EENG212 – Algorithms & Data Structures

Fall 07/08 – Lecture Notes # 5b

Outline

- **Dynamic Memory Allocation**
  - ***malloc( ), free( )* and *calloc( )* functions**
  - **Dynamically Allocated Arrays**

DYNAMIC MEMORY ALLOCATION

Dynamic allocation is the means by which a program can obtain memory while it is running.

- Pointers provide necessary support for C's powerful dynamic memory allocation.
- In general global variables are allocated storage at compile time. Local variables use the program stack. However neither global nor local variables can be added during program execution.
- Memory allocated by C's dynamic allocation functions come from the "heap": the region of free memory that lies between your program's permanent storage area and the stack.
- In general the heap contains a fairly large amount of free memory.

malloc( ) function

- *malloc( )* function dynamically allocates memory during the program. It is a library function included in the ***<stdlib.h>*** header file.
- *The **function prototype** is given below:*

```
void   * malloc(size_t number_of_bytes);
```

```
where, size_t is defined in header file as unsigned integer and
       number_of_bytes  is the amount of memory we wish to allocate
```

- After a successful call, ***malloc( )*** returns a pointer to the first byte of the region of memory. If memory is not enough the function returns a ***NULL***.

Ex: Code fragments below allocate 1000 bytes of ***contigious*** memory.

```
char  *p;
p =(char *)malloc(1000);    //(char *) forces void pointer to
                            //become a character pointer
```

Ex: Code below allocates space for 50 integers.

```
int *p;
p =(int *)malloc(50*sizeof(int));//(int *) forces void pointer
                                 // to become a integer pointer
```

Since heap is not infinite you ***must*** check the value returned by malloc( ) to make sure it is not null. You can use the following code fragment for this purpose.

```
if( !(p=(int *)malloc(100)))
{
      printf("Out of memory \n");
            exit(1);
      }
```

- ***free( )* function**
  - ***free( )*** function is the opposite of malloc( ). It returns previously allocated memory to the system. It has the prototype below:

```
void free(void *p);
```

  - It is critical that you never call free( ) with an invalid argument. This will destroy the free list.

- ***calloc( )* function**
  - *calloc( )* function allocates an amount of memory equal to num*size. That is *calloc( )* allocates enough memory for an **array** of num objects, each object being size bytes long. Memory allocated by *calloc( )* is released by *free( )* function.
  - The name calloc comes from " **c**ontiguous **alloc**ation".
  - *calloc( )* has the following function prototype.

```
void *calloc(size_t n, size_t number_of_bytes);
```

where, *size_t* is defined in header file as unsigned integer.
n is the number of object to be used in the dynamic array and
number_of_bytes is the size of each object in the array.

Ex: Code fragments below dynamically allocates 100 elements to an integer array.

```
int *p;
p = (int *) calloc(100,sizeof(int));//(int *) forces the void
                                    // pointer to become an int pointer
```

Dynamically Allocated Arrays

- Sometimes you will want to allocate memory using *malloc*( ) or *calloc*( ) but operate on that memory as if it were an array, using an array index.
- Since any pointer may be indexed as if it were a single-dimensional array this is not a problem.

Ex: In this program pointer s is used in the call to gets() and then indexed as an array to print the string backwards.

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

void main(void)
{
      char  *s;
      register   int   t;
      s= (char *)malloc(80); // (char *) forces the void * to
      if(!s)                 // an char *.
      {
            printf("Memory request failed \n");
            exit(1);
}
gets(s);  //used for passing arrays into functions
           //standard library function

for(t=strlen(s)-1; t>=0; t--)
      putchar(s[t]);

free(s);
}
```

Ex: The following program asks the user to specify the size of the dynamic integer array to be used, and then displays the array elements entered by the user in the reverse order.

```
#include<stdio.h>
#include<stdlib.h>
void main(void)
{
      int *aptr, i, n;
      printf("Specify the size of you integer array\n");
      scanf("%d",&n);

            aptr= (int *)calloc(n, sizeof(int)); // (int *) forces
                                 // the void * to int *.

      printf("The array is ready Enter %d numbers one by one\n", n);
      for(i=0;i<n;i++)
            scanf("%d",&aptr[i]);

      for(i=n-1;i>=0;i--)
            printf("%d,",aptr[i]);
      free(aptr);
}
```