CS200 - Computer Organization

Arrays, Structs, & Pointers

Aravind Mohan

Allegheny College

September 13, 2022

For starters!

- **Arrays:** Static Vs Dynamic
- Pointer Arithmetic
- Different ways for Pointer implementation

Review on Arrays

- Arrays are data structures consisting of related data items of the same type.
- A struct (structure) in C - a data structure consisting of related data items of possibly different types.
- Arrays and structures are usually ”static” entities in that they remain the same size throughout program execution. We may chose to implement Dynamic arrays with the use of **malloc** keyword.

Extreme Programming:

Initializing array with even numbers.

Try on your own:

Initializing array with odd numbers.

Extreme Programming:

Computing the sum of elements in an array.

Extreme Programming:

- Our next example uses arrays to summarize the results of data collected in a survey.
- Consider the problem statement.

  Forty students were asked to rate the quality of the food in the student cafeteria on a scale of 1 to 10 (1 means awful and 10 means excellent). Place the 40 responses in an integer array and summarize the results of the poll.
- **This is a typical array application.**

Important Properties of Arrays in C

- C has no array bounds checking to prevent the program from referring to an element that does not exist.
- Thus, an executing program can ”walk off” either end of an array without warning-a security problem.
- You should ensure that all array references remain within the bounds of the array.

Take home exercise:

Do exercise 1-13 and 1-14 on KR page 24.

Pointers:

- Pointers enable programs to simulate pass-by-reference, to pass functions between functions, and to create and manipulate dynamic data structures, i.e., data structures that can grow and shrink at execution time, such as linked lists, queues, stacks and trees.

Pointers:

- Pointers are variables whose values are memory addresses.
- Normally, a variable directly contains a specific value.
- A pointer, on the other hand, contains an address of a variable that contains a specific value.

Pointers:

- To recap, a variable name directly references a value, and a pointer indirectly references a value.
- Referencing a value through a pointer is called **Indirection**.

Extreme Programming:

- Revise the rating example code using **Pointer arithmetic**

For starters!

- How is the **Dynamic** array differ from a Static array?
- What is Structs? and how can we use them?

Variable Length Arrays

- Variable length arrays, use the space in the Stack and hence limited to store more data. It is often looked at an inefficient way of creating dynamic arrays.

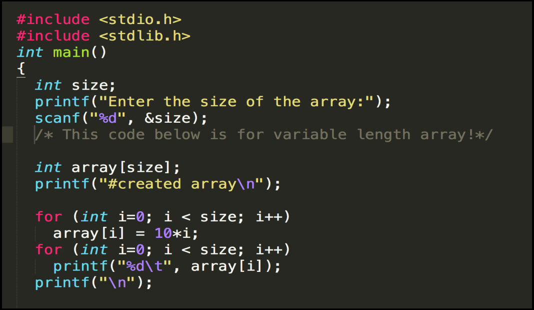
Dynamic Arrays

- Malloc based dynamic arrays, uses the space in Heap and hence can store data of larger size. It is the most efficient method to create a Dynamic array.
- Malloc is simply memory allocation procedure, that allocates blocks of memory based on programmers request.

Dynamic Arrays

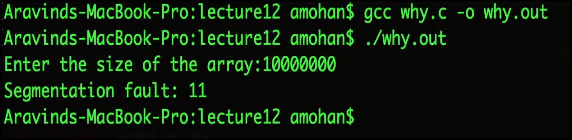
- A Pointer, can then be used to store, update, and read data elements from the memory. Malloc is part of the ”stdlib” standard library.
- Look at ”why.c” code in the src directory, to understand the difference.

Variable Length Array - Code



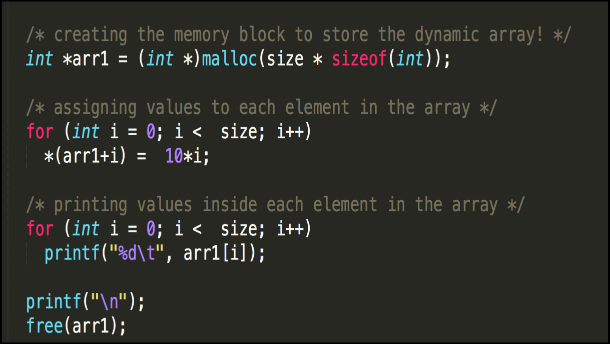
```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int size;
   printf("Enter the size of the array:");
   scanf("%d", &size);
   /* This code below is for variable length array!*/

   int array[size];
   printf("#created array\n");

   for (int i=0; i < size; i++)
     array[i] = 10*i;
   for (int i=0; i < size; i++)
     printf("%d\t", array[i]);
   printf("\n");
```

Variable Length Array - Code



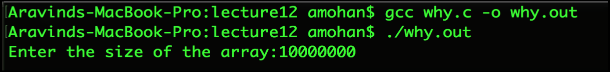
```
Aravinds-MacBook-Pro:lecture12 amohan$ gcc why.c -o why.out
Aravinds-MacBook-Pro:lecture12 amohan$ ./why.out
Enter the size of the array:10000000
Segmentation fault: 11
Aravinds-MacBook-Pro:lecture12 amohan$
```

Dynamic Array - Code

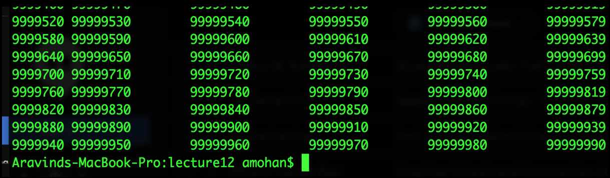


```c
/* creating the memory block to store the dynamic array! */
int *arr1 = (int *)malloc(size * sizeof(int));

/* assigning values to each element in the array */
for (int i = 0; i <  size; i++)
  *(arr1+i) =  10*i;

/* printing values inside each element in the array */
for (int i = 0; i <  size; i++)
  printf("%d\t", arr1[i]);

printf("\n");
free(arr1);
```

Dynamic Array - Code



```
Aravinds-MacBook-Pro:lecture12 amohan$ gcc why.c -o why.out
Aravinds-MacBook-Pro:lecture12 amohan$ ./why.out
Enter the size of the array:10000000
```

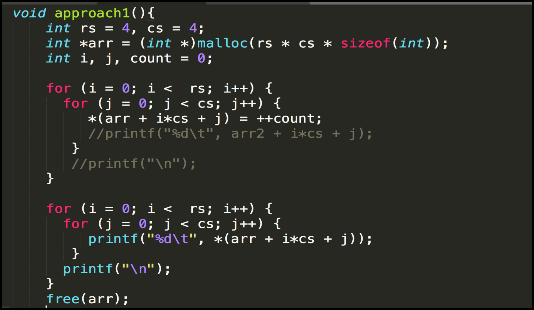


```
9999520 99999530      99999540      99999550      99999560      99999579
9999580 99999590      99999600      99999610      99999620      99999639
9999640 99999650      99999660      99999670      99999680      99999699
9999700 99999710      99999720      99999730      99999740      99999759
9999760 99999770      99999780      99999790      99999800      99999819
9999820 99999830      99999840      99999850      99999860      99999879
9999880 99999890      99999900      99999910      99999920      99999939
9999940 99999950      99999960      99999970      99999980      99999990
Aravinds-MacBook-Pro:lecture12 amohan$
```

2 Dimensional Dynamic Array

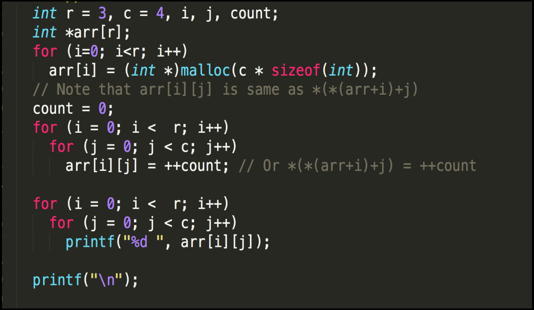
- How can we create a 2 Dimmensional Dynamic Array using a Malloc?
- **Approach 1:** A single pointer is used to allocate the memory block to hold the entire 2D array.
- **Approach 2:** An array of pointers is created for the individual rows. The element in the array [pointer], is used to allocate a memory block to hold all the columns in the individual rows.

Dynamic Array - Approach 1:



```
void approach1(){
    int rs = 4, cs = 4;
    int *arr = (int *)malloc(rs * cs * sizeof(int));
    int i, j, count = 0;

    for (i = 0; i <  rs; i++) {
      for (j = 0; j < cs; j++) {
         *(arr + i*cs + j) = ++count;
         //printf("%d\t", arr2 + i*cs + j);
      }
      //printf("\n");
    }

    for (i = 0; i <  rs; i++) {
      for (j = 0; j < cs; j++) {
         printf("%d\t", *(arr + i*cs + j));
      }
      printf("\n");
    }
    free(arr);
```

Dynamic Array - Approach 2:



```
int r = 3, c = 4, i, j, count;
int *arr[r];
for (i=0; i<r; i++)
    arr[i] = (int *)malloc(c * sizeof(int));
// Note that arr[i][j] is same as *(*(arr+i)+j)
count = 0;
for (i = 0; i <  r; i++)
    for (j = 0; j < c; j++)
        arr[i][j] = ++count; // Or *(*(arr+i)+j) = ++count

for (i = 0; i <  r; i++)
    for (j = 0; j < c; j++)
        printf("%d ", arr[i][j]);

printf("\n");
```

Structs in C

- A datatype declaration that defines a grouped list of variables to be placed under one name in a block of memory.
- Think of it as a class without functions, just variables.
- **Struct** can hold any type of members including arrays.
- **Possible to:** create a pointer to Struct.
- Let us do some examples.

Reading Assignment

Section [1.6], [5.6 - 5.9], [6.1 - 6.4] in **KR**

Questions

- Feel free to ask your questions.
- I welcome you to stop by after class time to clarify any confusion related to class topics.
- Also please stop by during my office hours so we can spend some time together.