]);  // insert Ron
    printf("\nPrinting circle forward:\n");
    printCircleFwd( circPtr);

    insertToCircleAlpha(circPtr, &players[3]);  // insert Mary
    printf("\nPrinting circle forward
(length=%d):\n",lengthOfCircle(circPtr));
    printCircleFwd( circPtr);

    // Printing circle backward
    printf("\nPrinting circle backward:\n");
    printCircleBack( circPtr->ptrPrev);

    // Removing from circle
    printf("\n\nRemoving from circle Mary:\n");
    removeFromCircle("Mary",circPtr);
    printCircleFwd( circPtr);
```

```
    printf("\nRemoving from circle Bob:\n");
    removeFromCircle("Bob",circPtr);
    printCircleFwd( circPtr);

    printf("\nRemoving from circle Ron:\n");
    removeFromCircle("Ron",circPtr);
    printf("Printing circle forward
(length=%d):\n",lengthOfCircle(circPtr));
    printCircleFwd( circPtr);

    printf("\nRemoving from circle Tom:\n");
    removeFromCircle("Tom",circPtr);
    printCircleFwd( circPtr);

    printf("\nRemoving from circle Dan:\n");
    removeFromCircle("Dan",circPtr);
    printCircleFwd( circPtr);


    printf("\n");
    return(0);
}
```

********** Start of code + Screen shot**********

********** End of code + Screen shot**********