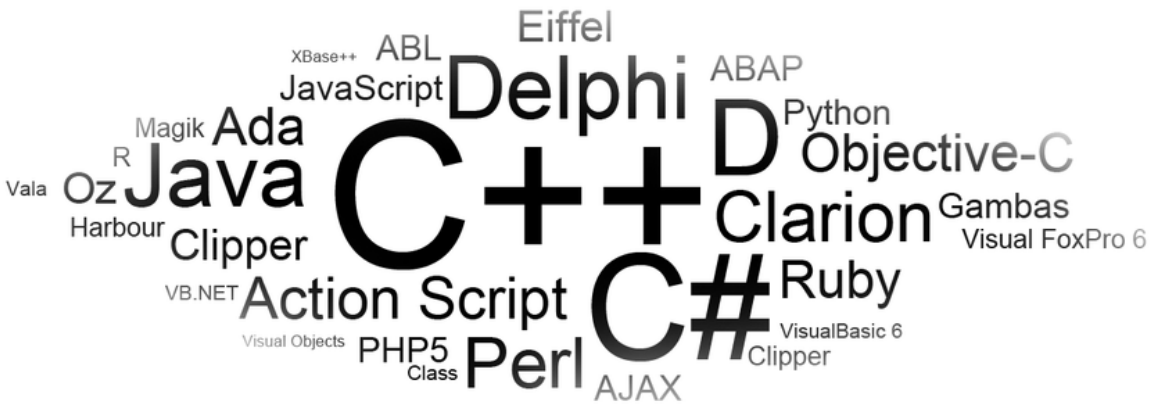
CECS 326
Operating Systems



Lauro Cabral

February 7, 2018

Lauro199471@gmail.com

Software Description

The purpose of the lab is demonstrating memory management. At start-up the program initializes memory to heap by using the following function:

*f(0)=2700 & f(n+1)=2*f(n), where n is an index of the integer array for 0< n < 20*

After we iterate through this function 20 times, have 20 char pointers that point the base address to each of the 20 locations in heap. With these 20 char pointers we populate the heap with random upper-case letters. The following image represents what the pointers are doing.

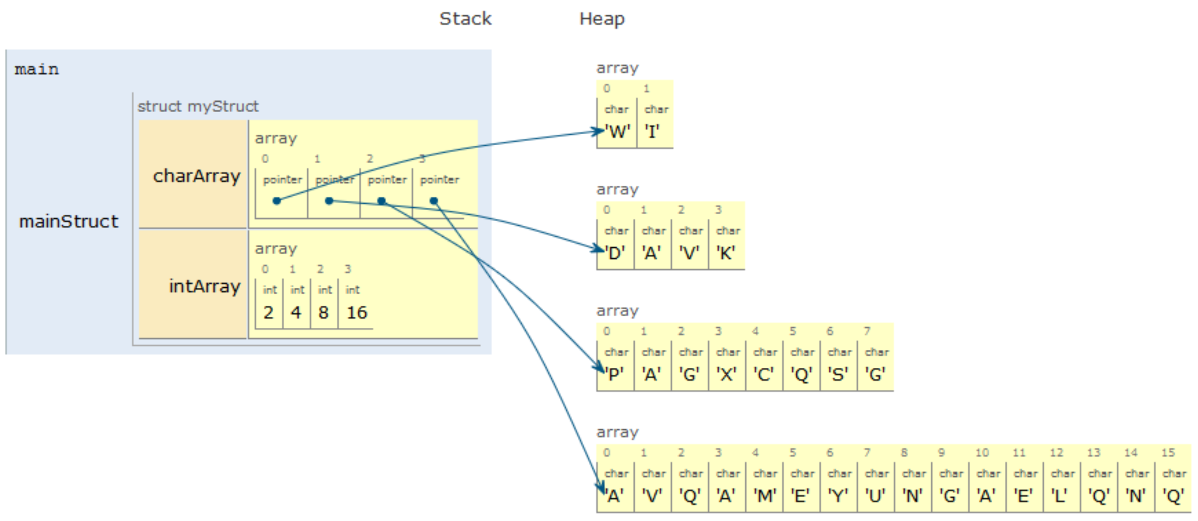


Figure (1)

As you can see in *Figure (1)* we create an object called *mainStruct* that has two data types which are char* array and an integer array. The *intArray* determines how much memory does the program use for each charter pointer. In the program due in lab is doing what the example is doing but in a much larger scale.

After Startup the user is presented with a menu that has 4 options. The options are:

1. Access a pointer
2. List deallocated Memory(Index)
3. Deallocate all Memory
4. Exit

If *(1)* is selected to access a pointer then the user is presented with a line saying "**Index>>** " , which is promoting the user to enter a number from 0-19 to access one of the twenty pointers in the object. Once an index has been selected, then the user is presented with another submenu with 3 options that are:

1. Print the first 10 Char's
2. Delete all the Char's associated with this pointer
3. Return to main menu

If *(1)* is selected the first 10 characters are printed to show the values that the pointer is pointing to in the heap. If *(2)* is selected, then all of the values that the pointer is pointing to is deleted in heap and the CPU can use the memory locations. If *(3)* is selected, the program returns to the main menu which was the first menu presented in page 1.

Once in the main menu if the user selects option *(2)* which is list *all deallocated memory(index)*, the user is presented with a list of all the pointers that have been deleted by deleting it by accessing it or by selecting option *(3)* which is *deallocate all memory.* If the user selects Option *(4)*, it will cause the program to shut down.

If the users access an index that has been deleted, then program notifies the user that the index is deallocated and will immediately allocate memory and give it a new set of string values.

Code

Main.c

```c
#include <stdio.h>
#include < stdlib.h >
#include < time.h >
#include "color.h"
#include "supportassignment1.h"

void initHeader(void)
{
    printf(yellow "\t\t\t\t\t\t  CECS 326\n" reset);
    printf(gray "\t\t\t\t\t\tLauro Cabral\n" reset);
}

void menuStart()
{
    printf(red "(1)" reset " Access a Pointer"
           green "  (2)"  reset " List Deallocated Memory(Index)"
           blue "  (3)"   reset " Deallocate All Memory"
           yellow "  (4)" reset " Exit Program\n>> ");
}

int menu1(void)
{
    int x;
    printf(red "Index: " reset);
    scanf("%d", & x);
    return x;
}

void menu1_2(void)
{
    printf(red "(1)" reset " Print Char's"
           green "  (2)" reset " Delete All Chars"
           blue "  (3)"  reset " Return to Main Menu\n>> ");
}

int main()
{
    int index;
    int userInput = 0, i = 0;
    int deallocatedIndex[21];
    int deallocatedInt = 0;

    for (i = 0; i < 21; i = i + 1)
    {
        deallocatedIndex[i] = 255;
    }
    printf("\n");
    myStruct mainStruct;
    srand(time(0));
    init_INT_ARRAY( & mainStruct);
    init_CHARPTR_ARRAY( & mainStruct);

    do
    {
        initHeader();
        menuStart();
        scanf("%d", & userInput);

        // Access a Pointer
        if (userInput == 1)
        {
            initHeader();
            index = menu1();
            if (deallocatedIndex[index] == index)
            {
                printf(cyan "**PT IS GONE ... re-Initialized**\n"reset);
                init_CHARPTR_ARRAY_index( & mainStruct, index);
                deallocatedIndex[index] = 255;
            }
            while (1)
            {
                deallocatedInt = deallocatedInt % 20;
                menu1_2();
                scanf("%d", & userInput); // Exit
                if (userInput == 3) break; // Delete all Char's
                else if (userInput == 2)
                {
                    if (deallocatedIndex[index] == 255)
                    {
                        deallocatedIndex[index] = index;
                        delete mainStruct.charArray[index];
                    }
                } // Print first 10 Chars
                else if (userInput == 1) print_CHARPTR_ARRAY( & mainStruct, index);
            }
        }
        // List Deallocated Pointers
        else if (userInput == 2)
        {
            i = 0;
            printf(gray "deallocated memory (index):\n" reset);
            for (i = 0; i < 20; i = i + 1)
            {
                if (deallocatedIndex[i] != 255) printf("%d " reset, deallocatedIndex[i]);
            }
            printf("\n");
        }
        // Deallocate All Pointer
        else if (userInput == 3)
       {
            deallocatedInt = 0;
            for (i = 0; i < 20; i = i + 1)
            {
              if (deallocatedIndex[i] == i || i == 20) {}
              else
              {
                deallocatedIndex[i] = i;
                delete mainStruct.charArray[i];
              }
             }
        }
        printf(reset "\n");
       } while (userInput != 4);
       printf(reset "Exiting...\n");
       return 0;
       }
```

SupportAssignment.H

```c
#ifndef SUPPORTASSIGNMENT1_H
#define SUPPORTASSIGNMENT1_H

struct CharAndInt
{   //charArray is an array of 20 char*
    char * charArray[20];
    // intArray is an array of 20 int
    int intArray[20];
};
typedef struct CharAndInt myStruct;

void init_INT_ARRAY(myStruct* c)
{
    c->intArray[0] = 2700;
    int i = 0;
    for (i = 0; i < 20; i = i + 1)
    {
        c-> intArray[i + 1] = 2 * c->intArray[i];
    }
}

void init_CHARPTR_ARRAY(myStruct* c)
{
    int i, j;
    for (i=0;i<20;i=i+1)
    {
        c-> charArray[i] = (char*)malloc(c-> intArray[i]);
        for (j = 0; j < c->intArray[i]; j=j+1)
        {
            c->charArray[i][j] = (rand()%25)+65;
        }
    }
}

void init_CHARPTR_ARRAY_index(myStruct * c, int index)
{
    int j;
    c->charArray[index] = (char*)malloc(c->intArray[index]);
    for (j = 0; j < c - > intArray[index]; j = j + 1)
    {
        c - > charArray[index][j] = (rand() % 25) + 65;
    }
}

void print_CHARPTR_ARRAY(myStruct* c, int index)
{
    int i;
    printf("\n");
    printf(reset "[" green "%15p" reset "]: ", c - > charArray[index]);
    for (i = 0; i < 10; i = i + 1)
        printf(magenta "%1c " reset, c - > charArray[index][i]);
    printf("\n");
}

void deallocateAllMemory(myStruct * c)
{
    int i;
    for (i = 0; i < 20; i = i + 1)
        c - > charArray[i] = NULL;
}
```

```c

void deallocateAMemory(myStruct* c, int i)
{
    free((*c).charArray[i]);
    c->charArray[i] = NULL;
}

void deleteChar(myStruct* c, int i)
{
    free((*c).charArray[i]);
    c->charArray[i] = NULL;
}
#endif // SUPPORTASSIGNMENT1_H

```

Colors.h

```c
#ifndef COLOR_H
#define COLOR_H

#define red     "\033[1;31m"
#define green   "\033[1;32m"
#define yellow  "\033[1;33m"
#define blue    "\033[1;34m"
#define purple  "\033[1;35m"
#define cyan    "\033[1;36m"
#define gray    "\033[1;37m"
#define bold    "\033[1;29m"
#define magenta "\033[1;35m"
#define reset   "\033[0m"

#endif // COLOR_H

```