14 Data Structure in C: Linked List

Programming in C

Teacher: Po-Wen Chi
neokent@gapps.ntnu.edu.tw

May 31, 2021

Department of Computer Science and Information Engineering,
National Taiwan Normal University

Data Structure

Now You Know How to Program in C

- Do you know how to write a good program?

Now You Know How to Program in C

- Do you know how to write a good program?
- However, what is a good program?
  - Performance Issue: run faster and cost less.
  - Non-Performance Issue: stability, maintenance cost.

Now You Know How to Program in C

- Do you know how to write a good program?
- However, what is a good program?
  - Performance Issue: run faster and cost less.
  - Non-Performance Issue: stability, maintenance cost.
- In this class, we will focus on the first issue. How about the second issue?

Now You Know How to Program in C

- Do you know how to write a good program?
- However, what is a good program?
  - Performance Issue: run faster and cost less.
  - Non-Performance Issue: stability, maintenance cost.
- In this class, we will focus on the first issue. How about the second issue?
  - Good naming rule.
  - Good style.
  - Good architecture.
  - Good habit.

Now You Know How to Program in C

- Do you know how to write a good program?
- However, what is a good program?
  - Performance Issue: run faster and cost less.
  - Non-Performance Issue: stability, maintenance cost.
- In this class, we will focus on the first issue. How about the second issue?
  - Good naming rule.
  - Good style.
  - Good architecture.
  - Good habit.
  - But you will learn nothing unless you burn your own fingers.

Algorithm

- Actually, lots of problems have been solved, which means there are lots of existing source codes that you can **reference**.

Algorithm

- Actually, lots of problems have been solved, which means there are lots of existing source codes that you can **reference**.
- **Algorithm**: an unambiguous specification of how to solve a class of problems.
  - So, you must take **Algorithm** class to learn how to solve well-known problems.
  - And you need to learn how to classify problems.
  - Implement algorithms through your programming skill.

Data Structure

- When implementing an algorithm, we find that sometimes if we store data in some kind of **structured** way, the problem will become very easy.
- Example: Given an array with n elements, please find the maximum value of this array.

Data Structure

- When implementing an algorithm, we find that sometimes if we store data in some kind of **structured** way, the problem will become very easy.
- Example: Given an array with n elements, please find the maximum value of this array.
    - You must traverse all n elements to get the maximum value.

Data Structure

- When implementing an algorithm, we find that sometimes if we store data in some kind of **structured** way, the problem will become very easy.
- Example: Given an array with n elements, please find the maximum value of this array.
  - You must traverse all n elements to get the maximum value.
  - Do we have better ways?

Data Structure

- When implementing an algorithm, we find that sometimes if we store data in some kind of **structured** way, the problem will become very easy.
- Example: Given an array with n elements, please find the maximum value of this array.
  - You must traverse all n elements to get the maximum value.
  - Do we have better ways?
  - If the array is a *sorted array*, we only need **one step** to get the maximum value.
  - **Sorted array** is a structured way to store data.
  - We call this structured way **Data Structure**.

Data Structure

- When implementing an algorithm, we find that sometimes if we store data in some kind of **structured** way, the problem will become very easy.
- Example: Given an array with n elements, please find the maximum value of this array.
    - You must traverse all n elements to get the maximum value.
    - Do we have better ways?
    - If the array is a *sorted array*, we only need one step to get the maximum value.
    - **Sorted array** is a structured way to store data.
    - We call this structured way **Data Structure**.
    - **Question:** What is the problem of **sorting an array**? How much time do you need to insert a element? How much time do you need to get an element which is not max?

Algorithm and Data Structure

- We want to solve problems, so we need **Algorithm**.
- For implementing algorithms, we need **Data Structure**.
- These two classes are the most important classes in CS. Please pay attention to these two classes!
- Actually, most programming tests focus on this field, like ICPC.

Algorithm and Data Structure

- We want to solve problems, so we need **Algorithm**.
- For implementing algorithms, we need **Data Structure**.
- These two classes are the most important classes in CS. Please pay attention to these two classes!
- Actually, most programming tests focus on this field, like ICPC.

However, as your programming teacher, what I care is

Do you have ability to implement what you learn in those two classes?

So I will show you some data structures and see how to implement them.

Before We Start

- Again, there have been lots of people implement these data structures already.
- In practice, you should use these well-known implementations.

Before We Start

- Again, there have been lots of people implement these data structures already.
- In practice, you should use these well-known implementations.
- Unfortunately, this is a programming class.

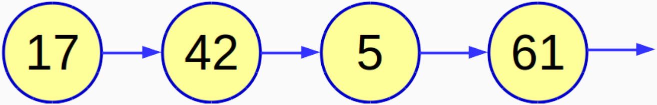
Before We Start

- Again, there have been lots of people implement these data structures already.
- In practice, you should use these well-known implementations.
- Unfortunately, this is a programming class.
- BTW, do not use languages other than C to study Data Structure or you will learn nothing.

Linked List

Linked List

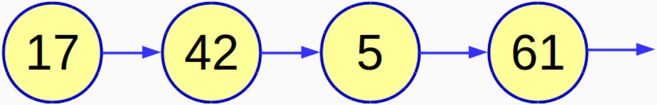
- A Linked list is a linear collection of data elements where each element points to the next.



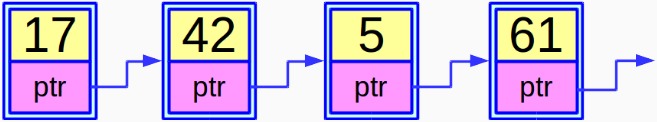
- It is a data structure consisting of a collection of nodes which together represent a **sequence**.

Linked List

- A Linked list is a linear collection of data elements where each element points to the next.



- It is a data structure consisting of a collection of nodes which together represent a **sequence**.
- Implementation in C.



ListNode

```
typedef struct _sListNode
{
    int32_t             data;
    struct _sListNode   *pNext;
} ListNode;
```

What if a List is Empty?

What if a List is Empty?

- It is simple. We can use a NULL pointer to present an empty list.

  ```
  ListNode *pList = NULL;
  ```

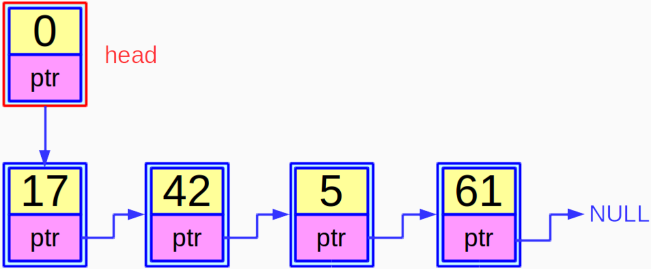
What if a List is Empty?

- It is simple. We can use a NULL pointer to present an empty list.

```
ListNode *pList = NULL;
```

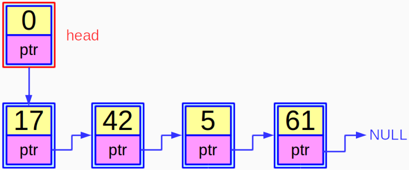
- Actually, I prefer the following way. You will see the reason soon.

```
ListNode head = { 0, NULL };
```



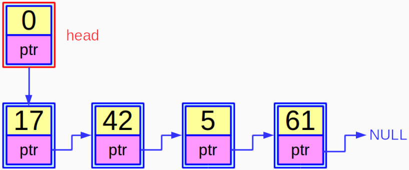
How to Add a new Node to a List?

- Check if the list exists.
    - Always check if the given input is valid.

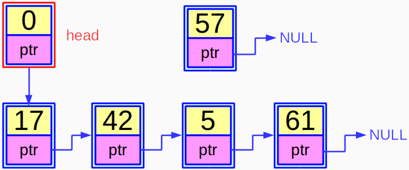


How to Add a new Node to a List?

- Check if the list exists.
  - Always check if the given input is valid.

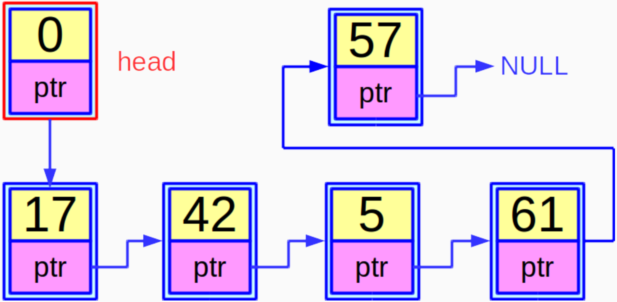


- Prepare a new node.
  - calloc a new node.



How to Add a new Node to a List?

- Append the new node to the end of the list.



Please see example/linked.list/linked_list_v01.c

Dummy Head

- The additional node is called Dummy Node.
- This is a useful technique to handle the **boundary condition**.
- Without dummy node, how can you know the difference between NULL pointer and empty list?
    - You may use double pointer to create a list.

Dummy Head

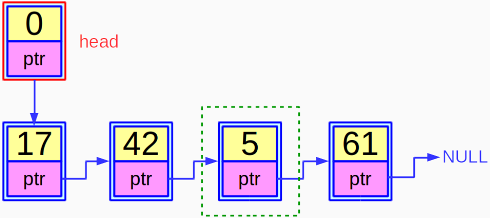
- The additional node is called Dummy Node.
- This is a useful technique to handle the **boundary condition**.
- Without dummy node, how can you know the difference between NULL pointer and empty list?
    - You may use double pointer to create a list.
- Undoubtedly, there are other ways to handle this case. It is up to you.

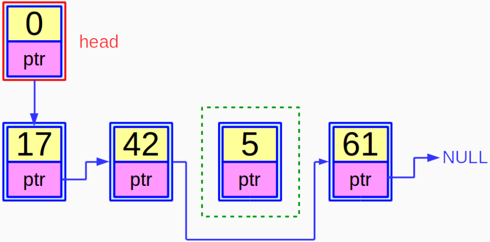
Practice

Please implement the following functions:

```
int32_t delNode( ListNode *pHead, int32_t val );
int32_t getListSize( ListNode *pHead );
```

How to Delete a Node?



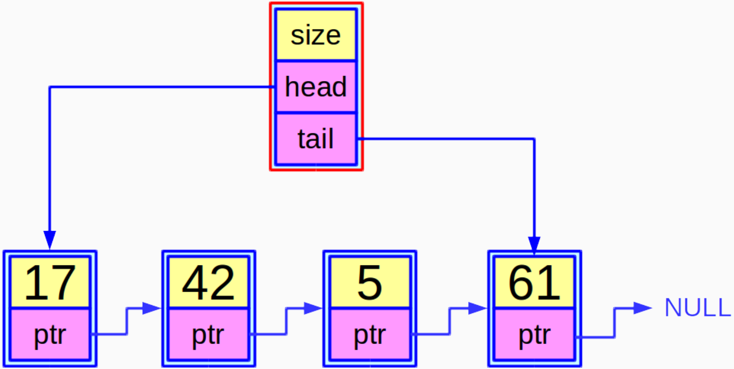


Caution! Do not forget to free memory!!

My Solution

Please see example/linked.list/linked_list_v02.c

My Solution



Header node can be used to store many global things if you need.

Practice

Please sorted the list in the descending order.

Please write functions to get max and min values.

I Have a Good Idea

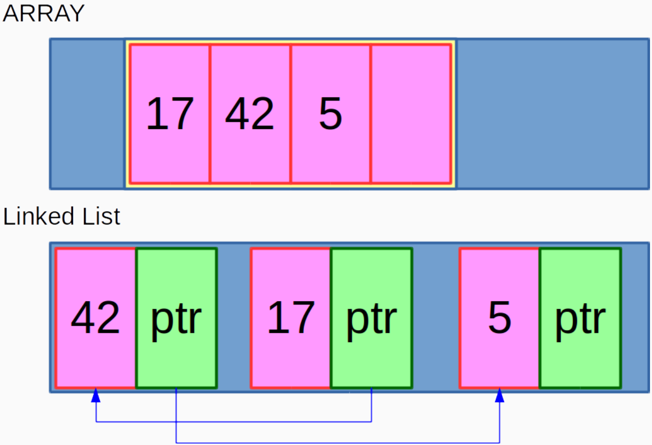
- I can save max and min in *sList*.
- Every time when a user add a new node, update these two values.
- So I can get min and max very soon.

I Have a Good Idea

- I can save max and min in *sList*.
- Every time when a user add a new node, update these two values.
- So I can get min and max very soon.
- Good Idea!! How about deleting node?

Wait a Minute! We Have Array!

- Array is also a sequence data structure.
- However, they have their own advantages.



Array

- Continuous memory layout.
- You can access data with constant time.
- The size must be predefined and cannot be modified.
- It implies you do not need to do memory management ... it is a good thing ... ?
- Delete a node may be a difficult thing.

Linked List

- Non-continuous memory layout.
- Data access with linear time.
- The size can be dynamic.
- Need to allocate and free ... memory leak risk??
- Delete a node or add is easy.

Practice

Please implement the following functions:

1. Delete all nodes.
2. Sum all nodes.
3. Show all nodes that are greater than some given value.

Something Like Template in C++

Open Question

I have shown you how to implement **linked list** in C. The problem is that this implementation supports only one structure. If you want to build a linked list of another structure, you need to implement again.

Of course, you can simply copy and paste with a small modification. It is still annoying.

Do you have any idea to implement **linked list** structure that supports different structures?

Before We Start

- CPP has a containers library.
- Container means that you can put any kind of objects into it.
  - In CPP, an object means a class.
  - Of course, all elements should have the same type.
- Three types of container:
  1. **Sequence containers**.
  2. **Associative containers**.
  3. **Unordered associative containers**.

Before We Start

- CPP has a containers library.
- Container means that you can put any kind of objects into it.
  - In CPP, an object means a class.
  - Of course, all elements should have the same type.
- Three types of container:
  1. **Sequence containers**.
  2. **Associative containers**.
  3. **Unordered associative containers**.

So CPP is greater than C since C cannot do this ... ?

Before We Start

- CPP has a containers library.
- Container means that you can put any kind of objects into it.
  - In CPP, an object means a class.
  - Of course, all elements should have the same type.
- Three types of container:
  1. **Sequence containers**.
  2. **Associative containers**.
  3. **Unordered associative containers**.

So CPP is greater than C since C cannot do this ... ? NO!!

void *

- A pointer to void is a **generic** pointer type.
- A void * can be converted to any other pointer type.
- **Important**: You cannot dereference a void * or do pointer arithmetic with it.
- Example: qsort.

```
void qsort(void *base, size_t nmemb, size_t size,
int (*compar)(const void *, const void *));
```

void *

- A pointer to void is a **generic** pointer type.
- A void * can be converted to any other pointer type.
- **Important**: You cannot dereference a void * or do pointer arithmetic with it.
- Example: qsort.

```
void qsort(void *base, size_t nmemb, size_t size,
int (*compar)(const void *, const void *));
```

So we can use void * to store data.

List Node Structure

```c
typedef struct _sListNode
{
    void                *pData;
    struct _sListNode   *pNext;
} sListNode;
```

That's all??

- Of course NO. If I do not know your structure, how can I do the following things?
  - How can I sort the list?
  - How can I delete a node from the list?
  - How can I print your list?

That’s all??

- Of course NO. If I do not know your structure, how can I do the following things?
    - How can I sort the list? You need to provide a comparison function.
    - How can I delete a node from the list? You need to provide a free function.
    - How can I print your list? You need to provide a print function.

That's all??

- Of course NO. If I do not know your structure, how can I do the following things?
    - How can I sort the list? You need to provide a comparison function.
    - How can I delete a node from the list? You need to provide a free function.
    - How can I print your list? You need to provide a print function.
- So you need to provide some callback register functions.

Structure

```
typedef struct _sList
{
    struct _sListParam  *pParam;
    struct _sListNode   *pHead;
    struct _sListNode   *pTail;
} sList;

typedef struct _sListParam
{
    int32_t size;
    int32_t ( *cmp )( const void *, const void * );
    void    ( *myfree )( void * );
    void    ( *myprint )( const void * );
} sListParam;
```

Callback Register Function

```
void regFreeCallback( sList *pList,
                      void  ( *myfree )( void * ) )
{
    if( pList == NULL )
    {
        printf( "%s(%d)␣%s:␣NULL␣pointer!\n",
                __FILE__, __LINE__, __FUNCTION__ );
        return;
    }

    pList -> pParam -> myfree = myfree;

    return;
}
```

Generic Linked List

- Now you can use this design as the generic linked list.
- Moreover, you can pack this as a library and use it when you need or share it with others.
  - You only need to provide the binary file, .o, and the header file.
  - Your implementation, your .c file, can keep secret.

Generic Linked List

- Now you can use this design as the generic linked list.
- Moreover, you can pack this as a library and use it when you need or share it with others.
  - You only need to provide the binary file, .o, and the header file.
  - Your implementation, your .c file, can keep secret.
- Wait! I want to keep the structure secret also.

Generic Linked List

- Now you can use this design as the generic linked list.
- Moreover, you can pack this as a library and use it when you need or share it with others.
  - You only need to provide the binary file, .o, and the header file.
  - Your implementation, your .c file, can keep secret.
- Wait! I want to keep the structure secret also.
- You can use Forward declaration.

Implementation

Please see linked.list.generic.

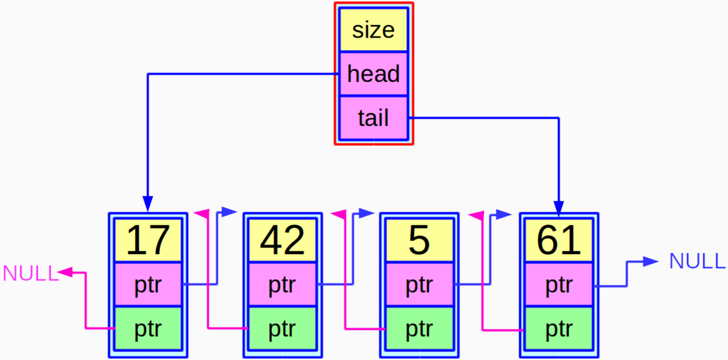
Practice

Please write the following functions:

1. Please implement a sort function that supports both ascending and descending order.
2. Please implement a function that only shows items with some given classes.

Practice: Double Linked List

Please modify the code and add the double linked list feature.



Sort a list and print from head to tail and tail to head.

Linked List Implementation in Linux Kernel

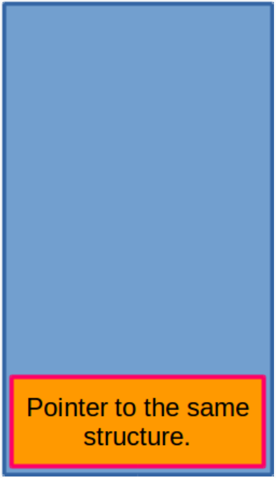
Linked List in the Real World

- If you want to improve your programming skill, write more and **read more**.
- Linked list is a popular data structure and it is impossible that no one implement it.
- At least you have one implementation from me.
  - Actually, this implementation is very textbook.
- How about other's implementation?
- Now let's see how **Linus Torvald** implements this.

In Most Data Structure Textbooks

- Pros:
  - Easy to understand.
  - Implementation is simple.
- Cons:
  - Not reusable. You need to do almost the same thing with every structure.

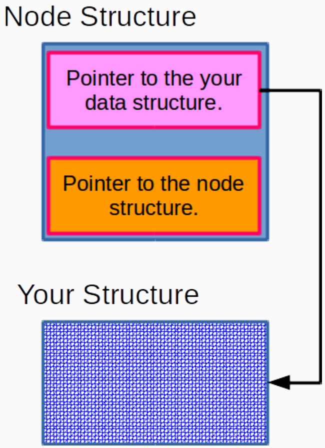
Your Structure



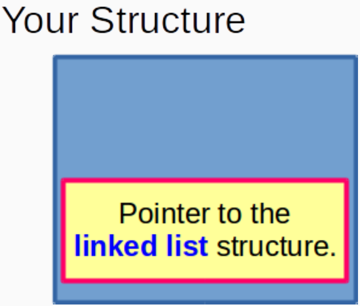
Pointer to the same structure.

Generic Linked List

- Pros:
    - Can be reusable.
- Cons:
    - Implementation is a little bit harder.
    - Need callback registration.


Node Structure
Pointer to the your data structure.
Pointer to the node structure.
Your Structure

Linus Torvald's Implementation



What you need to do is to insert this structure into your own structure.

Wait a minute! How can I access data?? There is no pointer to my structure!!

Let's see this implementation.

struct list_head

```c
struct list_head
{
    struct list_head *next, *prev;
};
```

This is very simple, right?

Initialization

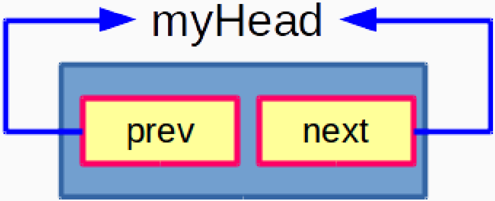
```
#define LIST_HEAD_INIT(name) { &(name), &(name) }

#define LIST_HEAD(name) \
struct list_head name = LIST_HEAD_INIT(name)
```

Initialization Example

```
LIST_HEAD( myHead );
```

The code can be described as follows:

```
struct list_head myHead = { &(myHead), &(myHead) };
```



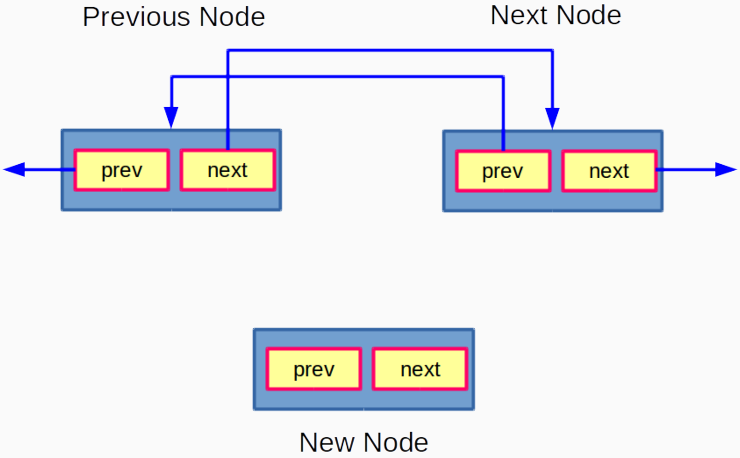
is_empty

```
static int list_empty(const struct list_head *head)
{
    return head -> next == head;
}
```

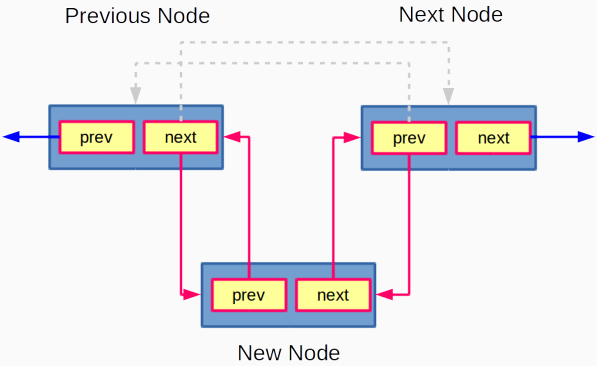
Add a New Node

```
static void __list_add(struct list_head *new,
                       struct list_head *prev,
                       struct list_head *next)
{
    next->prev = new;
    new->next = next;
    new->prev = prev;
    prev->next = new;
}
```

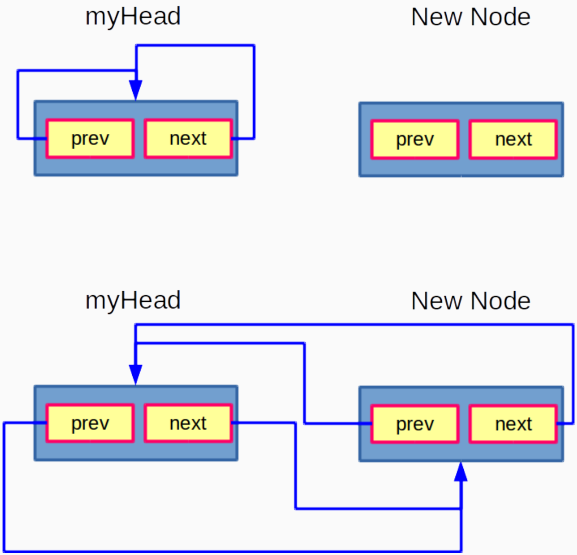
Add a New Node



Add a New Node



Add a New Node



Actually, this is a **ring** structure.

Quiz

```
static void list_add(struct list_head *new,
                     struct list_head *head)
{
    __list_add(new, head, head->next);
}
```

Where does this function add a new node?

Quiz

```
static void list_add(struct list_head *new,
                     struct list_head *head)
{
    __list_add(new, head, head->next);
}
```

Where does this function add a new node?

On the beginning of the list.

What if I Want to Add a New Node on the Tail?

```
static void list_add_tail(  struct list_head *new,
                            struct list_head *head)
{
    __list_add(new, head->prev, head);
}
```

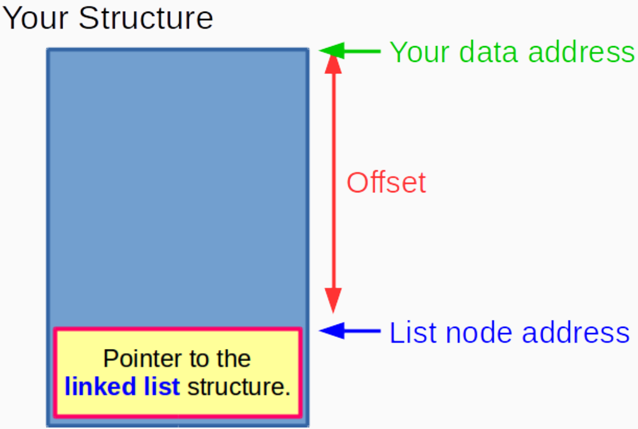
Delete a Node

```
static void __list_del( struct list_head * prev,
                        struct list_head * next)
{
    next->prev = prev;
    prev->next = next;
}

static void __list_del_entry(struct list_head *entry)
{
    if (entry == NULL)
        return;

    __list_del(entry->prev, entry->next);
}

static void list_del(struct list_head *entry)
{
    __list_del_entry(entry);
    entry->next = NULL;
    entry->prev = NULL;
}
```

Though This Linked List is Simple, How can I Access My Data?

```
#define list_entry(ptr, type, member) \
    container_of(ptr, type, member)

#define container_of(ptr, type, member) ({\
    void *__mptr = (void *)(ptr); \
    ((type *)(__mptr - offsetof(type, member))); })

#define offsetof(TYPE, MEMBER) \
    ((size_t)&((TYPE *)0)->MEMBER)
```

Container Of



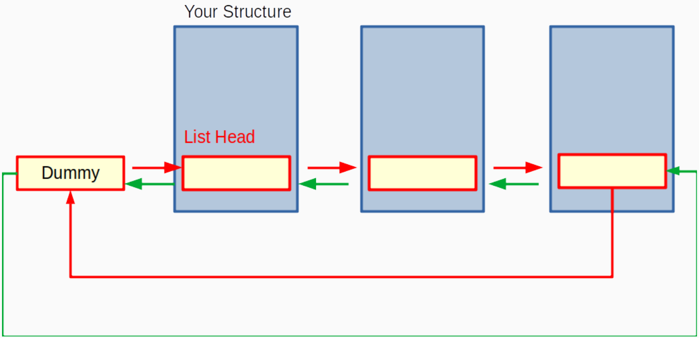
Get Head and Tail

```
#define list_first_entry(ptr, type, member) \
    list_entry((ptr)->next, type, member)

#define list_last_entry(ptr, type, member) \
    list_entry((ptr)->prev, type, member)
```

List Traversal

```
#define list_for_each(pos, head) \
    for (pos = (head)->next; pos != (head);
         pos = pos->next)

#define list_for_each_prev(pos, head) \
    for (pos = (head)->prev; pos != (head);
         pos = pos->prev)
```

Usage



Implementation

Please see linked.list.generic.

I just extract these macros from linux kernel code and make a list.h.

Practice

1. Please write a function to delete a character with a given ID.
2. Please write a function to find a character with the max total ability.
3. Please write a function to find the character with the best ability of each field.
4. Please write a function to delete all characters.

Note

- Though I introduce how Linus implement linked list, you may find some difference between my slide and linux kernel codes.
- The main reason is that OS kernel needs more protection, like avoiding race conditions.
- So what I introduce here is some kind of simplified version.

Reference

- https://github.com/torvalds/linux/blob/master/include/linux/list.h
- https://github.com/torvalds/linux/blob/master/include/linux/kernel.h
- https://github.com/torvalds/linux/blob/master/tools/include/linux/types.h
- https://github.com/torvalds/linux/blob/master/include/linux/stddef.h