C-Refresher: Session 09 Dynamic Memory

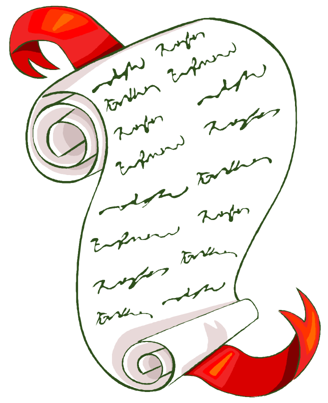
Arif Butt
Summer 2017

I am Thankful to my student Muhammad Zubair bcsf14m029@pucit.edu.pk for preparation of these slides in accordance with my video lectures at

http://www.arifbutt.me/category/c-behind-the-curtain/

Today's Agenda

- Heap Management
- Growing String on Heap
- 2D-Integer Arrays
- 2D-Character Arrays



Heap Management

Heap Management(cont…)

- We use heap when at the time of writing the program, we don't have idea that how much memory we may need during the working of the program

❑**alloca()**

- `alloca()` is a function that is used to allocate specified number of bytes in the caller's stack during the program execution
- `void alloca(int size);` /*`size` number of bytes*/

Heap Management(cont…)

- But the limitation of this memory is that it does not survive after the return statement of the function, in which it is called, i.e. it is freed automatically when the function, that has called `alloca()`, returns
- In stack, all the memory that is allocated is automatically freed, that is why the variables that are allocated on stack are called automatic variables
- In heap, the programmer is responsible for freeing the memory allocated

Heap Management(cont…)

- In C++, `new` and `delete` operators are used for allocating and deallocating the memory on heap
- In C, `malloc()` family of functions is used for heap memory management
- `malloc()` family includes
- `void* malloc(size_t size);`
- `void free(void *ptr);`
- `void *calloc(size_t nmemb, size_t size);`
- `void *realloc(void *ptr,size_t size);`

Heap Management(cont…)

1. malloc()

- `void* malloc(size_t size);`
- It takes an integer as argument and allocate those many number of bytes
- In case of success, it returns a pointer of type `void` to the allocated memory on heap, which can be casted to the required datatype
- In case of failure, it returns `NULL`
- e.g.
- `char* str=(char*)malloc(20);`

Heap Management(cont…)

- It allocates `20` bytes on heap and returns its starting address which is casted to `char*` and then assigned to `str`
- Here, `str` is declared on stack and it is pointing to a memory on heap
- Basically, memory allocated on heap is an unnamed memory whose address has been assigned to `str`
- It is a better practice to tell the number of bytes like this
- `int size=20;`
- `sizeof(char)*size` /*it will pass `1*20` where size of `char` is `1-byte` on my machine*/

Heap Management(cont…)

- In case of `int` it would be
- `sizeof(int)*size` /*it will pass `4*20` where size of `int` is `4` bytes on my machine*/

2. free()

- `void free(void *ptr);`
- This function is used to free the memory on heap, to which `ptr` is pointing
- e.g.
- `free(str);`

Heap Management(cont…)

- Note that the memory allocated must be freed, as when `str` goes out of scope, although we no more have access to the memory allocated, but it remains allocated, so it must be freed

3. realloc()

- `void *realloc(void *ptr, size_t size);`
- It is used to change the size of previously allocated memory, pointed by `ptr`, to the new size specified in the second argument
- e.g.
- `char* str2=realloc(str,sizeof(char)*30);`

Heap Management(cont…)

- There is no trouble if you are increasing the size of memory, but be careful, if decreasing the size because it can cause loss of data

4. calloc()

- `void *calloc(size_t nmemb, size_t size);`
- `malloc()` is only for primitive datatypes
- `calloc()` is used to allocate memory for user defined datatypes
- `size` is the size of datatype
- `nmemb` is the number of objects to be allocated
- Now let's write some programs to understand this

Heap Management(cont…)

/*Simple program showing allocation, using and deallocation of memory on heap*/

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  int size;
  printf("Enter size of the array: ");
  scanf("%d",&size);
  int* arr=(int*)malloc(sizeof(int)*size);
  if(arr==NULL)
    perror(“malloc failed\n”);
```

Heap Management(cont…)

```
printf("The initial values in the array are:\n");
 for(int i=0;i<size;i++)
   printf("%d\t",arr[i]);    //mostly 0 is there
printf("\nEnter %d elements of the array:\n",size);
 for(int i=0;i<size;i++)
   scanf("%d",&arr[i]);
 printf("The elements of the array are:\n");
 for(int i=0;i<size;i++)
   printf("%d\t",arr[i]);
 printf("\n");
 free(arr);   /*deallocating the previously allocated memory*/
 return 0;}
```

Heap Management(cont…)

- Output of the above program is:

```
Enter size of the array: 5
The initial values in the array are:
0  0     0     0     0
Enter 5 elements of the array:
1
2
3
4
5
The elements of the array are:
1  2     3     4     5
```

Heap Management(cont…)

```
/*In this program, size of the array  is changed using
realloc()*/
#include<stdio.h>
#include<stdlib.h>
int main(){
  int size;
  printf("Enter size of the array: ");
  scanf("%d",&size);
  int* arr=(int*)malloc(sizeof(int)*size);
  if(arr==NULL)     /*checking if malloc() is successful or
not*/
    perror("malloc failed\n");
  printf("Enter %d elements of the array:\n",size);
```

Heap Management(cont…)

```
  for(int i=0;i<size;i++)
    scanf("%d",&arr[i]);
  printf("The elements of the array are:\n");
  for(int i=0;i<size;i++)
    printf("%d\t",arr[i]);
  printf("\nEnter new size of the array: ");
  scanf("%d",&size);     /* taking size for the new array to
be allocated*/
  int* arr2=(int*)realloc(arr,size);    /*size of the
previously allocated array changed according to the newly entered
size*/
```

Heap Management(cont…)

```
if(arr2==NULL)
    perror("realloc failed\n");
  else
    arr=arr2;  /*assigning the address of new memory to arr*/
  printf("Elements of the new array are:\n");
  for(int i=0;i<size;i++)
    printf("%d\t",arr[i]);
  printf("\n");
  free(arr);   /* deallocating the memory on heap*/
  return 0;
}
```

Heap Management(cont…)

- Output of the above program is:

```
Enter size of the array: 3
Enter 3 elements of the array:
1
2
3
The elements of the array are:
1 2    3
Enter new size of the array: 5
Elements of the new array are:
1 2    3    0    0
```

Heap Management(cont…)

```
/*The program takes size of input string from the user, allocates
that much memory on heap and stores a string in it*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main(){
  int size;
  printf("Enter maximum size of sentence you want to
enter and store: ");
  scanf("%d",&size);
  getchar();    //to eat the \n in input queue
  char* sen=(char*)malloc(sizeof(char)*size);
```

Heap Management(cont…)

```
  printf("Now enter the string:\n");
  fgets(sen,size,stdin);
/* To replace the new line character at the end of the string with the null character*/
  char* q;
  if((q=strchr(sen,'\n'))!='\0')
    *q='\0';
  fputs(sen,stdout);
  printf("\n");
  free(sen);    //deallocating the memory
  return 0;
}
```

Heap Management(cont…)

- Output of the above program is:

```
Enter maximum size of sentence you want to enter
and store: 100
All is well that ends well
All is well that ends well
```

- The limitation of this program is that the user has to enter the size first
- If user enters a larger size and then enters a small string, this can cause wastage of memory
- And, on the other hand, if user enters smaller size and enters a large string, it may cause the program to crash

Growing String on Heap

- **Algorithm for dynamically growing the string on heap**
- **Step-1**
  - Get 1 Byte on heap and place `NULL` on it
- **Step-2**

```
do
   read a character
   if it is '\n', break
   if this is 1st character,
      allocate 2 Bytes
   else
      resize array to size+2
   Place character in the array
while input character is not '\n'
```

Growing String on Heap(cont…)

- **Step-3**
  - Place `NULL` at the end of the string
- Now let’s write a program for this algorithm

Growing String on Heap(cont…)

```
/*The program implements above algorithm*/
#include<stdio.h>
#include<stdlib.h>
int main(){
  char* sen=(char*)malloc(sizeof(char)*1);
  sen[0]='\0'; /*this will prevent the program from crash, if
user presses ENTER without entering any character*/
  char ch;
  int size=0;
  printf("Enter string of any size and I will store it
for you:\n");
```

Growing String on Heap(cont…)

```
do{ch=getchar();
   if(ch=='\n')
      break;    //break the loop if user presses ENTER
if(sen[0]=='\0')
      sen=(char*)realloc(sen,2);
   else
      sen=(char*)realloc(sen,size+2);
   sen[size++]=ch;
}while(ch!='\n');
sen[size]='\0';
printf("%s\n",sen);
free(sen);
return 0;}
```

Growing String on Heap(cont…)

- Output of the above program is:

```
Enter string of any size and I will store it for
you:
All is well that ends well
All is well that ends well
```

- You can input string of any size and the program can store it
- Now let's write a program to show the use of `calloc()` for allocating memory for user-defined objects

Use of calloc() function

/*The program first declares a structure named `Student` and then declares memory on heap of `Student` type using `calloc()*/`

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Student{
  int rollNum;
  char name[30];
}s1;
int main(){
  int size=5;
  struct Student *arrayOfStudent=(struct
Student*)calloc(size,sizeof(s1)*size); /*calloc() used
to allocate memory of Student type*/
```

Use of calloc() function(cont…)

```
if(arrayOfStudent==NULL)
  perror("calloc failed\n");
for(int i=0;i<size;i++){
  arrayOfStudent[i].rollNum=i+1;
  strcpy(arrayOfStudent[i].name,"Default");
}
printf("The elements of the array are:\n");
for(int i=0;i<size;i++){
  printf("Roll Number: %d",arrayOfStudent[i].rollNum);
  printf(", Name: %s\n",arrayOfStudent[i].name);
}
free(arrayOfStudent); //deallocating memory
return 0;}
```

Use of Calloc() function(cont…)

- Output of the above program is:

```
The elements of the array are:
Roll Number: 1, Name: Default
Roll Number: 2, Name: Default
Roll Number: 3, Name: Default
Roll Number: 4, Name: Default
Roll Number: 5, Name: Default
```

2D-Integer Arrays

- For declaring 2D integer arrays, we first declare an array of pointers on heap corresponding to the number of rows
- A double pointer, declared on stack, points to the array of pointers on heap
- After declaring array of pointers, we declare array(s) of integers on heap
- Each element in the array of pointers points to a different array on heap
- Let's write a program to understand this
- **Note:** Only double pointer is declared on stack

2D-Integer Arrays(cont…)

```
/* The program declares and uses a 2D array on heap according to
the number of rows and columns specified by the user*/
#include<stdio.h>
#include<stdlib.h>
void init(int**,int,int);
void print(int**,int,int);
int main(){
  int rows,cols;
  printf("Enter number of rows: ");
  scanf("%d",&rows);
  printf("Enter number of columns: ");
  scanf("%d",&cols);
```

2D-Integer Arrays(cont…)

```
/*first allocating an array of pointers equal to the number of
rows*/
  int** table=(int**)malloc(sizeof(int*)*rows);
/*Allocating an integer array equal to the number of cols*/
  for(int i=0;i<rows;i++)
    table[i]=(int*)malloc(sizeof(int)*cols);
  init(table,rows,cols);
  print(table,rows,cols);
  for(int i=0;i<rows;i++)
    free(table[i]);
  free(table);
  return 0;
}
```

2D-Integer Arrays(cont…)

```
void init(int** arr,int rows,int cols){
 printf("Enter %d elements of 2D array:\n“,rows*cols);
/*Using nested loop for traversing the array*/
  for(int i=0;i<rows;i++)
    for(int j=0;j<cols;j++)
      scanf("%d",&arr[i][j]);
}
void print(int** arr,int rows,int cols){
  printf("The elements are:\n");
  for(int i=0;i<rows;i++){
    for(int j=0;j<cols;j++)
      printf("%d\t",arr[i][j]);
    printf("\n");}}
```

2D-Integer Arrays

- Output of the above program is:

```
Enter number of rows: 4
Enter number of columns: 3
Enter elements of array
1 2 3 4 5 6 7 8 9 10 11 12
The elements are:
1   2   3
4   5   6
7   8   9
10  11  12
```

2D-Character Arrays

- To understand this, we consider a scenario in which we want to take number of names from the user
- We declare an array of character pointers equal to the number of names
- Then we allocate character arrays on heap equal to the length of name
- Each pointer in the array of pointers points to a character array
- Let’s understand this by writing a program

2D-Character Arrays(cont…)

```
/*Program declares a 2D character array on heap*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main(){
  int rows,cols;
  printf("Enter no. of rows: ");
  scanf("%d",&rows);
  printf("Enter max length of the name: ");
  scanf("%d",&cols);
  getchar();
```

2D-Character Arrays(cont…)

/*Allocating no. of rows equal to the number of names*/

```
  char** names=(char**)malloc(sizeof(char*)*rows);
```

/*Now allocating each character array equal to the max length of a name given*/

```
  for(int i=0;i<rows;i++)
    names[i]=(char*)malloc(sizeof(char)*cols);
  printf("Enter %d names:\n",rows);
  char* q;
  for(int i=0;i<rows;i++){
    fgets(names[i],cols,stdin);
    if((q=strchr(names[i],'\n'))!='\0')
      *q='\0';
  }
```

2D-Character Arrays(cont…)

```
  printf("The names entered are:\n");
  for(int i=0;i<rows;i++){
    fputs(names[i],stdout);
    printf("\n");
  }
/*freeing the memory on heap*/
  for(int i=0;i<rows;i++)
    free(names[i]);
  free(names);
  return 0;
}
```

2D-Character Arrays(cont…)

- Output of the above program is:

```
Enter no. of rows: 3
Enter max length of the name: 20
Enter 3 names:
Arif Butt
Kakamanna
Hadeed
The names entered are:
Arif Butt
Kakamanna
Hadeed
```

- There is a problem with this program, let's see this from another output

2D-Character Arrays(cont…)

- Output of the above program is:

```
Enter no. of rows: 3
Enter max length of the name: 10
Enter 3 names:
Muhammad Arif Butt
The names entered are:
Muhammad
Arif Butt
```

- You see that here when the length was 10, the program didn't produce correct results
- The reason is max length was 10 and the first name entered had more number of characters!

2D-Character Arrays(cont…)

- One solution is:

```
Enter no. of rows: 3
Enter max length of the name: 100  //large length
Enter 3 names:
Muhammad Arif Butt
Kakamanna
Hadeed
The names entered are:
Muhammad Arif Butt
Kakamanna
Hadeed
```

- You see, now its working OK
- But here, memory is being wasted, as the 2nd and the 3rd names are very small but they are too consuming 100 bytes of memory!

2D-Character Arrays(cont…)

- A better solution to this problem is that we use the code for variable length of name, we previously wrote
- Let's see this in a program

2D-Character Arrays(cont…)

/*The program is similar to the previous one but here each element in the array of pointers has its size according to the length of the name it is containing*/

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  int rows;
  printf("Enter number of names: ");
  scanf("%d",&rows);
  getchar();
  char** names=(char**)malloc(sizeof(char*)*rows);
  printf("Enter %d names, each on a new
line:\n",rows);
```

/*Now using that piece of code here*/

2D-Character Arrays(cont…)

```
for(int i=0;i<rows;i++){
    names[i]=(char*)malloc(sizeof(char)*1);
    names[i][0]='\0';
     char ch;
     int size=0;
     do{
          ch=getchar();
          if(ch=='\n')
              break;
          if(names[i][0]=='\0')
              names[i]=(char*)realloc(names[i],2);
          else
              names[i]=(char*)realloc(names[i],size+2);
```

2D-Character Arrays(cont…)

```
                    names[i][size++]=ch;
              }while(ch!='\n');
              names[i][size]='\0';
    }
    printf("The names entered are:\n");
    for(int i=0;i<rows;i++)
        printf("%s\n",names[i]);
    for(int i=0;i<rows;i++)
       free(names[i]);
    free(names);
    return 0;
}
```

2D-Character Arrays(cont…)

- Output of the above program is:

```
Enter number of names: 3
Enter 3 names, each on a new line:
Muhammmad Arif Butt
Kakamanna
Hadeed Ur Rehman
The names entered are:
Muhammmad Arif Butt
Kakamanna
Hadeed Ur Rehman
```

- You see that now name of any length can be entered and also space allocated for each name is according to its length

SUMMARY