An interesting C linked list idiom

Asked 16 years ago Modified 10 years, 1 month ago Viewed 3k times

11

I was at an interview for a C position in which they presented me with an idiom that I haven't previously encountered. This is a trick that simplifies implementation of various algorithms involving linked lists and I'm wondering if anybody else has encountered this.

Say we have a linked list record defined so:

```
typedef struct _record
{
    char* value;
    struct _record* next;
} record;
```

We need a function that inserts a new record so that the entire list remains sorted with respect to the value's in the records. The following implementation is simpler than anything I would have used, albeit less readable.

```
void insert_sorted(record** r, const char* value)
{
    record* newrec = NULL;
    while(*r && strcmp(value, (*r)->value) > 0)
        r = &((*r)->next); /* move r to point to the next field of the record
*/
    newrec = malloc(sizeof(record));
    newrec->value = strdup(value);
    newrec->next = *r;
    *r = newrec;
}
```

When the function is called, r points to the head pointer of the list. During the while loop, r is updated to point to the `next` field of the record that comes just before the point where we want to put the new record in. The last line of the function either updates the head pointer of the list (if the insertion happens at the beginning) or the `next` field of the previous record, which is quite cool.

A couple of questions:

- Does this idiom have a name or is it mentioned in any literature?
- Are there others like it in the C language?

I thought I knew C pretty well and had pointers and indirection pretty well figured out, but this one took me a while to fully understand.

c idioms

Share Improve this question Follow

asked Dec 1, 2008 at 22:19

elifiner
7,565 ● 8 ● 42 ● 48

`char* value` instead of `char *value`? ugh. Don't work there. – finnw Jul 14, 2010 at 9:36

1 @finnw That's a matter of personal (or workplace) style. For me it would also be `char* value`. – zentrunix Sep 1, 2013 at 14:53

1 @JoséX. Like most C programmers I have made the mistake of writing `char* pointer1, pointer2;` a few times. The `'char*'` (without a space) makes it more likely that a human will read it differently to how the compiler parses it (making this mistake more likely.) – finnw Sep 3, 2013 at 3:23

This technique is discussed, without giving it a name, in Writing Solid Code by Steve Maguire. There are those who excoriate the book (see the ACCU review); I think it is reasonable, though now rather dated in places (primarily because it was written before standard C compilers were uniformly available). – Jonathan Leffler Nov 9, 2014 at 23:30

11 Answers

Sorted by: Highest score (default)

7

I've used similar to this to insert into a binary tree. Because when iterating the tree, you usually stop when your pointer becomes `NULL` (you ran off the tree).

So to insert, you have 3 options,

1: use a variable which tracks the previous value of your iterating pointer.

2: stop when the pointer you would follow is NULL before you follow it, works but slightly less elegant in my opinion.

3: or a more elegant solution is simply use a pointer to a pointer, so you can just do: `*it = new_node();` and it'll add it where the `NULL` used to be in your tree.

For a linked list, while this code works nicely, I usually just use a doubly linked list which makes it trivial to insert at any location.

Share
Improve this answer
Follow

edited May 18, 2012 at 21:11

answered Dec 1, 2008 at 23:52

Evan Teran
90.3k ● 32 ● 187 ● 243

This is what I was looking for - both a recognition of the pattern and another use for it - in binary trees. Thanks! – elifiner Dec 2, 2008 at 8:20

6

I'd say the idiom is "the kind of code which gave 'c' a bad name"

- Unwarrantedly clever
- Unwarrantedly compact
- Surprising side effects on caller
- No error handling on malloc
- Only works for US English strings

Share Improve this answer Follow

answered Dec 1, 2008 at 22:38

Will Dean

39.5k ● 11 ● 92 ● 118

I mistakenly took a job in 2001 for a company that was using C. I knew it was a bad idea, And bad went to worse. I quit a few months later. I shudder thinking about having to go back to such a language. – Tim Dec 1, 2008 at 22:45

5 It looks straightforward, not "clever", and typical example code, which disregards error checking and uses an obvious library function. – Paul Nathan Dec 1, 2008 at 22:48

3 Wow, Dean really hates C. It's not too clever or compact, it is a typical C code. It does to caller exactly what caller wants. It is good practice to omit error handling while posting here if it obscures the point. – buti-oxa Dec 2, 2008 at 0:07

2 @Tim, I love that "I mistakenly took a job in 2001 for a company" - what, you were walking past their building and accidentally fell in through the door and signed the acceptance form? :-) – paxdiablo Dec 2, 2008 at 1:26

2 The passed-in pointer isn't marked const, so of course that signals to the caller that it might be changed? – unwind Dec 2, 2008 at 8:58

4

I don't see anything I'd call an idiom per se. It looks like standard coding for when you deal with datastructures in C.

My only complaint would be that the callers pointer (*r) is modified. Depending on the use of the function, I'd expect thats an unexpected side effect. Besides removing the unexpected side effect, using a local variable to play the role of *r would make the code more readable.

Share Improve this answer Follow

answered Dec 1, 2008 at 22:29

Frank Schwieterman
24.5k ● 15 ● 95 ● 132

2 Updating *r has the effect of returning a pointer to the new node. That is intentional, otherwise there is no clear way to get at the new node if the values are not unique. – ConcernedOfTunbridgeWells Dec 1, 2008 at 22:38

Why not just make the function return the record* to the new node? – Adam Jaskiewicz Dec 1, 2008 at 22:48

1 r is used to modify the head pointer of the list, in case it's empty. record *newlist = NULL; insert_sorted(&newlist, "value"); Now newlist points to a single-element linked list. – aib Dec 1, 2008 at 23:19

1 aib, thats a good scenario/idea. But when the new element does not become the list's head, the caller's pointer is then points to something besides the head. If the point is to tell the caller the head, it should do just that. – Frank Schwieterman Dec 1, 2008 at 23:55

You're right, I missed that the last *r= assignment is unconditional. It should only happen when the head pointer needs to be updated. – aib Dec 2, 2008 at 0:13

3

What would be the idiom here? Surely not the implementation of a linked list. The usage of a pointer to pointer construct? The compact loop?

Personally I'd use a pointer return value instead of operating on an input value. Because seeing this input data type would ring a bell, which makes me copy my value before handing it to your function.

Share Improve this answer Follow

answered Dec 1, 2008 at 22:29

Ronny Brendel
4,835 ● 5 ● 37 ● 55

3

This is a well known thing - double pointer iteration (that's my name for it, I don't know the official name). The goal is to be able to locate a position in single linked list, and then insert *before* that position (inserting after it is trivial). Implemented naively, this requires two pointers (current and prev) and special code for beginning of the list (when prev == NULL).

The code the way I usually write it is something along the lines of

```
void insertIntoSorted(Element *&head, Element *newOne)
{
  Element **pp = &head;
  Element *curr;
  while ((curr = *pp) != NULL && less(curr, newOne)) {
    pp = &(pp->next);
  }
  newOne->next = *pp;
```

```
  *pp = newOne;
}
```

Update:

I've forgot theother purpose for this trick - a more important one. It's used to delete elements from single linked lists:

```
// returns deleted element or NULL when key not found
Element *deleteFromList(Element *&head, const ElementKey &key)
{
  Element **pp = &head;
  Element *curr;
  while ((curr = *pp) != NULL && !keyMatches(curr, key)) {
    pp = &(pp->next);
  }
  if (curr == NULL) return NULL;
  *pp = (*pp)->next; // here is the actual delete
  return curr;
}
```

Share

Improve this answer

Follow

edited Sep 3, 2013 at 15:03

answered Dec 1, 2008 at 23:07

user3458

C doesn't have references and the type of pp should be Element **, not Element *.
– Evan Teran Dec 1, 2008 at 23:55

Fine, so I allowed myself some liberties here. This trick is as valuable in C++ as it is in C
– user3458 Dec 2, 2008 at 13:31

2

I don't know if this has a name or even if it's some special idiom but, since memory is relatively plentiful nowadays, my linked lists (where the language libraries don't make them available) are a special variant which simplifies the code greatly.

For a start, they're always doubly-linked since this makes traversal in both directions easier, for both processing and insert/delete operations.

An 'empty' list actually consists of two nodes, the head and the tail. By doing this, inserts and deletes do not need to worry about whether the node they're deleting is the head or tail, they can just assume it's a middle node.

Insertion of a new node y before node x then become a simple:

```
x -> prev -> next = y
y -> next = x
```

```
y -> prev = x -> prev
x -> prev = y
```

Deletion of node x is a simple:

```
x -> prev -> next = x -> next
x -> next -> prev = x -> prev
free x
```

Traversal is adjusted to ignore the extraneous head and tail:

```
n = head -> next
while n != tail
    process n
    n = n -> next
```

This all serves to make the code a lot easier to understand without all the special handling of the edge cases, at the cost of a couple of nodes of memory.

Share

Improve this answer

Follow

edited Dec 1, 2008 at 23:21

answered Dec 1, 2008 at 22:32

paxdiablo

880k ● 241 ● 1.6k ● 2k

Your traversal doesn't allow the pointer to n to be modified, to complete the insert.
– Frank Schwieterman Dec 1, 2008 at 22:33

Oops, thanks, @Frank, it's been a while since I did this since I'm mostly doing Java now and it has just about every data structure under the sun already implemented somewhere. Fixed it.
– paxdiablo Dec 1, 2008 at 23:22

AmigaOS use this kind of double-linked list extensively. – user52898 Feb 27, 2009 at 6:42

A useful pattern, but this does not answer the question – finnw Jul 14, 2010 at 9:51

@finnw, the question had two parts: (1) "Does this idiom have a name or is it mentioned in any literature?" (2) "Are there others like it in the C language?" - I think you'll find I was answering the second part. That's from vague memory, of course, it was a year and a half ago :-) – paxdiablo Jul 14, 2010 at 11:51

1

Instead of returning the value of the new node as an in/out parameter, you are better off having that be the return value of the function. This simplifies both the calling code, and the code inside the function (you can get rid of all those ugly double indirections).

```
record* insert_sorted(const record* head, const char* value)
```

You are missing error handling for the malloc/strdup failing case btw.

Share Improve this answer Follow

answered Dec 1, 2008 at 22:52

Greg Rogers
36.4k ● 17 ● 69 ● 94

I don't think `*head` should be const, because this function might need to modify its `next` pointer. – finnw Jul 14, 2010 at 9:49

1

This idiom is given in the Chapter 12 of "Pointers on C".This is used to insert a node into a linked list without list head.

Share Improve this answer Follow

answered Sep 1, 2013 at 14:22

placebo
11 ● 1

1

To answer the original question, this is known as a pointer centric approach instead of the naive node centric approach. Chapter 3 of "Advanced Programming Techniques" by Rex Barzee available at amazon.com includes a much better example implementation of the pointer centric approach.

Share

Improve this answer

Follow

edited Nov 14, 2014 at 6:33

Barzee
905 ● 3 ● 16 ● 31

answered Dec 10, 2008 at 2:29

Rex Barzee

Advanced Programming Techniques is currently a 2011 edition. The technique has been around a lot longer than that. It is discussed (somewhat in passing) in Writing Solid Code by S Maguire from 1993, for example. (WSC has a mixed reputation; it isn't a recommendation per se — though I think it is better than its critics allow; rather, it is simply a 'for instance' of prior publication. I'd be surprised to find that WSC was the first, too. I just happen to know of it.) – Jonathan Leffler Mar 22, 2017 at 22:01

0

I have also come up with this use of a double pointer, I have used it, but I don't really like it. The code that I came up with has this kernel to search for certain objects and remove them from the list:

```
Element** previous = &firstElement, *current;
while((current = *previous)) {
    if(shouldRemove(current)) {
        *previous = current->next;  //delete
    } else {
        previous = &current->next;  //point to next
```

```
    }
}
```

The reason I don't like my code is the subtle difference between the two if clauses: the syntax is almost identical, but the effect is entirely different. I do not think, we should write code as subtle as this, but doing it differently makes the code really lengthy. So, either way is bad - you might go for brevity or for readability, it's your choice.

Share

Improve this answer

Follow

edited Sep 1, 2013 at 15:21

answered Sep 1, 2013 at 15:03

cmaster - reinstate monica

40.5k ● 9 ● 66 ● 109

0

despite of the tricks, isn't the role of variable "r" changed? how does the caller tell what "*r" is for after calling? or after execution, what is the header of the list?

I could not believe this can be exemplified (even in some book?!). Did I miss anything?

If you do not return any pointer (like the others suggested), then I would suggest following changes to keep the role of the input.

```
void insert_sorted(record** head, const char* value)
{
    record** r = head;
    bool isSameHead = false;
    record* newrec = NULL;
    while(*r && strcmp(value, (*r)->value) > 0) {
        r = &((*r)->next); isSameHead = true; }
    newrec = malloc(sizeof(record));
    newrec->value = strdup(value);
    newrec->next = *r;
    *r = newrec;
    if (!isSameHead) *head = newrec;
}
```

actually, probably another better way to do it is using the "dummy head node", which links its next to the beginning of the list.

```
    void insert_sorted(record** head, const char* value)
    {
        record dummyHead;
        dummyHead.next = *head;
        record* r = &dummyHead;
        while(r->next) {
           if(strcmp(value, r->next->value) < 0)
              break;
           r = r->next;}
```

```
        newrec = malloc(sizeof(record));
        newrec->value = strdup(value);
        newrec->next = r->next;
        r->next = newrec;
        *head = dummyHead.next;
    }
```

Share

Improve this answer

Follow

edited Feb 2, 2014 at 19:28

answered Feb 2, 2014 at 17:20

pepero

7,483 ● 9 ● 47 ● 72