

# Linked List

☰ Tags	
🕒 Created time	@November 12, 2024 11:04 AM
☑ Reviewed	<input type="checkbox"/>

## Why do we need linked lists?

Sometimes we need to use linked lists to optimize the usage of resources. Usually not because you don't have enough memory, but because the device doesn't have the chunk of memory together that you are looking for.

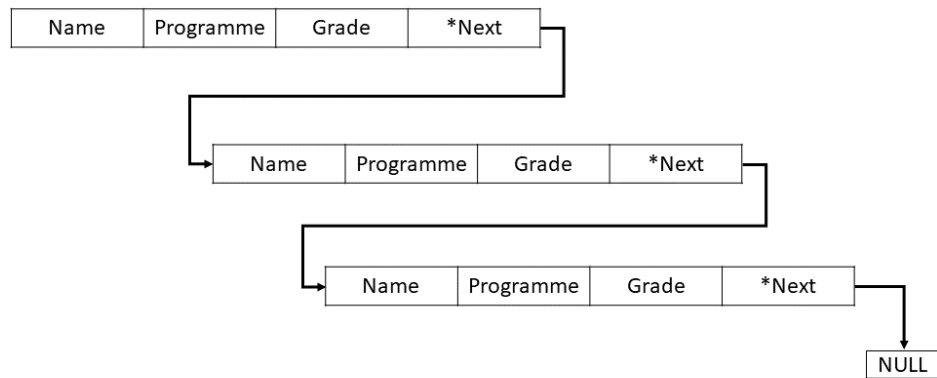
## Singly Linked List

**?** Write a program to handle a list of students, each student contains the following information:

- Name (an array of 20 characters)
- Programme (an array of 20 characters)
- Grade (float)

The program also provide some functions such as print the student lists, sorting the list based on grades, calculate the average of grades.

You have dealt with such a problem by using an array of the structure, i.e. something like `Student student[50]` to store the information. A drawback of this is that you have to ensure that a block of memory is available to be allocated to store the data. What if the memory is fragmented and it is not possible to allocate a block of memory for data storage? Let's use linked list as follows.



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

/*Declare a structure of Student */
typedef struct Student Student;

struct Student {
    char name[20];
    char programme[20];
    float grade;
    Student *next;
};

/*Functional prototype */
Student* get_students(); /* You may notice that the function get
void printStudentList(Student *start); /*to print the student l:

int main()
{
```

```

Student *start = NULL;

start = get_students();
printStudentList(start);
return 0;
}

Student* get_students() /* This mean that the function will return a pointer to a Student struct */
{
    Student *current, *first; /*declare two pointers */
    int selection; /* to ask if the user keep entering new students */

    first = (Student*)calloc(1, sizeof(Student)); /*create the first node */
    current = first; /*Now the current node is also the first node */

    /*fill data for the first node */
    printf("Student name: \n");
    scanf("%s", current->name);
    printf("Programme: \n");
    scanf("%s", current->programme);
    printf("Grade: \n");
    scanf("%f", &current->grade);

    printf("Add more student? (1=Y, 0 = N): \n");
    scanf("%d", &selection);

    /*create the following nodes until the user select No */
    while(selection) //while selection is 1 (Yes)
    {
        /* allocate node and change the current point */
        current->next = (Student*)calloc(1, sizeof(Student));
        current = current->next;

        /*fill the new node */
        printf("Student name: \n");

```

```

        scanf("%s", current->name);
        printf("Programme: \n");
        scanf("%s", current->programme);
        printf("Grade: \n");
        scanf("%f", &current->grade);

        printf("Add more student? (1=Y, 0 = N): \n");
        scanf("%d", &selection);
    }
    current->next = NULL; /* in case the last node */
    return first; /* return the address of the first node */
}

/* This is the function to display the list of the students. Pay
void printStudentList(Student *start)
{
    int count = 0;
    Student* p = NULL;
    for(p = start; p != NULL; p = p->next)
    {
        ++count;
        printf("Student #%d: ", count);
        printf("%s, %s, %.2f\n", p->name, p->programme, p->grade);
    }
}

```

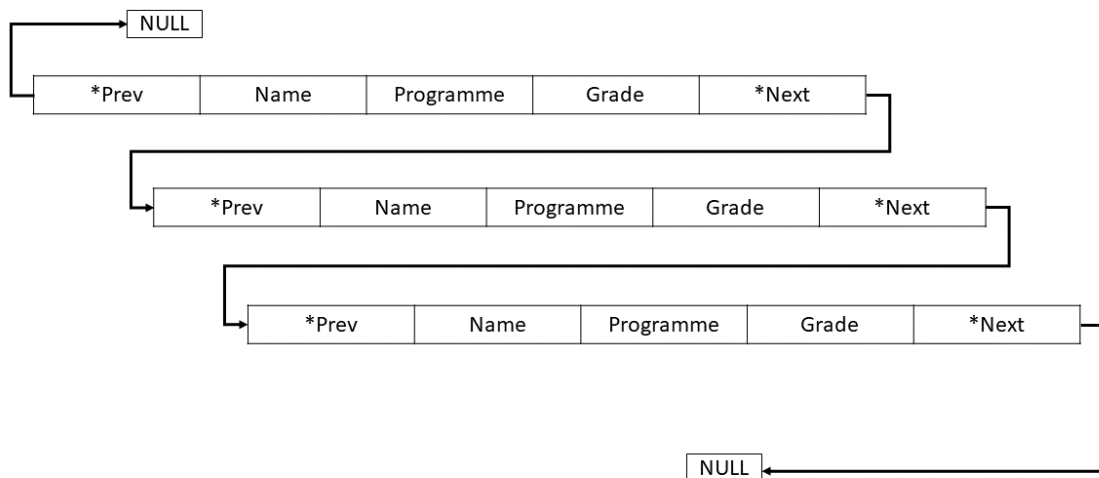
- Linked lists are invaluable in applications in which you need to process an unknown number of structures, such as you have here.
- The main advantages of a linked list relate to memory usage and ease of handling. You occupy only the minimum memory necessary to store and process the list. Although the memory used may be fragmented, you have no problem progressing from one structure to the next. As a consequence, in a practical situation in which you may need to deal with several different types

of objects simultaneously, each can be handled using its own linked list, with the result that memory use is optimized.

- Looking back at the code above, we have repeated the input process twice (for the first node and the following nodes separately). Can you re-write the program above without doing this?

## Doubly Linked Lists

A disadvantage of a singly linked lists is that you can only go forward, you cannot go backwards. A small modification of adding a previous pointer creates a doubly linked list, which will allow you to go through a list in either direction.



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

/*Declare a structure of Student */
typedef struct Student Student;
```

```

struct Student {
    char name[20];
    char programme[20];
    float grade;
    Student *next;
    Student *prev;
};

/*Functional prototype */
Student* get_students(); /* You may notice that the function get
void printStudentListInReverse(Student *last); /*to print the st

int main()
{
    Student *start = NULL;
    start = get_students();
    printStudentListInReverse(start);
    return 0;
}

Student* get_students() /* This mean that the function will retu
{
    Student *current, *first, *prev; /*declare two pointers */
    int selection; /* to ask if the user keep entering new stud

    first = (Student*)calloc(1,sizeof(Student)); /*create the fi
    current = first; /*Now the current node is also the first no

    /*fill data for the first node */
    printf("Student name: \n");
    scanf("%s", current->name);
    printf("Programme: \n");
    scanf("%s", current->programme);
    printf("Grade: \n");

```

```

scanf("%f", &current->grade);
current->prev = NULL;

printf("Add more student? (1=Y, 0 = N): \n");
scanf("%d", &selection);

/*create the following nodes until the user select No */
while(selection) //while selection is 1 (Yes)
{
    /* allocate the new node */
    current->next = (Student*)calloc(1, sizeof(Student));

    /*get the previous node before move the current pointer
    prev = current;

    /* move the current pointer to the next node */
    current = current->next;

    /*fill the new node */
    printf("Student name: \n");
    scanf("%s", current->name);
    printf("Programme: \n");
    scanf("%s", current->programme);
    printf("Grade: \n");
    scanf("%f", &current->grade);
    current->prev = prev;

    printf("Add more student? (1=Y, 0 = N): \n");
    scanf("%d", &selection);
}

current->next = NULL; /* in case the last node */
return current; /* return the address of the last node */
}

// allows printing in reverse simply
void printStudentListInReverse(Student *last)

```

```

{
    int count = 0;
    Student* p = NULL;
    for(p = last; p != NULL; p = p->prev)
    {
        ++count;
        printf("Student #%d: ", count);
        printf("%s, %s, %.2f\n", p->name, p->programme, p->grade);
    }
}

```

- Using a doubly linked list (using pointers at the end) makes our data structure more useful to create other methods for inserting and removing items of the list, which cannot be done in a fixed array.
- It is straightforward if the item is at the beginning or end of the list, but if it is a distinct item, then we need to iterate until finding it (we must iterate almost the complete list in the worst scenario).
- Another disadvantage is to have more significant memory footprint for handling the items, and again, it is slower by not having random (direct) access to the items.

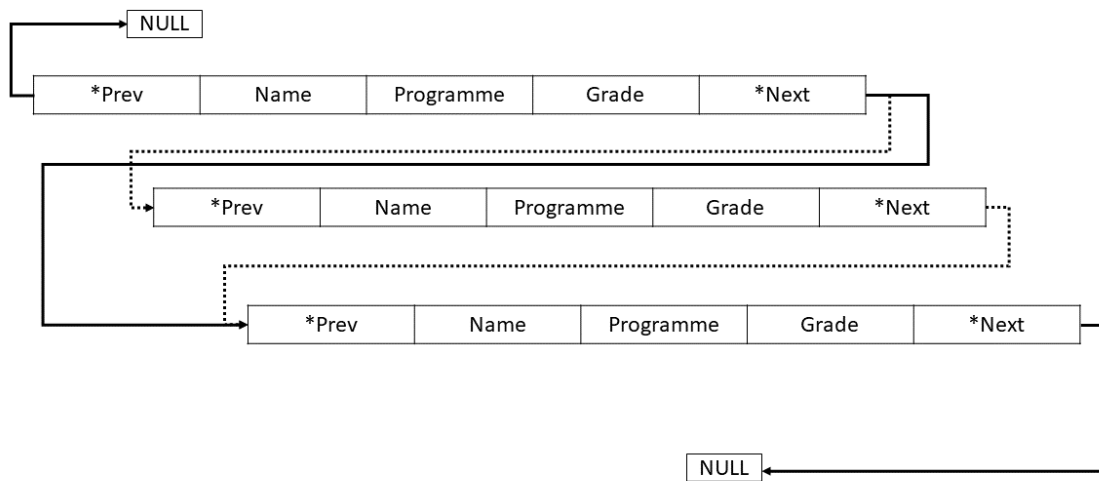
## Remove an Item from the list

### Steps:

1. Find the target item
2. Link the previous item with the next item
3. Release the memory of the current item.

View the diagram below:





```

void deleteStudentByName(char* name, Student *first)
{
    Student* p = first;
    Student *temp = NULL;

    for(p = first; p != NULL; p = p->next)
    {
        printf("Check %s ...\n", p->name );
        if(strcmp(p->name,name) == 0)
        {
            /* link the previous item with the next item */
            p->prev->next = p->next;
            p->next->prev = p->prev;
            temp = p->prev; /* this is used for marking the new
            free(p); /*delete the item */
            p = temp; /* new position of p */
        }
        /* then p will move to the next node from here before st
  
```

```
    }  
  
}
```

The above code will only work if the to-be deleted node is in the middle of the linked list. It will not work if the node to be deleted is the first or last.

## Code for Resolving This