Lab 3: B+ Tree 3

Instructor: Beom Heyn Kim

beomheynkim@hanyang.ac.kr

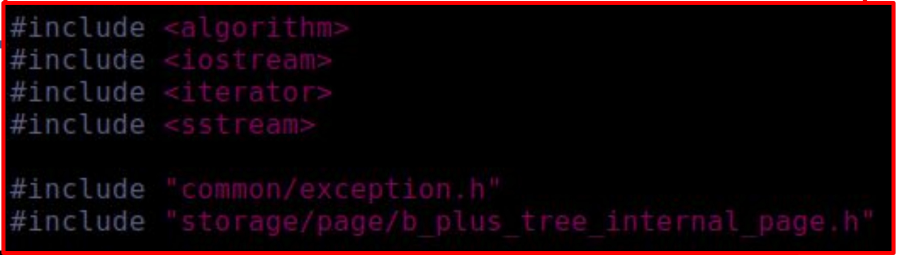
Department of Computer Science

Outline

- B+Tree Leaf Page
- Assignment

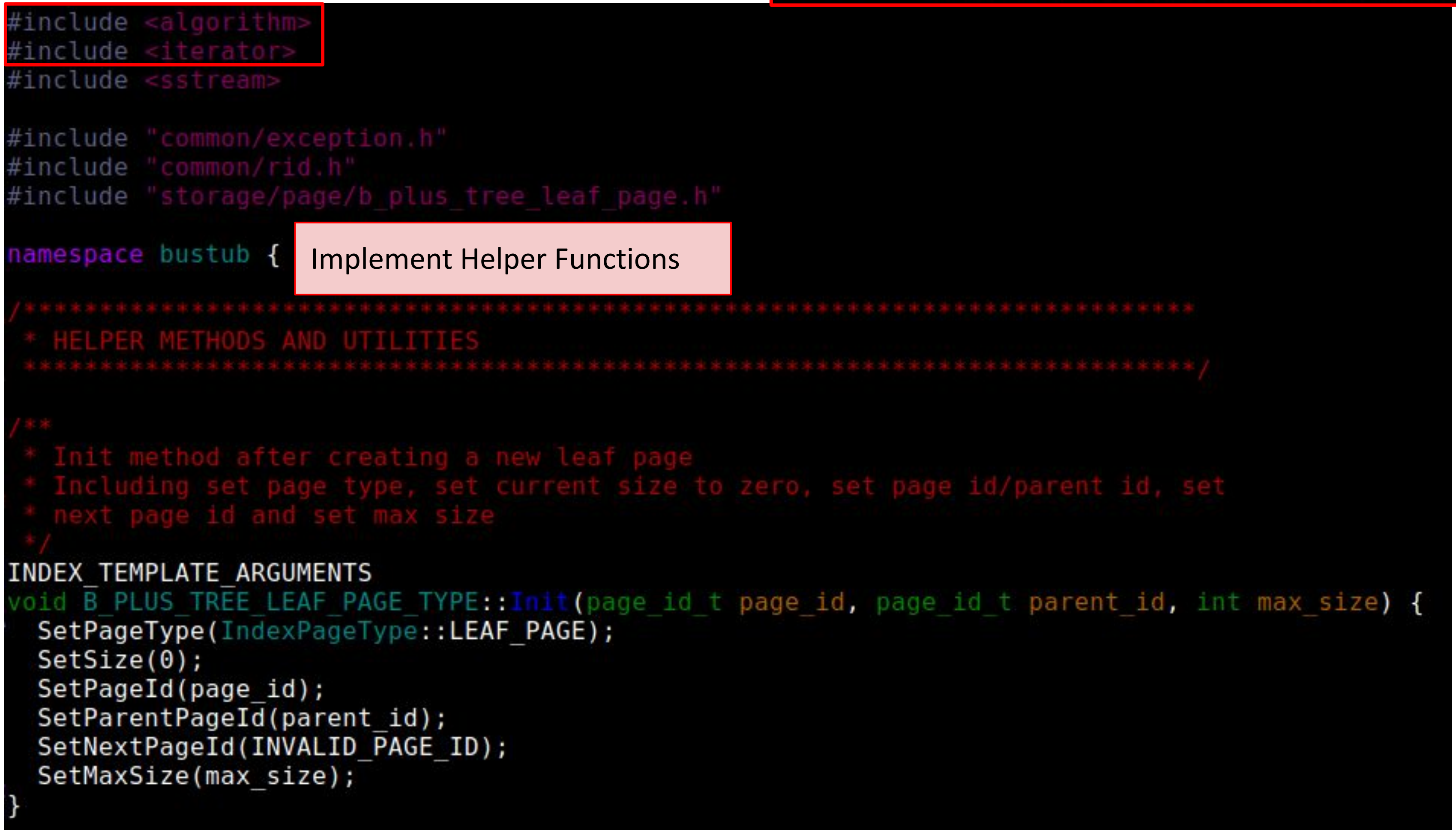
B+Tree Leaf Page

*During the last lab, we missed 2 files to include at the top of "b_plus_tree_internal.cpp"



```
#include <algorithm>
#include <iostream>
#include <iterator>
#include <sstream>

#include "common/exception.h"
#include "storage/page/b_plus_tree_internal_page.h"
```

b_plus_tree_leaf_page.cpp

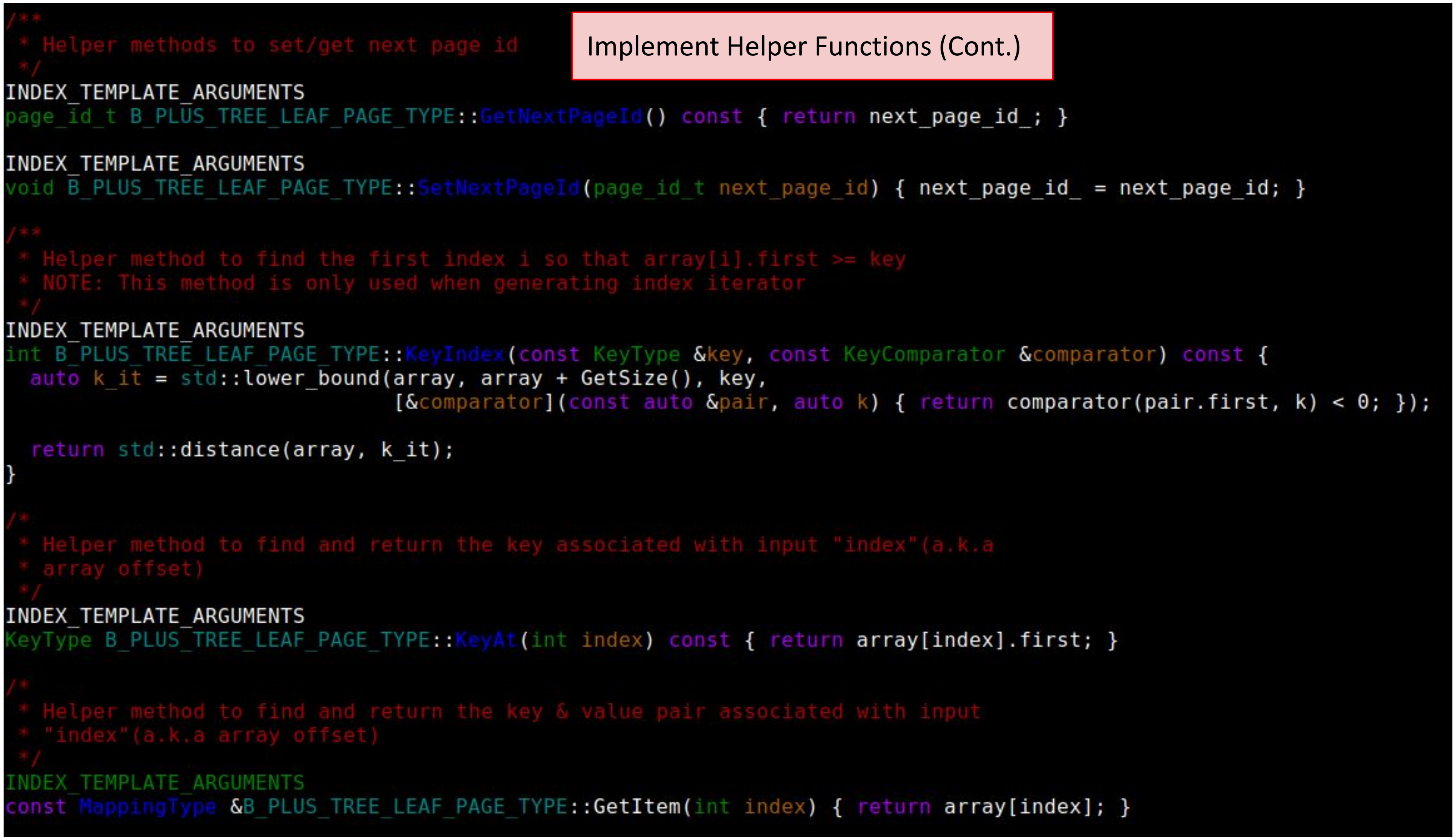


```
#include <algorithm>
#include <iterator>
#include <sstream>

#include "common/exception.h"
#include "common/rid.h"
#include "storage/page/b_plus_tree_leaf_page.h"

namespace bustub {

/*****************************************************************************
 * HELPER METHODS AND UTILITIES
 *****************************************************************************/

/**
 * Init method after creating a new leaf page
 * Including set page type, set current size to zero, set page id/parent id, set
 * next page id and set max size
 */
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::Init(page_id_t page_id, page_id_t parent_id, int max_size) {
  SetPageType(IndexPageType::LEAF_PAGE);
  SetSize(0);
  SetPageId(page_id);
  SetParentPageId(parent_id);
  SetNextPageId(INVALID_PAGE_ID);
  SetMaxSize(max_size);
}
```

Implement Helper Functions

B+Tree Leaf Page

b_plus_tree_leaf_page.cpp

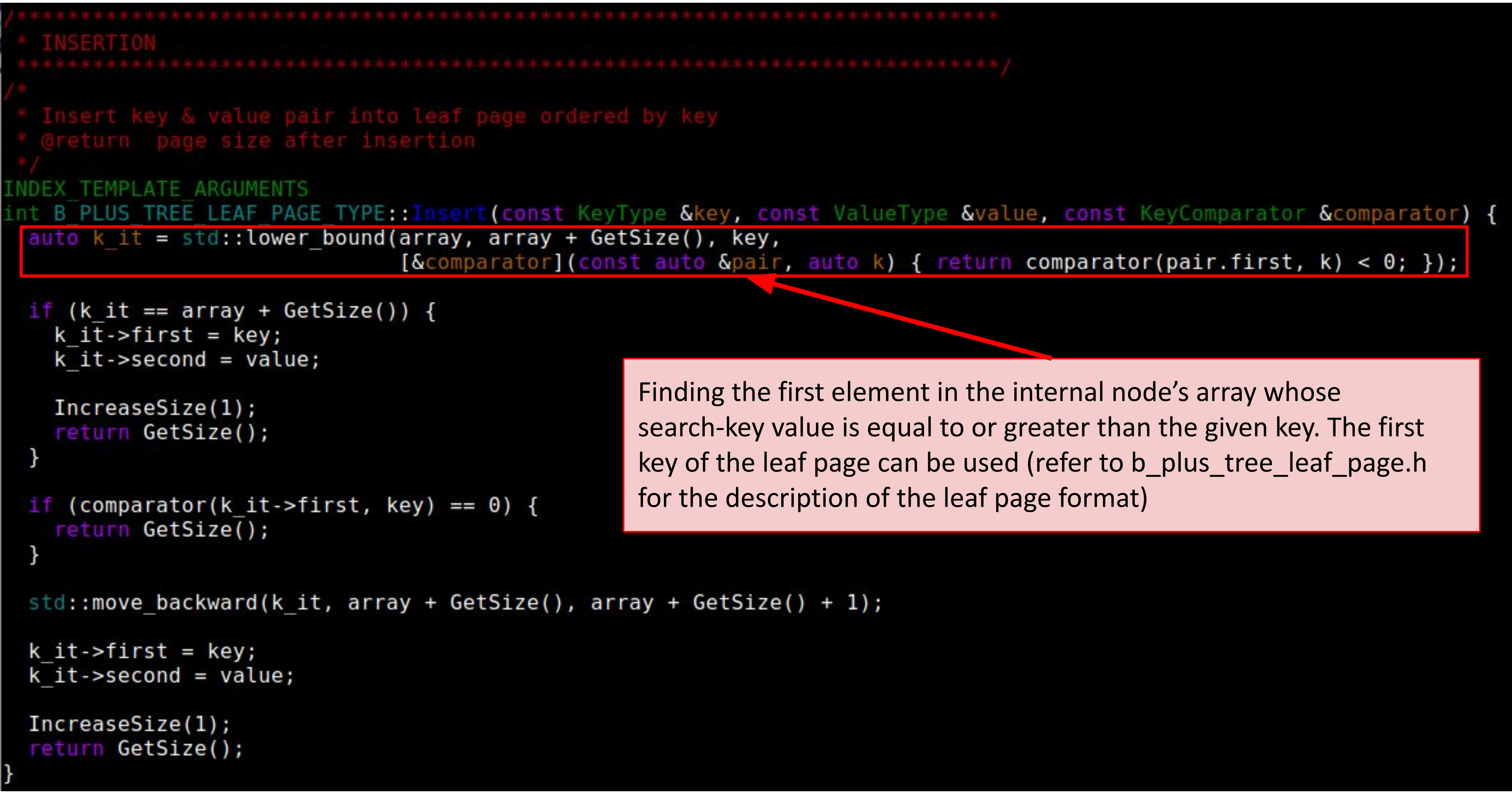


Implement Helper Functions (Cont.)

```
/**
 * Helper methods to set/get next page id
 */
INDEX_TEMPLATE_ARGUMENTS
page_id_t B_PLUS_TREE_LEAF_PAGE_TYPE::GetNextPageId() const { return next_page_id_; }

INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::SetNextPageId(page_id_t next_page_id) { next_page_id_ = next_page_id; }

/**
 * Helper method to find the first index i so that array[i].first >= key
 * NOTE: This method is only used when generating index iterator
 */
INDEX_TEMPLATE_ARGUMENTS
int B_PLUS_TREE_LEAF_PAGE_TYPE::KeyIndex(const KeyType &key, const KeyComparator &comparator) const {
  auto k_it = std::lower_bound(array, array + GetSize(), key,
                               [&comparator](const auto &pair, auto k) { return comparator(pair.first, k) < 0; });

  return std::distance(array, k_it);
}

/*
 * Helper method to find and return the key associated with input "index"(a.k.a
 * array offset)
 */
INDEX_TEMPLATE_ARGUMENTS
KeyType B_PLUS_TREE_LEAF_PAGE_TYPE::KeyAt(int index) const { return array[index].first; }

/*
 * Helper method to find and return the key & value pair associated with input
 * "index"(a.k.a array offset)
 */
INDEX_TEMPLATE_ARGUMENTS
const MappingType &B_PLUS_TREE_LEAF_PAGE_TYPE::GetItem(int index) { return array[index]; }
```

B+Tree Leaf Page

b_plus_tree_leaf_page.cpp

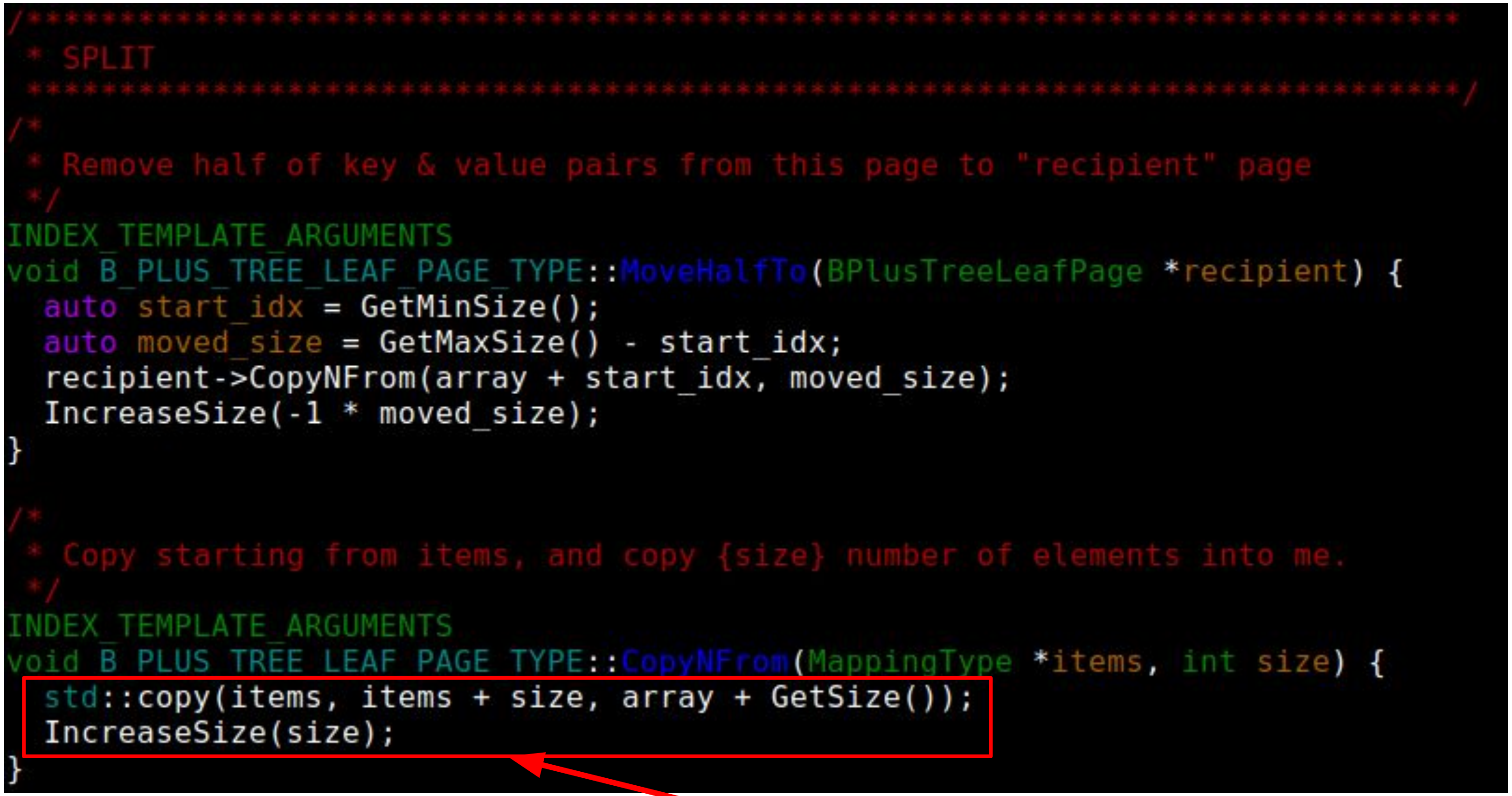


```cpp
/*****************************************************************************
 * INSERTION
 *****************************************************************************/
/*
 * Insert key & value pair into leaf page ordered by key
 * @return  page size after insertion
 */
INDEX_TEMPLATE_ARGUMENTS
int B_PLUS_TREE_LEAF_PAGE_TYPE::Insert(const KeyType &key, const ValueType &value, const KeyComparator &comparator) {
  auto k_it = std::lower_bound(array, array + GetSize(), key,
                               [&comparator](const auto &pair, auto k) { return comparator(pair.first, k) < 0; });

  if (k_it == array + GetSize()) {
    k_it->first = key;
    k_it->second = value;

    IncreaseSize(1);
    return GetSize();
  }

  if (comparator(k_it->first, key) == 0) {
    return GetSize();
  }

  std::move_backward(k_it, array + GetSize(), array + GetSize() + 1);

  k_it->first = key;
  k_it->second = value;

  IncreaseSize(1);
  return GetSize();
}
```

Finding the first element in the internal node’s array whose search-key value is equal to or greater than the given key. The first key of the leaf page can be used (refer to b_plus_tree_leaf_page.h for the description of the leaf page format)

B+Tree Leaf Page

b_plus_tree_leaf_page.cpp

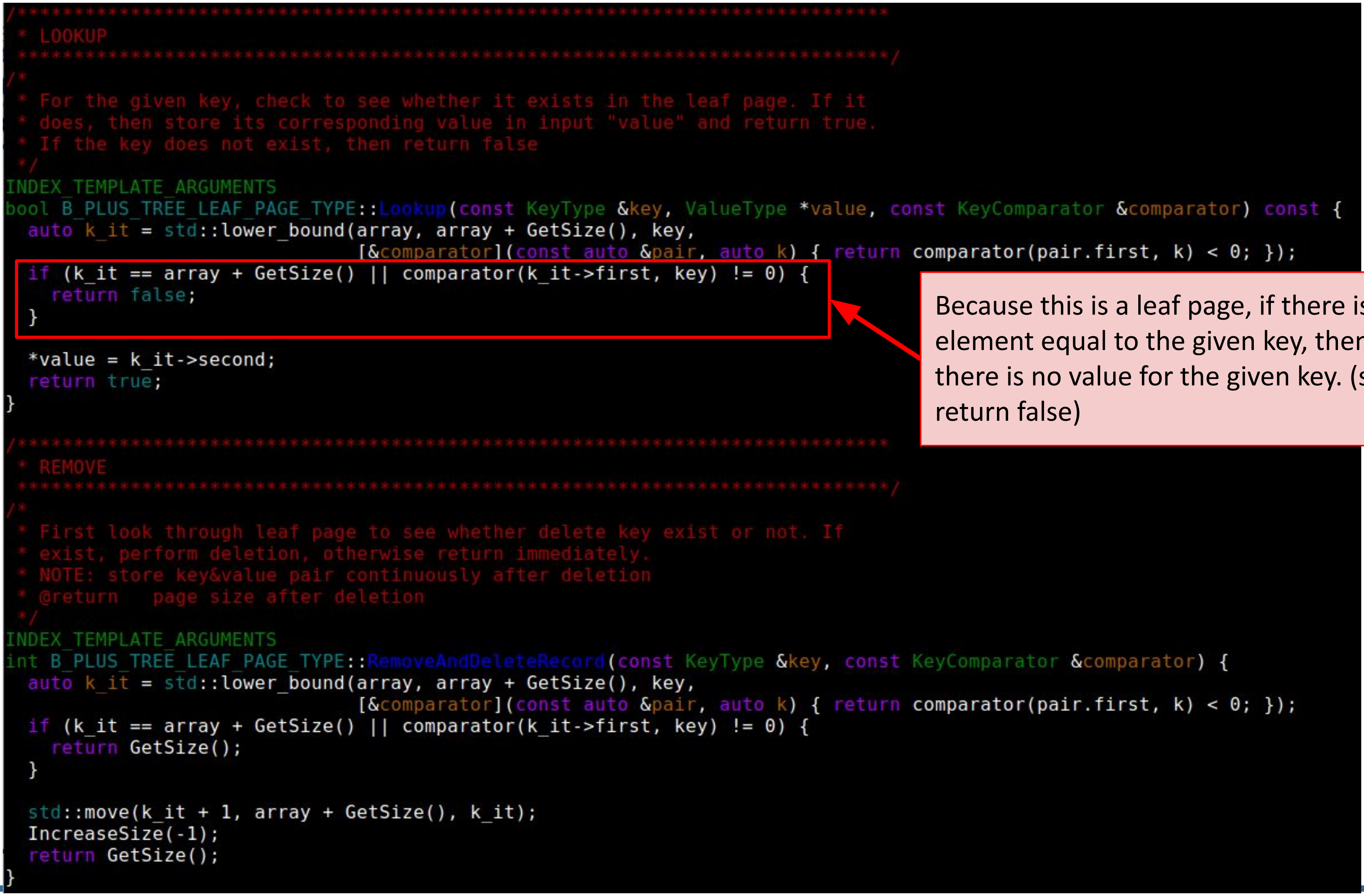


```
/*****************************************************************************
 * SPLIT
 *****************************************************************************/
/*
 * Remove half of key & value pairs from this page to "recipient" page
 */
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::MoveHalfTo(BPlusTreeLeafPage *recipient) {
  auto start_idx = GetMinSize();
  auto moved_size = GetMaxSize() - start_idx;
  recipient->CopyNFrom(array + start_idx, moved_size);
  IncreaseSize(-1 * moved_size);
}

/*
 * Copy starting from items, and copy {size} number of elements into me.
 */
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::CopyNFrom(MappingType *items, int size) {
  std::copy(items, items + size, array + GetSize());
  IncreaseSize(size);
}
```

Copy the half of elements (key and value pairs). Unlike the internal page, do not have children pages whose header should be updated. Refer to Split method in b_plus_tree.cpp and also compare it with the internal page's case.

B+Tree Leaf Page

b_plus_tree_leaf_page.cpp

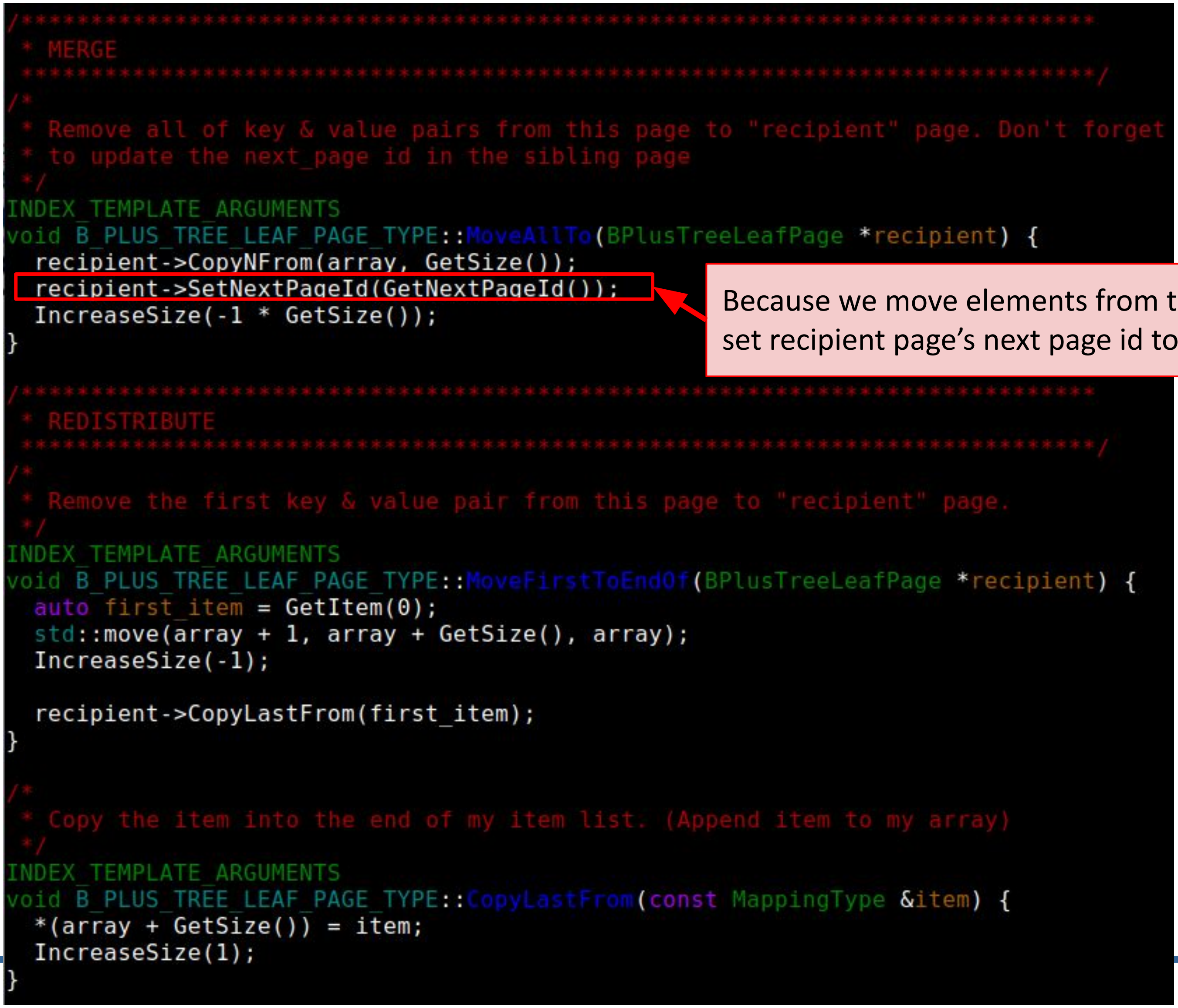


```
/*****************************************************************************
 * LOOKUP
 *****************************************************************************/
/*
 * For the given key, check to see whether it exists in the leaf page. If it
 * does, then store its corresponding value in input "value" and return true.
 * If the key does not exist, then return false
 */
INDEX_TEMPLATE_ARGUMENTS
bool B_PLUS_TREE_LEAF_PAGE_TYPE::Lookup(const KeyType &key, ValueType *value, const KeyComparator &comparator) const {
  auto k_it = std::lower_bound(array, array + GetSize(), key,
                               [&comparator](const auto &pair, auto k) { return comparator(pair.first, k) < 0; });
  if (k_it == array + GetSize() || comparator(k_it->first, key) != 0) {
    return false;
  }

  *value = k_it->second;
  return true;
}

/*****************************************************************************
 * REMOVE
 *****************************************************************************/
/*
 * First look through leaf page to see whether delete key exist or not. If
 * exist, perform deletion, otherwise return immediately.
 * NOTE: store key&value pair continuously after deletion
 * @return   page size after deletion
 */
INDEX_TEMPLATE_ARGUMENTS
int B_PLUS_TREE_LEAF_PAGE_TYPE::RemoveAndDeleteRecord(const KeyType &key, const KeyComparator &comparator) {
  auto k_it = std::lower_bound(array, array + GetSize(), key,
                               [&comparator](const auto &pair, auto k) { return comparator(pair.first, k) < 0; });
  if (k_it == array + GetSize() || comparator(k_it->first, key) != 0) {
    return GetSize();
  }

  std::move(k_it + 1, array + GetSize(), k_it);
  IncreaseSize(-1);
  return GetSize();
}
```

Because this is a leaf page, if there is no element equal to the given key, then there is no value for the given key. (so, return false)

B+Tree Leaf Page

b_plus_tree_leaf_page.cpp

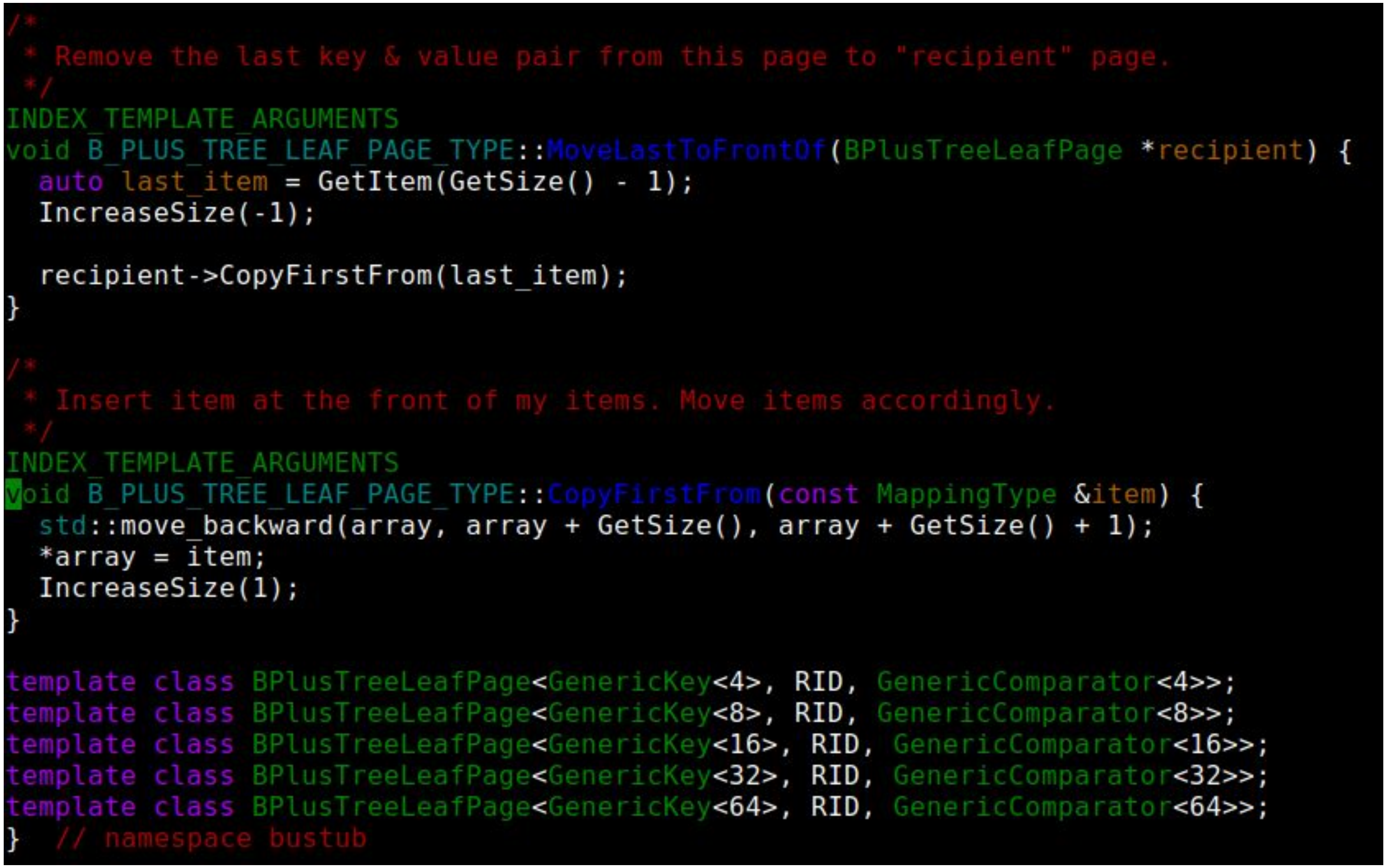


```
/*****************************************************************************
 * MERGE
 *****************************************************************************/
/*
 * Remove all of key & value pairs from this page to "recipient" page. Don't forget
 * to update the next_page id in the sibling page
 */
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::MoveAllTo(BPlusTreeLeafPage *recipient) {
  recipient->CopyNFrom(array, GetSize());
  recipient->SetNextPageId(GetNextPageId());
  IncreaseSize(-1 * GetSize());
}

/*****************************************************************************
 * REDISTRIBUTE
 *****************************************************************************/
/*
 * Remove the first key & value pair from this page to "recipient" page.
 */
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::MoveFirstToEndOf(BPlusTreeLeafPage *recipient) {
  auto first_item = GetItem(0);
  std::move(array + 1, array + GetSize(), array);
  IncreaseSize(-1);

  recipient->CopyLastFrom(first_item);
}

/*
 * Copy the item into the end of my item list. (Append item to my array)
 */
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::CopyLastFrom(const MappingType &item) {
  *(array + GetSize()) = item;
  IncreaseSize(1);
}
```

Because we move elements from this page to “recipient” page, we set recipient page’s next page id to be this page’s next page id.

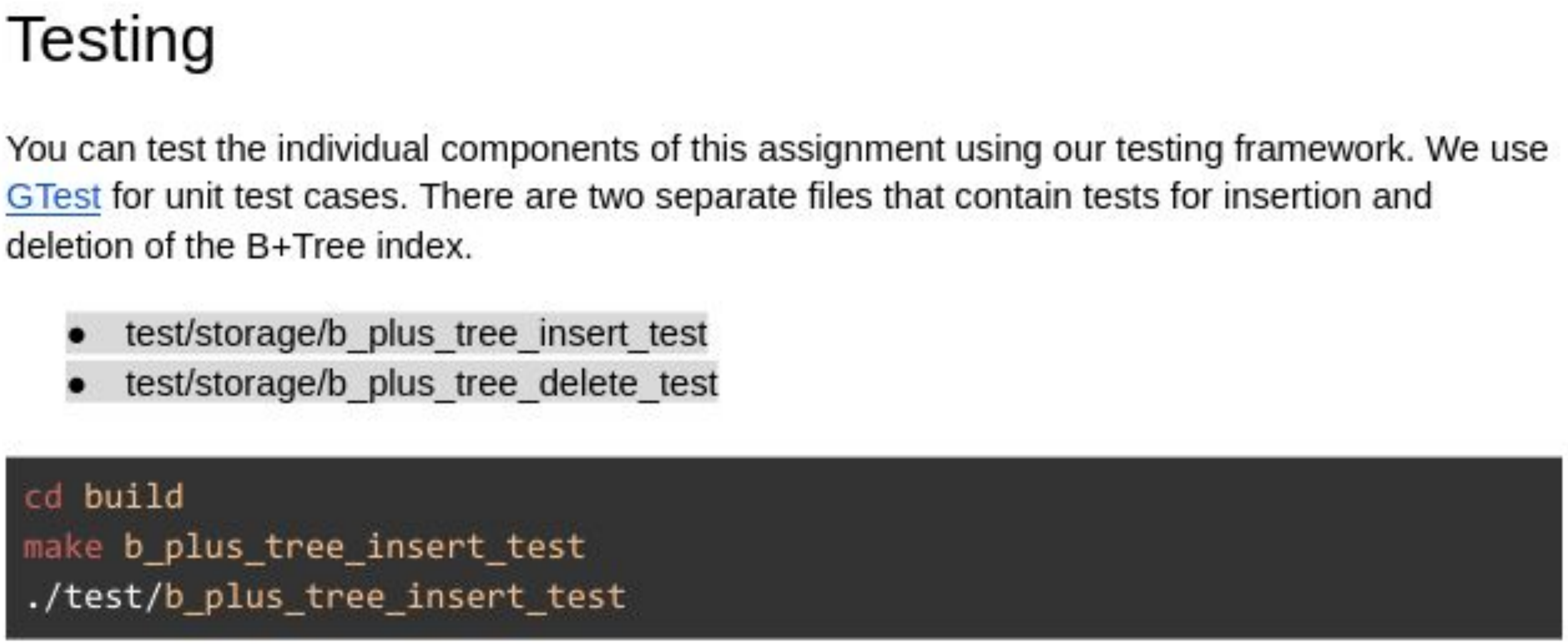
B+Tree Leaf Page

b_plus_tree_leaf_page.cpp



```
/*
 * Remove the last key & value pair from this page to "recipient" page.
 */
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::MoveLastToFrontOf(BPlusTreeLeafPage *recipient) {
  auto last_item = GetItem(GetSize() - 1);
  IncreaseSize(-1);

  recipient->CopyFirstFrom(last_item);
}

/*
 * Insert item at the front of my items. Move items accordingly.
 */
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::CopyFirstFrom(const MappingType &item) {
  std::move_backward(array, array + GetSize(), array + GetSize() + 1);
  *array = item;
  IncreaseSize(1);
}

template class BPlusTreeLeafPage<GenericKey<4>, RID, GenericComparator<4>>;
template class BPlusTreeLeafPage<GenericKey<8>, RID, GenericComparator<8>>;
template class BPlusTreeLeafPage<GenericKey<16>, RID, GenericComparator<16>>;
template class BPlusTreeLeafPage<GenericKey<32>, RID, GenericComparator<32>>;
template class BPlusTreeLeafPage<GenericKey<64>, RID, GenericComparator<64>>;
}  // namespace bustub
```

Testing



Testing

You can test the individual components of this assignment using our testing framework. We use GTest for unit test cases. There are two separate files that contain tests for insertion and deletion of the B+Tree index.

- test/storage/b_plus_tree_insert_test
- test/storage/b_plus_tree_delete_test

```
cd build
make b_plus_tree_insert_test
./test/b_plus_tree_insert_test
```

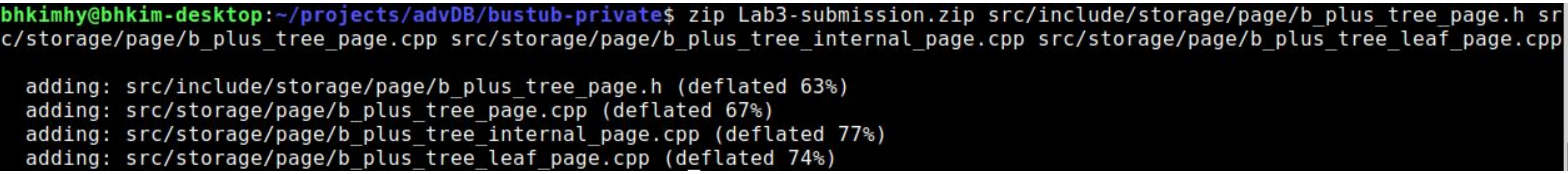
Don’t forget to use files provided in “lab3-prep.zip” during the first B+Tree Lab.
Also, if you cannot make the test pass even after implementing everything, try to clean by doing ‘make clean’ first and then rebuild and retry the tests as above

Outline

- B+Tree Leaf Page
- Assignment

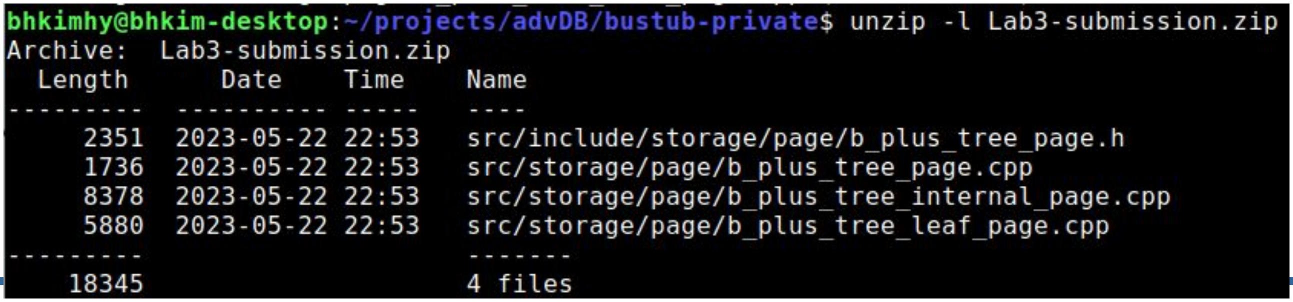
Assignment: Finish up and Submit!

- Finish your implementation, test it, zip it and submit it on LMS as Lab3-submission.zip
    - You only need to include the following files with their full path in your submission zip file:
        - src/include/storage/page/b_plus_tree_page.h
        - src/storage/page/b_plus_tree_page.cpp
        - src/storage/page/b_plus_tree_internal_page.cpp
        - src/storage/page/b_plus_tree_leaf_page.cpp
    - zip it as follows:



```
bhkimhy@bhkim-desktop:~/projects/advDB/bustub-private$ zip Lab3-submission.zip src/include/storage/page/b_plus_tree_page.h sr
c/storage/page/b_plus_tree_page.cpp src/storage/page/b_plus_tree_internal_page.cpp src/storage/page/b_plus_tree_leaf_page.cpp

  adding: src/include/storage/page/b_plus_tree_page.h (deflated 63%)
  adding: src/storage/page/b_plus_tree_page.cpp (deflated 67%)
  adding: src/storage/page/b_plus_tree_internal_page.cpp (deflated 77%)
  adding: src/storage/page/b_plus_tree_leaf_page.cpp (deflated 74%)
```

- Verify it using unzip as follows:



```
bhkimhy@bhkim-desktop:~/projects/advDB/bustub-private$ unzip -l Lab3-submission.zip
Archive:  Lab3-submission.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
     2351  2023-05-22 22:53   src/include/storage/page/b_plus_tree_page.h
     1736  2023-05-22 22:53   src/storage/page/b_plus_tree_page.cpp
     8378  2023-05-22 22:53   src/storage/page/b_plus_tree_internal_page.cpp
     5880  2023-05-22 22:53   src/storage/page/b_plus_tree_leaf_page.cpp
---------                     -------
    18345                     4 files
```

The End