llist.c

```
#include "llist.h"
#include "alloc.h"

#include <stdlib.h> // size_t
#include <assert.h> // assert

typedef struct linkednode_t
{
  struct linkednode_t* prev;
  struct linkednode_t* next;
  void* data;
} LinkedNode;

typedef struct linkedlist_t
{
  size_t size;
  LinkedNode* head;
  LinkedNode* tail;
} LinkedList;

LinkedList* ll_new()
{
  LinkedList* list = (LinkedList*)mt_malloc(sizeof(LinkedList));
  if (list)
  {
    list->size = 0;
    list->head = NULL;
    list->tail = NULL;
  }
  return list;
}

size_t ll_size(const LinkedList* const ll)
{
  assert(ll);
  return ll->size;
```

```
}

bool ll_empty(const LinkedList* const ll)
{
  assert(ll);
  return !ll->size;
}

LinkedNode* ll_node_new()
{
  LinkedNode* node = (LinkedNode*)mt_malloc(sizeof(LinkedNode));
  if (!node)
    abort();
  node->prev = node->next = NULL;
  node->data = NULL;
  return node;
}
void ll_node_delete(LinkedNode* node)
{
  mt_free(node->data);
  mt_free(node);
}

LinkedNode* ll_front_node(const LinkedList* const ll)
{
  assert(ll);
  return ll->head;
}
LinkedNode* ll_back_node(const LinkedList* const ll)
{
  assert(ll);
  return ll->tail;
}

void* ll_front(const LinkedList* const ll)
{
  LinkedNode* head = ll_front_node(ll);
  return head ? head->data : NULL;
}
void* ll_back(const LinkedList* const ll)
{
  LinkedNode* tail = ll_back_node(ll);
  return tail ? tail->data : NULL;
}

void ll_push_front(LinkedList* const ll, void* p)
```

```
{
  assert(ll);
  //assert(p);

  LinkedNode* n = ll_node_new();
  n->data = p;

  if (ll->head)
  {
    ll->head->prev = n;
    n->next = ll->head;

    ll->head = n;
  }
  else
  {
    ll->head = n;
    ll->tail = n;
  }
  ++(ll->size);
}

void ll_push_back(LinkedList* const ll, void* p)
{
  assert(ll);
  //assert(p);

  LinkedNode* n = ll_node_new();
  n->data = p;

  if (ll->tail)
  {
    ll->tail->next = n;
    n->prev = ll->tail;

    ll->tail = n;
  }
  else
  {
    ll->tail = n;
    ll->head = n;
  }
  ++(ll->size);
}

void ll_pop_front(LinkedList* const ll)
```

```
{
  assert(ll);

  LinkedNode* front = ll_front_node(ll);
  if (!front)
    return;

  ll->head = front->next;
  if (ll->head)
    ll->head->prev = NULL;
  ll_node_delete(front);
  --(ll->size);
}

void ll_pop_back(LinkedList* const ll)
{
  assert(ll);

  LinkedNode* back = ll_back_node(ll);
  if (!back)
    return;

  back->prev->next = NULL;
  ll_node_delete(back);
  --(ll->size);
}

void ll_insert(LinkedIterator* const it, void* p)
{
  if (!it)
    return;
  if (!it->list)
    return;

  LinkedNode* node = ll_node_new();
  node->data = p;
  LinkedNode* target = it->current;
  if (target->prev)
  {
    target->prev->next = node;
    node->prev = target->prev;
  }
  target->prev = node;
  node->next = target;
  ++(it->list->size);
}
```

```
void* ll_erase(LinkedIterator* const it)
{
  if (!it)
    return NULL;
  if (!it->list)
    return NULL;

  // fix the pointers to remove the element
  if (it->current == it->list->head) // if it's the head, then head becomes next
    it->list->head = it->list->head->next;
  if (it->current == it->list->tail) // if it's the trail, the tail becomes prev
    it->list->tail = it->list->tail->prev;
  if (it->current->prev) // if we have a prev node, make that point to next
    it->current->prev->next = it->current->next;
  if (it->current->next) // if we have a next node, make that point to prev
    it->current->next->prev = it->current->prev;

  void* p = it->current->data;
  mt_free(it->current);
  it->current = NULL;
  --(it->list->size);
  return p;
}

LinkedNode* ll_at_node(const LinkedList* const ll, const size_t n)
{
  assert(ll);
  LinkedNode* node = ll->head;

  if (n >= ll->size)
    return NULL;

  for (size_t i = 0; i < n; ++i)
    node = node->next;
  return node;
}


LinkedIterator ll_at(const LinkedList* const ll, const size_t n)
{
  LinkedIterator it = ll_it_begin((LinkedList*)ll);
  for (size_t i = 0; it.current && i < n; ++i)
    ll_it_next(&it);
  return it;
}
```

```
void swap(void** a, void** b)
{
  void* p = *a;
  *a = *b;
  *b = p;
}
void ll_bsort(LinkedList* const ll,
            int32_t (*comparison)(const void* const a, const void* const b))
{
  assert(ll);
  if (ll->size <= 1) // nothing to do
    return;

  bool swapped;
  do
  {
    swapped = false;
    LinkedNode* node = ll_front_node(ll);
    while (node && node->next)
    {
      if ((*comparison)(node->data, node->next->data) > 0)
      {
        swap(&node->data, &node->next->data);
        swapped = true;
      }
      node = node->next;
    }
  }
  while (swapped);
}

void ll_clear(LinkedList* ll)
{
  while (ll->head)
  {
    LinkedNode* n = ll->head->next;
    mt_free(ll->head->data);
    mt_free(ll->head);
    ll->head = n;
  }
  ll->tail = NULL;
  ll->size = 0;
}
void ll_delete(LinkedList* ll)
{
```

```
  while (ll_front_node(ll))
    ll_pop_front(ll);
  mt_free(ll);
}
void ll_purge(LinkedList* ll)
{
  while (ll->head)
  {
    LinkedNode* n = ll->head->next;
    mt_free(ll->head);
    ll->head = n;
  }
  mt_free(ll);
}

LinkedIterator ll_it_begin(LinkedList* ll)
{
  LinkedIterator it = {ll, ll_front_node(ll)};
  return it;
}
LinkedIterator ll_it_rbegin(LinkedList* ll)
{
  LinkedIterator it = {ll, ll_back_node(ll)};
  return it;
}

void* ll_it_next(LinkedIterator* const it)
{
  it->current = it->current->next;
  return it->current ? it->current->data : NULL;
}
void* ll_it_rnext(LinkedIterator* const it)
{
  it->current = it->current->prev;
  return it->current ? it->current->data : NULL;
}

bool ll_it_valid(const LinkedIterator* const it)
{
  return it ? it->current : NULL;
}

void* ll_it_data(const LinkedIterator* const it)
{
  return it && it->current ? it->current->data : NULL;
}
```