CSE115L – Programming Language I Lab

Lab - 29

Dynamic Memory Allocation

In this lab, we will learn about dynamic memory allocation in C. An array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it. Sometimes the size of the array you declared may be insufficient. To solve this issue, you can allocate memory manually during run-time. This is known as dynamic memory allocation in C programming. C provides 4 library functions for this purpose. These are defined in the <stdlib.h> header file. The functions are:

1. malloc(): The "malloc" or "memory allocation" function is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form. The general syntax is as follows.

   ptr = (cast-type*) malloc(byte-size);

   If the space is insufficient, allocation fails and returns a NULLpointer.
2. calloc(): The "calloc" or "contiguous allocation" function is used to dynamically allocate the specified number of blocks of memory of the specified type. It initializes each block with a default value '0'. The general syntax is as follows.

   ptr = (cast-type*) calloc(n, element-size);

   If the space is insufficient, allocation fails and returns a NULLpointer.
3. free(): The "free" function is used to dynamically de-allocate the memory. The memory allocated using functions malloc() and calloc() are not de-allocated on their own. Hence the free() function is used, whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it. The general syntax is as follows.

   free(ptr);
4. realloc(): The "realloc" or "re-allocation" function is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc() or calloc() is insufficient, realloc()can be used to dynamically re-allocate memory. The general syntax is as follows.

   ptr = realloc(ptr, newSize);

   If the space is insufficient, allocation fails and returns a NULLpointer.

Example 1: Usage of `malloc()`, `calloc()`, `free()` and `realloc()`.

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

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

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

	// Dynamically allocate memory using calloc() or malloc()
	ptr = (int*)malloc(size * sizeof(int));
	// Or you can use calloc()
	//ptr = (int*)calloc(size, sizeof(int));

	// Check if the memory has been successfully allocated or not
	if (ptr == NULL) {
		printf("Memory cannot be allocated.\n");
	}
	else {

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

		// Get the elements of the array
		for (i = 0; i < size; ++i)
			ptr[i] = i + 1;

		// Print the elements of the array
		printf("The elements of the array are: ");
		for (i = 0; i < size; ++i)
			printf("%d, ", ptr[i]);

		// Get the new size for the array
		printf("\n\nEnter the new size of the array: ");
		scanf("%d", &newSize);

		// Dynamically re-allocate memory using realloc()
		ptr = realloc(ptr, newSize * sizeof(int));

		if(ptr != NULL){
		// Memory has been successfully allocated
		printf("Memory successfully re-allocated using realloc.\n");

		// Get the new elements of the array
		for (i = size; i < newSize; ++i)
		    ptr[i] = i + 1;

		// Print the elements of the array
		printf("The elements of the array are: ");
		for (i = 0; i < newSize; ++i)
		    printf("%d, ", ptr[i]);
	  }
	  else printf("Memory cannot be reallocated.\n");

		free(ptr);
	}

	return 0;
}
```