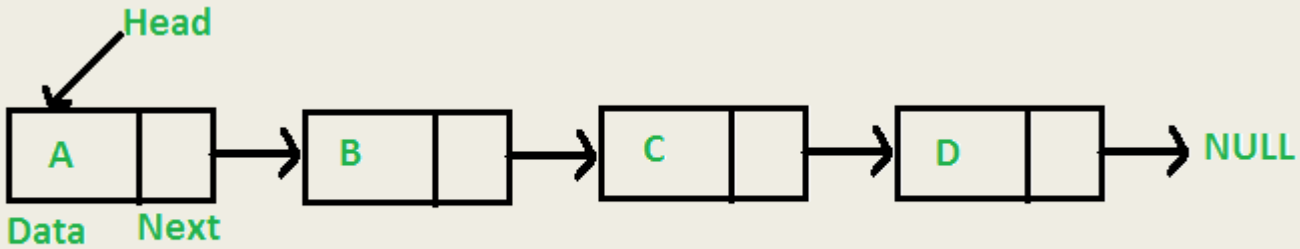
LINKED LIST

(Part 1)

Linked List Data Structure

- A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.



- In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

https://www.geeksforgeeks.org/data-structures/linked-list/#singlyLinkedList

Why Linked List?

Advantages of Linked Lists

- They are a dynamic in nature which allocates the memory when required.
- Linked List can grow and shrink during run time.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed

Disadvantages of Linked List

- The memory is wasted as pointers require extra memory for storage.
- No element can be accessed randomly; it has to access each node sequentially.
- Reverse Traversing is difficult in linked list.

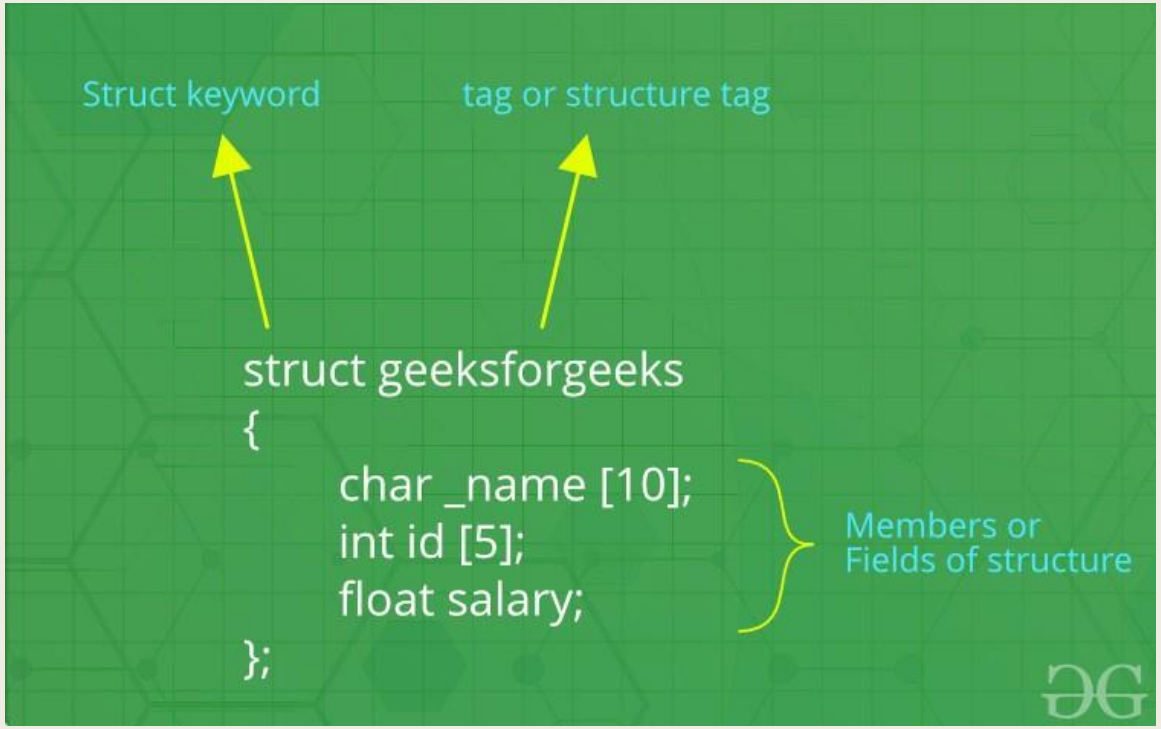
https://www.quora.com/What-are-the-advantages-and-disadvantages-of-linked-list

Prerequisites for learning Linked list…

- Structures
- Pointers

What is a structure?

- A structure is a user-defined data type in C/C++.
- A structure creates a **data type** that can be used to group items of possibly different types into a single type.



https://www.geeksforgeeks.org/structures-in-cpp/

How to create a structure?

- The ‘struct’ keyword is used to create a structure. The general syntax to create a structure is as shown below:

```
struct structureName{
    member1;
    member2;
    member3;
    .
    .
    .
    memberN;
};
```

How to declare structure variables?

```
// A variable declaration with structure declaration.
struct Point
{
    int x, y;
} p1;  // The variable p1 is declared with 'Point'


// A variable declaration like basic data types
struct Point
{
    int x, y;
};

int main()
{
    struct Point p1;  // The variable p1 is declared like a normal variable
}
```

- Note: In C++, the ‘struct’ keyword is optional before in declaration of a variable. In C, it is mandatory.

How to initialize structure members?

- Structure members **cannot be** initialized with declaration. For example the following C program fails in compilation.

```
struct Point
{
   int x = 0;  // COMPILER ERROR:  cannot initialize members here
   int y = 0;  // COMPILER ERROR:  cannot initialize members here
};
```

- The reason for above error is simple, when a datatype is declared, no memory is allocated for it. Memory is allocated only when variables are created.

How to initialize structure members? (Cont…)

- Structure members **can be** initialized using curly braces '{}'.
- For example

```
struct Point
{
    int x, y;
};

int main()
{
    // A valid initialization. member x gets value 0 and y
    // gets value 1.  The order of declaration is followed.
    struct Point p1 = {0, 1};
}
```

How to access structure elements?

- Structure members are accessed using dot (.) operator.

```c
#include <stdio.h>

struct Point {
    int x, y;
};

int main()
{
    struct Point p1 = { 0, 1 };

    // Accessing members of point p1
    p1.x = 20;
    printf("x = %d, y = %d", p1.x, p1.y);

    return 0;
}
```

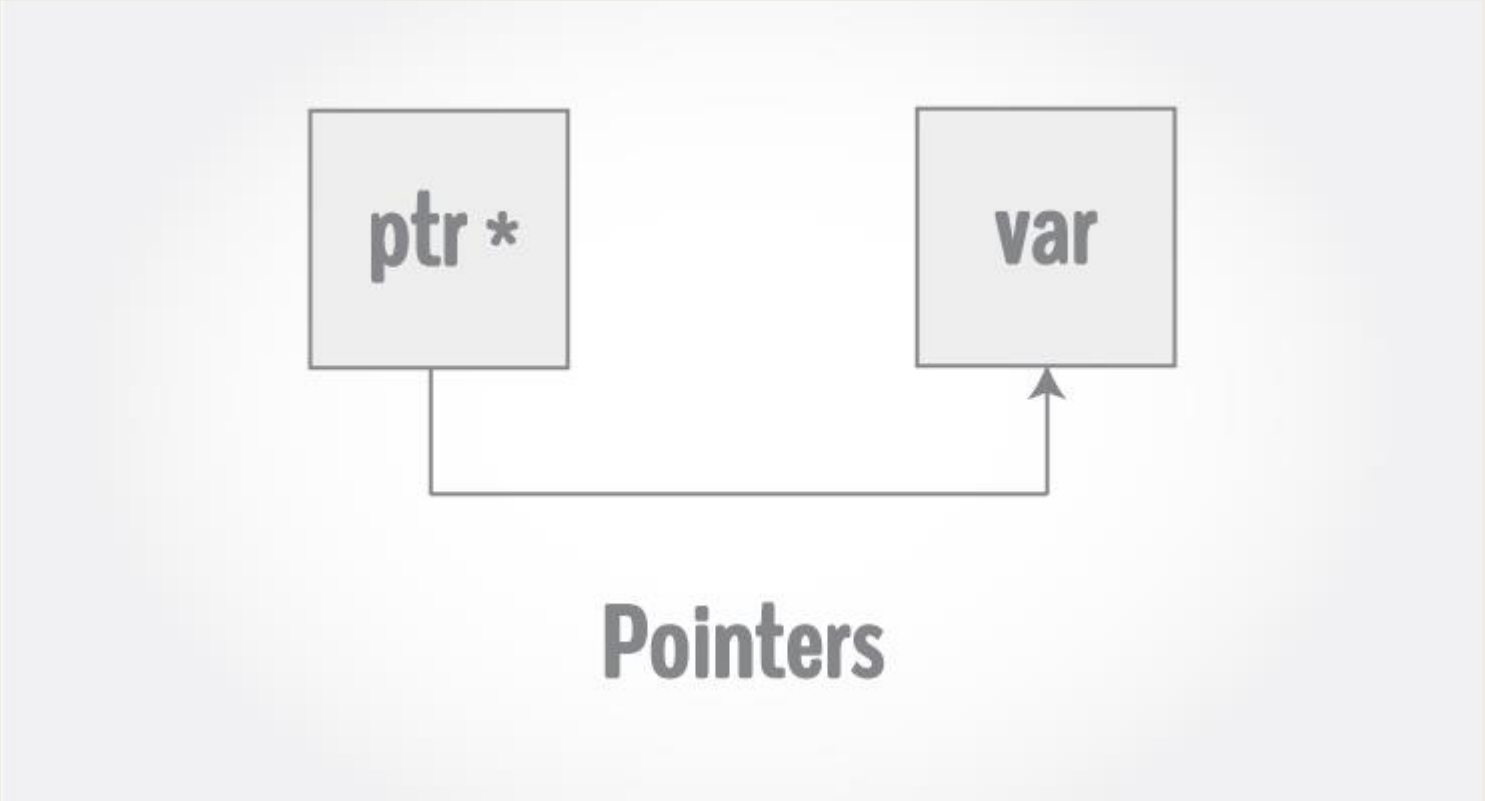
Problem for practice

- Write a program using structure to generate the result of your class using the steps below.
  - Declare a structure where there will be two string for first and last name.
  - Declare another structure for storing the student info. Like student id, two subject marks and previously declared name structure. (nested structure)
  - Take input for 10 students in the student info structure (array of structure).
  - Now prepare and print the result of the 10 students on the basis of subject 1.
  - That is, the student who got highest marks in subject 1 will be at the top and the one with lowest will be at the bottom.
  - Also if two or more students get same marks in subject 1, then arrange their position on the basis of subject 2. Among them the one who got more marks in subject 2 will be listed before others.

```
struct nameStruct {
    char fName[10];
    char lName[10];
};

struct studentInfo{
    int id;
    float sub1No, sub2No;
    struct nameStruct name;
};
```

Pointers

- Pointers are variables that store address of another variables



Pointers (Cont...)

- To understand pointers, you should first know **how data is stored** on the computer.
- Each variable you create in your program is assigned a location in the computer's memory.
- The value the variable stores is actually stored in the location assigned.
- To know where the data is stored, C/C++ has an & operator.
- The **&** (reference) operator gives you the address occupied by a variable.

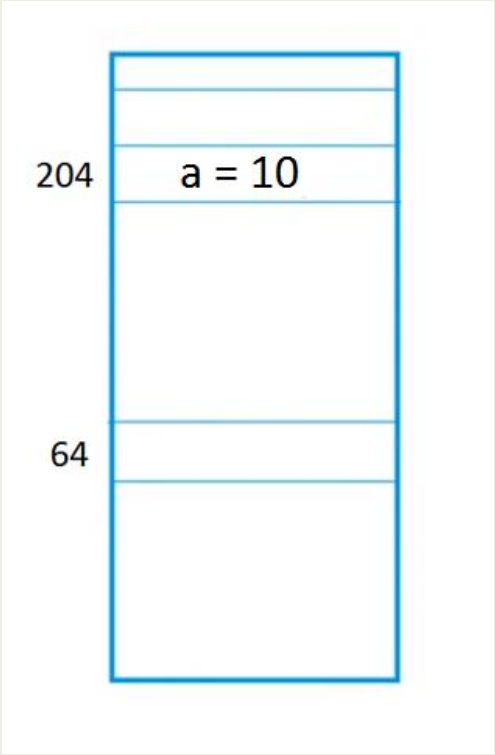
```c
#include <stdio.h>

int main()
{
    int x = 10;

    printf("Value of x = %d\n", x);
    printf("Address of x = %p\n", &x);

    return 0;
}
```

Pointers (Cont...)

- C/C++ gives you the power to manipulate the data in the computer's memory directly.
- You can assign and de-assign any space in the memory as you wish. This is done using Pointer variables.

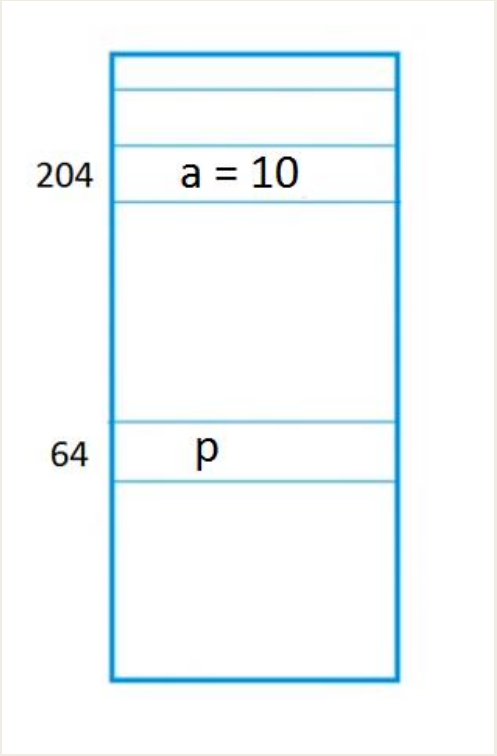
Pointers (Cont…)

- What if when we write
  - *int a = 10;*



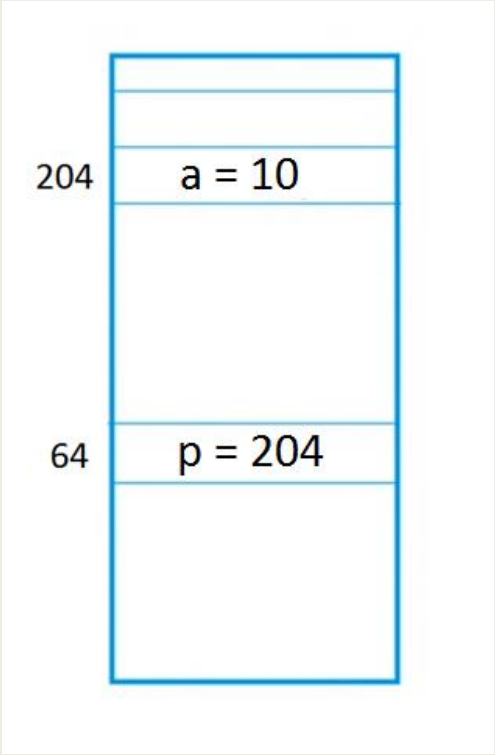
Pointers (Cont…)

- Working with pointers
  - *int *p;*



Pointers (Cont…)

- Working with pointers
  - *int *p;*
  - *p = &a;*



Pointers (Cont...)

```c
#include <stdio.h>

int main()
{
    int a = 10;
    int *p; //integer pointer p

    p = &a; //referencing or address of a

    printf("Value of x is %d\n", a);
    printf("Address of x is %p\n", &a);
    printf("*p = %d\n", *p); //dereferencing or content of p
    printf("Value of p is %p\n", p);

    return 0;
}
```

Pointer Applications in C Programming

- Passing Parameter by Reference
- Accessing Array element
- Dynamic Memory Allocation
- Passing Strings to function
- Provides effective way of implementing the different data structures such as tree, graph, linked list

http://www.c4learn.com/c-programming/c-pointer-applications/

Passing Parameter by Reference

```
void interchange(int *num1,int *num2)
{
    int temp;
    temp  = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

Accessing Array element

```
int main()
{
int a[5] = {1,2,3,4,5};
int *ptr;
ptr = a;

for(i=0;i<5;i++) {
 printf("%d",*(ptr+i));
}

return(0);
}
```

Dynamic Memory Allocation

```c
int main()
{
    int *arr;
    int i,n;

    printf("Enter n: ");
    scanf("%d",&n);

    arr = (int *)malloc(sizeof(int)*n);

    for(i=0; i<n; i++)
    {
        printf("Enter no %d: ",i+1);
        scanf("%d", &arr[i]);
        printf("\n");
    }
    printf("Your list: ");
    for(i=0; i<n; i++)
    {
        printf(" %d ", arr[i]);
    }
    free(arr);
    return 0;
}
```

Some functions

malloc()

- void * malloc (size_t size)
- Input: This is the size of the memory block, in bytes.
- Return Value: This function returns a pointer to the allocated memory, or NULL if the request fails.

free()

- void free(void *ptr)
- Input: This is the pointer to a memory block previously allocated with malloc() to be deallocated.
- Return Value: This function does not return any value.

Null Pointer

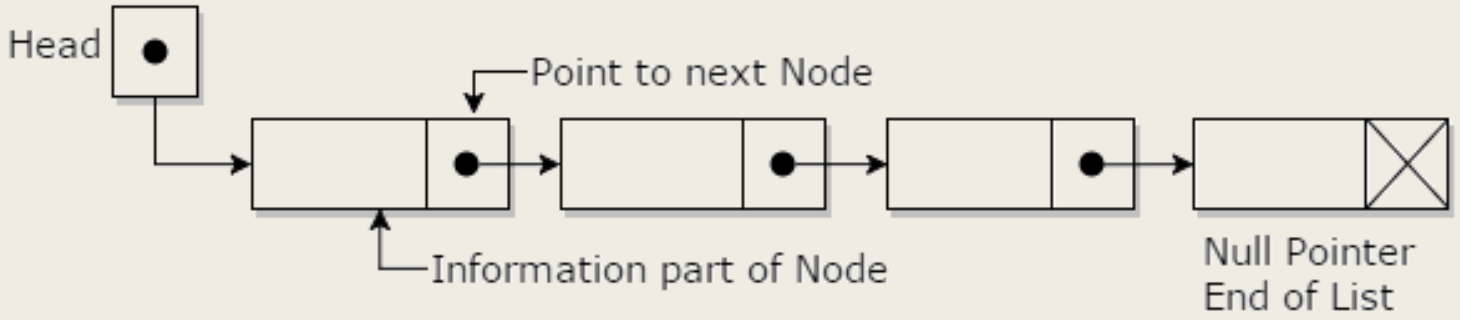
- To initialize a pointer variable when that pointer variable isn’t assigned any valid memory address yet.

Future topics (Self Study)

- Pointer types, void pointers
- Pointer arithmetic
- Pointers to pointer
- String manipulation using pointer
- Pointers as function arguments
- Dynamic memory allocation
- Memory leak

Linked list

- The first part holds the information of the element or node
- The second piece contains the address of the next node (link / next-pointer field) in this structure list.



Linked list (Creation of a node)

```
struct Node
{
    int data;
    struct Node* next;
};
```

Linked list (Creation of HEAD)

```
struct Node* head;
```

- Take a **Node** pointer to use it as the **head** of the linked list.
- Declare the head **globally** so that it can be used by any function.

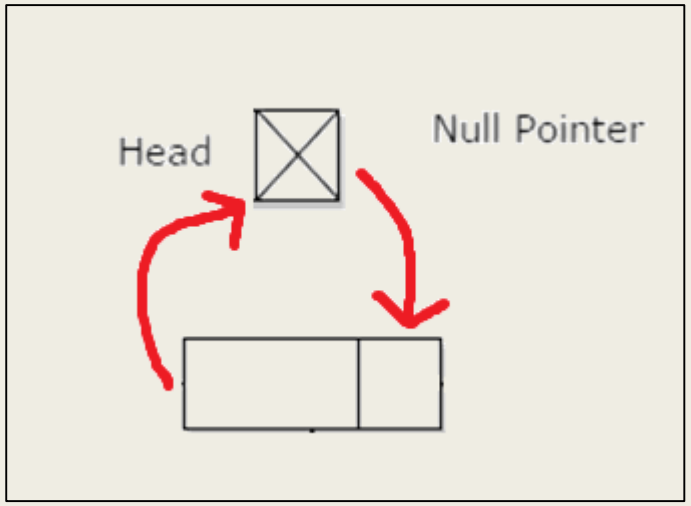
Linked list (Main function)

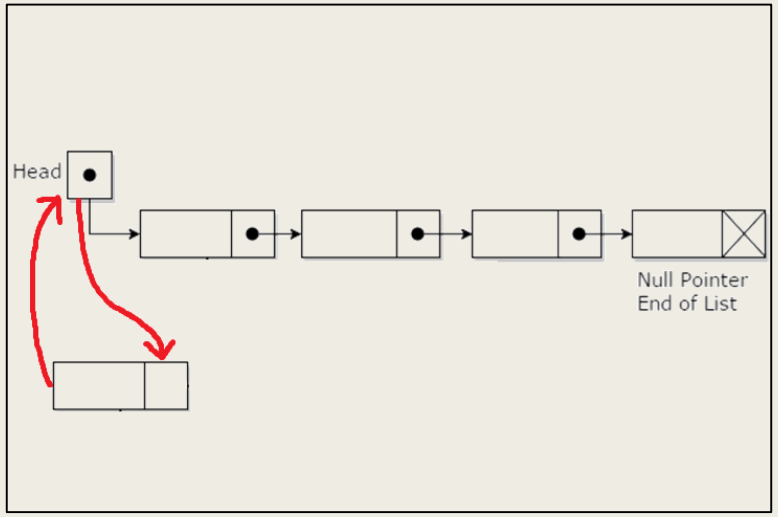
- Suppose there are two functions named Insert() and Print().
- One of them inserts elements in the list and other prints all the elements in the list

```
head = NULL;
printf("How many numbers?\n");
int n,i,x;
scanf("%d",&n);
for(i=0;i<n;i++)
{
    printf("Enter the number \n");
    scanf("%d",&x);
    Insert(x);
    Print();
}
```

Linked list (Insert)

```
void Insert(int x)
{
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = x;
    temp->next = head;
    head = temp;

}
```





Linked list (Print)

```
void Print()
{
    struct Node* temp = head;
    printf("List is:");
    while(temp != NULL)
    {
        printf(" %d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

Linked list (Delete)

```
void Delete(int n)
{
    struct Node* temp1 = head;
    if(n==1)
    {
        head = temp1->next;
        free(temp1);
        return;
    }
    int i;
    for(i=1;i<=n-2;i++)
        temp1=temp1->next;

    struct Node* temp2 = temp1->next;
    temp1->next = temp2->next;
    free(temp2);
}
```

Useful resources

- https://www.geeksforgeeks.org/data-structures/linked-list/#singlyLinkedList
- https://bit.ly/2EIJma0 (Pointers playlist of mycodeschool)
- https://bit.ly/1ElhMUI (mycodeschool data structure video)

Thank you