C under Linux

Dr. Naeem Odat

Department of Computer and Communications Engineering

C - Pointers and malloc

C - Pointers and malloc

C - Pointers and malloc

Dynamic Memory Allocation

Write a c program which accepts the number of students in the class, accepts their marks obtained and print their progress card.

C - Pointers and malloc

Can I use array?

- Number of students is not known to the programmer.
- Every time the program is executed, the number of students will be different.

C - Pointers and malloc

Disadvantages of using array

- Array size has to be specified by the programmer.
- No provision to expand or shrink the array size.
- The memory area is occupied and cannot be released till the end of the program.

C - Pointers and malloc

Allocating Dynamic Memory

- Syntax:

  ```
  void* malloc(int bytes) ;
  ```

- Allocates an un-initialized block of memory from the **heap** and returns a pointer to the first byte.
- The pointer is usually cast and saved in a pointer to data of the desired type.
- A **NULL** is returned when insufficient heap space is available.

C - Pointers and malloc

Releasing Dynamic Memory

- Syntax:

```
void free(void* pointer2block) ;
```

- De-allocates the block and returns it to the heap for reuse.
- Do not use contents of block after release!
- Assign the pointer to NULL after a free (good programming practice)

C - Pointers and malloc

Using Dynamic Memory

```
#include<stdio.h>
#include<stdlib.h>
typedef ... ANYTYPE;
...
p = (ANYTYPE*) malloc(sizeof(ANYTYPE)); /* Memory Allocated */
if ( NULL == p){
    Error("Insufficient heap space");
    exit(0);
}
...
The object is initialized and used here via the pointer.

free(p);                                /* Memory Deallocated.*/
```

C - Pointers and malloc

Advantages and Disadvantages of Using Dynamic Memory

- Advantages:
  - Dynamic Allocation is done at run time.
  - Allocated memory can grow and shrink to fit changing data requirements.
  - We can allocate (create) additional storage whenever we need them.
  - We can de-allocate (free/delete) dynamic space whenever we are done with them.
  - Thus we can always have exactly the amount of space required - no more, no less.
- Disadvantages:
  - As the memory is allocated during run-time, it requires more time.
  - Memory needs to be freed by the user when done. This is important as it is more likely to turn into bugs that are difficult to find.

C - Pointers and malloc

Using **malloc**

- Example:

```
int *ip = NULL;
ip = malloc(10 * sizeof(int));
if(NULL == ip){
    /* Handle Error! */
}
```

- Options for handling error:
  - Abort.
  - Ask again.
  - Save user data.

C - Pointers and malloc

Prototypes

```
#include<stdlib.h>

void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
```

C - Pointers and malloc

What is the output?

```
#include<stdio.h>
int main(){
    int* pArr = NULL, nSz, nCnt;
    printf("Enter Size :");
    scanf("%d", &nSz);
    pArr = (int*) malloc( nSz * sizeof(*pArr));
    if (NULL == pArr){
        printf("Allocation Failed. Exiting!!!\n");
        exit(1);
    }
    for (nCnt = 0; nCnt < nSz; nCnt++){
        *pArr = 100 * (nCnt+1);
        printf("\n%d", *pArr);
    }
    free(pArr);
    return 0;
}
```

C - Pointers and malloc

Memory leaks

- Example:

```
void foo(void){
    char *ca = malloc(...);
    /* no free */
    return;
}
```

- Memory leaks occur when the programmer **loses track** of memory allocated by malloc or other functions that call malloc.

C - Pointers and malloc

What is the output?

```
#include<stdlib.h>
int main(void){
    int nCtr = 0;
    int* p = (int*) malloc( 10 * sizeof(int));
    ...
    free(p);

    for(i =0; i <10; i++)
        printf("\t%d", *(p+i));
    return(0);
}
```

- p is a dangling pointer.
- Assign the pointer to NULL after a free.
- Always check pointer for NULL before use.

C - Pointers and malloc

Dynamic Allocation, What Can Go Wrong?

- Allocate a block of memory and use the contents without initialization.
- Free a block but continue to use the contents.
- Allocate a block and lose it by losing the value of the pointer.
- Read or write beyond the boundaries of the block.
- FAIL TO NOTICE ERROR CONDITIONS.