Programming Fundamentals CT-175

Pointers & Dynamic Memory Allocation

Objectives

The objective of this lab is to familiarize students with dynamic allocation of memory. By the end of this lab students will be able to allocate memory using malloc() and calloc() functions and free it using free() function.

Tools Required

DevC++ IDE

Course Coordinator –
Course Instructor –
Lab Instructor –
Department of Computer Science and Information Technology
NED University of Engineering and Technology

Introduction

Creating and maintaining dynamic data structures requires dynamic memory allocation—the ability for a program **to obtain more memory space at execution time to hold new nodes, and to release space no longer needed.**

sizeof Operator

There is a useful operator in C called the **sizeof** operator. You can use it to determine the size of a particular variable in memory. You can also use it to determine the size of a type. We will use this operator to dynamically allocate memory space.
The number of elements in an array also can be determined with sizeof. For example, consider the following array definition:

```
double real[ 22 ];
```

Variables of type double normally are stored in 8 bytes of memory. Thus, array real contains a total of 176 bytes. To determine the number of elements in the array, the following expression can be used:

```
sizeof( real ) / sizeof( real[ 0 ] );
```

Example 01: sizeof returns size of each data type

```
#include <stdio.h>
int main( ){
   printf("%lu\n", sizeof(char));
   printf("%lu\n", sizeof(int));
   printf("%lu\n", sizeof(float));
   printf("%lu", sizeof(double));
   return 0;
}
```

malloc( ) Function

The name "malloc" stands for memory allocation. The malloc() function **reserves a block of memory of the specified number of bytes** and it returns a pointer of void which can be casted into pointers of any form. Function malloc is normally used with the sizeof operator. For example,

```
ptr = (float*) malloc(100 * sizeof(float));
```

The above statement allocates 400 bytes of memory. It's because the size of float is 4 bytes and the pointer ptr holds the address of the first byte in the allocated memory.

The expression results in a NULL pointer if the memory cannot be allocated.

Example 02: Taking number of elements of array as user input and initializing it using malloc( ).

```
#include <stdio.h>
#include <stdlib.h>
int main( ){
   // This pointer will hold the base address of the block created
   int *ptr;
   int n, i;

   // Get the number of elements for the array
   printf("Enter number of elements:");
   scanf("%d",&n);
   printf("Entered number of elements: %d\n", n);
```

```
  // Dynamically allocate memory using malloc()
  ptr = (int*)malloc(n * sizeof(int));

  // Check if the memory has been successfully allocated by malloc or not
  if (ptr == NULL) {
    printf("Memory not allocated.\n");
    exit(0);
  }
  else {
    // Memory has been successfully allocated
    printf("Memory successfully allocated using malloc.\n");

    // Get the elements of the array
    for (i = 0; i < n; ++i) {
      ptr[i] = i + 1;
    }
    // Print the elements of the array
    printf("The elements of the array are: ");
    for (i = 0; i < n; ++i) {
      printf("%d, ", ptr[i]);
    }
  }
  return 0;
}
```

calloc( ) Function

“calloc” or “contiguous allocation” method in C is used to dynamically allocate the **specified number of blocks of memory** of the specified type. it is very much similar to malloc() but has two differences and these are:

- It initializes each block with a default value ‘0’.
- It has two parameters or arguments as compare to malloc().

Consider the following line of code:

```
ptr = (float*) calloc(25, sizeof(float));
```

This statement allocates contiguous space in memory for 25 elements each with the size of the data type float.

Example 03: Taking number of elements of array as user input and initializing it using calloc().

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

int main(){
    // This pointer will hold the base address of the block created
    int *ptr;
    int n, i;

    // Get the number of elements for the array
    n = 5;
    printf("Enter number of elements: %d\n", n);
    // Dynamically allocate memory using calloc()
    ptr = (int*)calloc(n, sizeof(int));

    // Check if the memory has been successfully allocated by calloc or not
    if (ptr == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {
        // Memory has been successfully allocated
        printf("Memory successfully allocated using calloc.\n");

        // Get the elements of the array
        for (i = 0; i < n; ++i) {
            ptr[i] = i + 1;
        }
        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", ptr[i]);
        }
    }
    return 0;
}
```

free() Function

Dynamically allocated memory created with either calloc( ) or malloc( ) doesn't get freed on their own. You must explicitly use free( ) to release the space. Following is its syntax:

```
free(ptr);
```

Example 04: Allocating memory using malloc( ) and calloc( ) and then deallocating it using free( ).

```
#include<stdio.h>
int main( ){
   // These pointers will hold the base address of the blocks created
   int *ptr, *ptr1;

   // Dynamically allocate memory using malloc()
   ptr = (int*)malloc(5 * sizeof(int));

   // Dynamically allocate memory using calloc()
   ptr1 = (int*)calloc(5, sizeof(int));

   // Check if the memory has been successfully
   // allocated by malloc or not
   if (ptr == NULL || ptr1 == NULL) {
      printf("Memory not allocated.\n");
      exit(0);
   }
   else {
      // Memory has been successfully allocated
      printf("Memory successfully allocated using malloc.\n");
```

```
    // Free the memory
    free(ptr);
    printf("Malloc Memory successfully freed.\n");

    // Memory has been successfully allocated
    printf("\nMemory successfully allocated using calloc.\n");

    // Free the memory
    free(ptr1);
    printf("Calloc Memory successfully freed.\n");
  }
  return 0;
}
```

Memory Leaks

A memory leak is an issue that occurs when all pointers to a block of dynamically-allocated memory are lost before the block of memory is freed. Memory allocated on the heap persists beyond function calls and the computer has no way of knowing when a block of memory is not needed any more. In Java, this is handled by garbage collection, but there are no such niceties in C. The programmer must explicitly release a block of memory back into the heap using the free() command when it is done being used.

Exercise

1. Write a program that does the following:
   a. Ask the user to type the size of the array.
   b. Use malloc or calloc to create an integer array of that size.
   c. Use the function read to read the numbers.
   d. Display the sum and average or these numbers. Then display the array sorted.
      ✓ Show 2 numbers after the floating point in the average.
   e. Free the allocated memory.
2. Write a program that ask the user to enter the total 'N' no of characters in user's name {First Name + Last Name} to create a dynamic array of characters. After create a dynamic array of that 'N' no of characters using malloc or calloc function. Finally copy your full name in it that has already been taken from the user before

   Dynamic Array = "Muhib Ahmed";
3. Using above question (2), resize that dynamic array of character and append the array with your studentId. That student id must be taken input from the user.

   DynamicArray = "Muhib Ahmed"; // Before
   DynamicArray = "K211234 Muhib Ahmed"; // After the text append