/\* Singly Linked List with all operations in C\*/

/\* Singly Linked List Example - Insert, Delete, Display and Count Operations\*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

struct node {

int value;

struct node \*next;

};

void insert();

void display();

void delete();

int count();

typedef struct node DATA\_NODE;

DATA\_NODE \*head\_node, \*first\_node, \*temp\_node = 0, \*prev\_node, next\_node;

int data;

int main()

{ int option = 0;

printf("Singly Linked List - All Operations\n");

while (option < 5) {

printf("\nOptions\n");

printf("1 : Insert into Linked List \n");

printf("2 : Delete from Linked List \n");

printf("3 : Display Linked List\n");

printf("4 : Count Linked List\n");

printf("Others : Exit()\n");

printf("Enter your option:");

scanf("%d", &option);

switch (option)

{ case 1: insert();

break;

case 2: delete();

break;

case 3: display();

break;

case 4: count();

break;

default: break;

}

}

return 0;

}

void insert()

{ printf("\nEnter Element for Insert Linked List : \n");

scanf("%d", &data);

temp\_node = (DATA\_NODE \*) malloc(sizeof (DATA\_NODE));

temp\_node->value = data;

if (first\_node == 0) {

first\_node = temp\_node;

} else {

head\_node->next = temp\_node;

}

temp\_node->next = 0;

head\_node = temp\_node;

fflush(stdin);

}

void delete()

{ int countvalue, pos, i = 0;

countvalue = count();

temp\_node = first\_node;

printf("\nDisplay Linked List : \n");

printf("\nEnter Position for Delete Element : \n");

scanf("%d", &pos);

if (pos > 0 && pos <= countvalue)

{ if (pos == 1) {

temp\_node = temp\_node -> next;

first\_node = temp\_node;

printf("\nDeleted Successfully \n\n");

}

else

{ while (temp\_node != 0)

{ if (i == (pos - 1))

{ prev\_node->next = temp\_node->next;

if(i == (countvalue - 1))

{ head\_node = prev\_node;

}

printf("\nDeleted Successfully \n\n");

break;

}

else

{ i++;

prev\_node = temp\_node;

temp\_node = temp\_node -> next;

}

}

}

}

else

printf("\nInvalid Position \n\n");

}

void display()

{ int count = 0;

temp\_node = first\_node;

printf("\nDisplay Linked List : \n");

while (temp\_node != 0)

{ printf("# %d # ", temp\_node->value);

count++;

temp\_node = temp\_node -> next;

}

printf("\nNo Of Items In Linked List : %d\n", count);

}

int count()

{ int count = 0;

temp\_node = first\_node;

while (temp\_node != 0)

{ count++;

temp\_node = temp\_node -> next;

}

printf("\nNo Of Items In Linked List : %d\n", count);

return count;

}

Note:-

calloc() versus malloc()

The name **malloc** and calloc() are library functions that allocate memory dynamically. It means that memory is allocated during runtime(execution of the program) from heap segment.

* **Initialization:** malloc() allocates memory block of given size (in bytes) and returns a pointer to the beginning of the block. malloc() doesn’t initialize the allocated memory. If we try to acess the content of memory block then we’ll get garbage values.

|  |
| --- |
| void \* malloc( size\_t size ); |

* calloc() allocates the memory and also initializes the allocates memory block to zero. If we try to access the content of these blocks then we’ll get 0.

|  |
| --- |
| void \* calloc( size\_t num, size\_t size ); |

* **Number of arguments:** Unlike malloc(), calloc() takes two arguments:  
  1) Number of blocks to be allocated.  
  2) Size of each block.
* **Return Value:** After successfull allocation in malloc() and calloc(), a pointer to the block of memory is returned otherwise **NULL** value is returned which indicates the failure of allocation.

For instance, If we want to allocate memory for array of 5 integers, see the following program:-

|  |
| --- |
| // C program to demonstrate the use of calloc()  // and malloc()  #include <stdio.h>  #include <stdlib.h>    int main()  {     int \*arr;       // malloc() allocate the memory for 5 integers     // containing garbage values     arr = (int \*)malloc(5 \* sizeof(int)); // 5\*4bytes = 5 bytes       // Deallocates memory previously allocated by malloc() function     free( arr );       // calloc() allocate the memory for 5 integers and     // set 0 to all of them     arr = (int \*)calloc(5, sizeof(int));       // Deallocates memory previously allocated by calloc() function     free(arr);       return(0);  } |