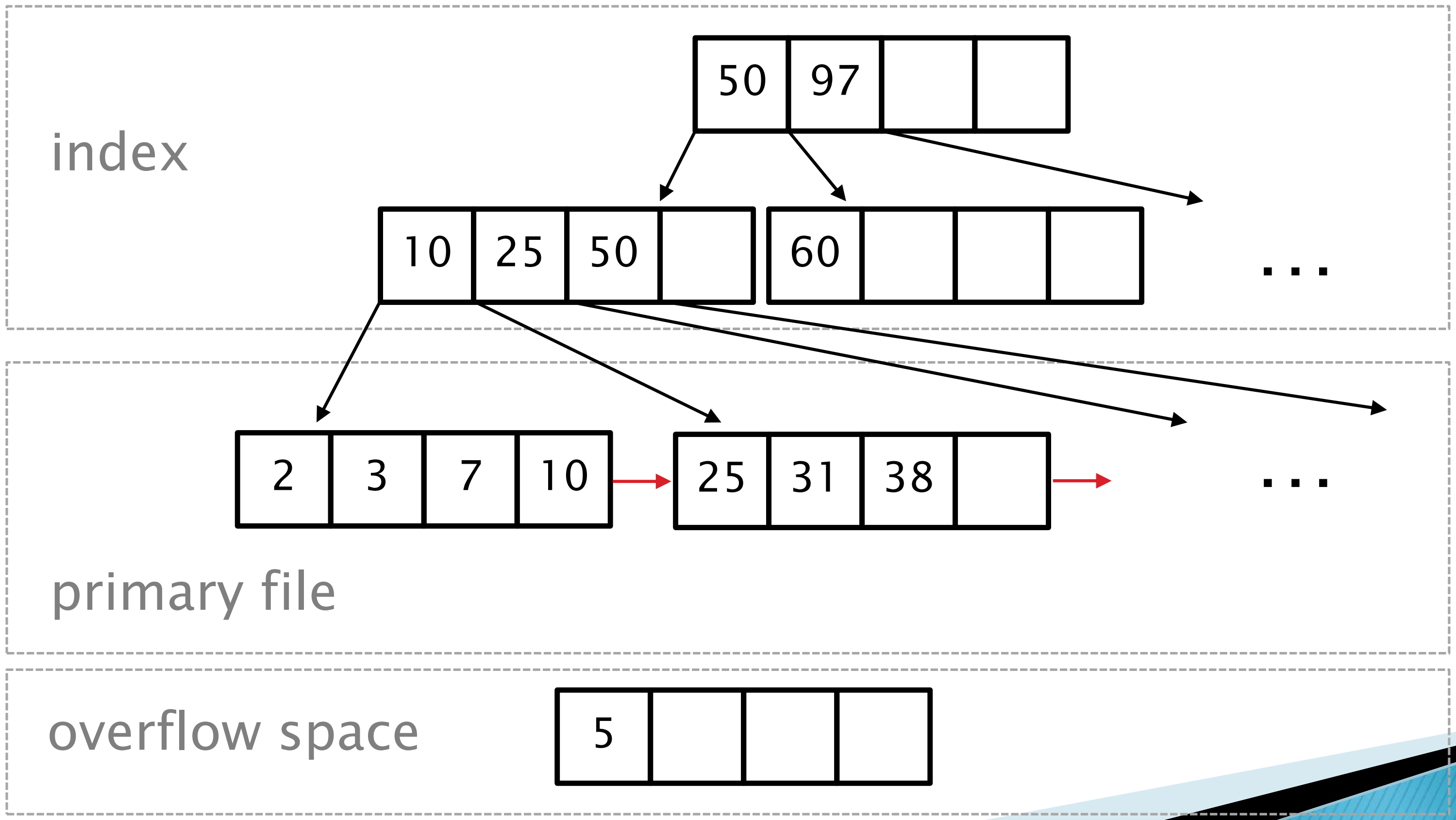
Advanced C++ Programming

Homework Assignment 3
2017/2018
Přemysl Čech

ISAM (Indexed Sequential Access Method)

- Goal is to implement:
  - a class `isam` which uses memory blocks
  - a forward `isam_iter` iterator over `isam`



ISAM

- Utilizes memory blocks
  - blocks are obtained through `block_provider` namespace
  - all methods should be self explanatory – file block_provider.hpp
  - simulates I/O operations, block IDs start from 1 (never 0)

- Three main parts
  - Index – block IDs can fit in the main memory
    - you can implement any variant you want
    - search time – in the worst case $\mathcal{O}(\log N)$

  - Primary file – blocks stored through the `block_provider`
    - blocks can be read sequentially – there should be a pointer to the following block
    - all records in blocks are stored in a sorted order by the key property

  - Overflow space – in memory
    - has given size
    - when it is full, reorganization is needed

ISAM

- `template <class TKey, class TValue> class isam`
  - `TKey` – simple value type, no duplicates, comparable: `operator<`
  - `TValue` – default constructible, assume reasonable usage

  - constructor – two parameters
    - block size B
    - size of the overflow space

    $\left.\begin{array}{l}\text{block size } B\\ \text{size of the overflow space}\end{array}\right\}$ number of records (`TKey`,`TValue`) ≥ 1

  - associative container – indexer: `TValue& operator[](TKey key)`
    - for both read and write operations into a container

  - no deletion

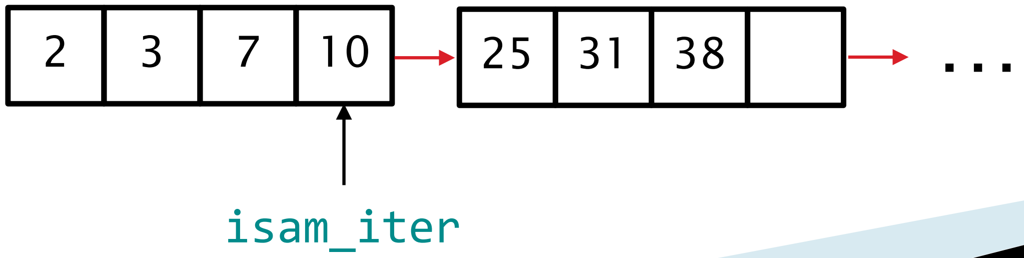
  - `isam_iter begin()` – points to the first record of the first data block
  - `isam_iter end()` – points after the last record in the last block

ISAM

- `TValue& operator[](TKey key)`
  - find appropriate *block* in the primary file using the index
  - in case the `key` exists in the container, returns the value
  - in case the `key` does not exist in the container
    - if the *block* is not full insert new record to the *block* with default value
    - else insert new record to the overflow space
      - if the overflow space is full, reorganize
- reorganization
  - insert all records from the overflow space to the primary file (allocate new blocks if necessary)
  - rebuild (correct) the index
  - can be naive – no specific restrictions
    - implement as efficiently as possible

ISAM

- `isam_iter`
  - the forward iterator category is good enough (ref)
  - main purpose – read all records sorted by the keys from the primary file in a sequential order
  - should expose both *key* and *value* properties (like `std::pair<TKey,TValue>`)
  - beware of constant/non constant variant
  - operators accessing one record should be fast
    - an iterator can hold one block in the main memory
  - operators moving the pointer could be „slow“ (possible block I/Os)



ISAM

- Specifications
  - only <u>one</u> primary block can be loaded in the main memory at the time
    - for both `isam` and `isam_iter`
    - during reorganization max. 2 blocks can be temporarily loaded
  - size of one block should not exceed: $B \cdot sizeof(record) + C$
    - C is small
  - `isam` copy constructor and operator can be deleted
    - `isam_iter` can be copied
  - be careful of proper indexer and iterator writes

ISAM

- Examples:

```
isam<int, string*> index(1, 2);
index[5] = new string("5");
index[2] = new string("2");
index[4] = new string("4");  //any records in the overflow space?
for (auto&& it : index)
{
    cout << it.first << ":" << it.second << " ";
}
//output: 2:2 4:4 5:5
```

```
isam<int, double> index(1, 1);
index[1] = 1;
{ auto it = index.begin(); it->second = 2; }
cout << index[1] << endl;
//output: 2
```

Code and submissions

- You should follow best C++ practices
  - proper naming, no duplicate code
  - effective parameters passing
  - correct allocation and deletion of all variables
  - reasonable distribution of your code into functions and classes

  - DO NOT CHANGE THE API – public function names must remain the same!

- Submit your final solution to the ReCodEx system
  - submit only one file: isam.hpp
  - write your name in a comment at the beginning of the file

  - 4 tests
    - 1 – basic container and iterator tests
    - 2 – iterator write tests
    - 3 – max. block loads tests
    - 4 – larger scale (speed) tests