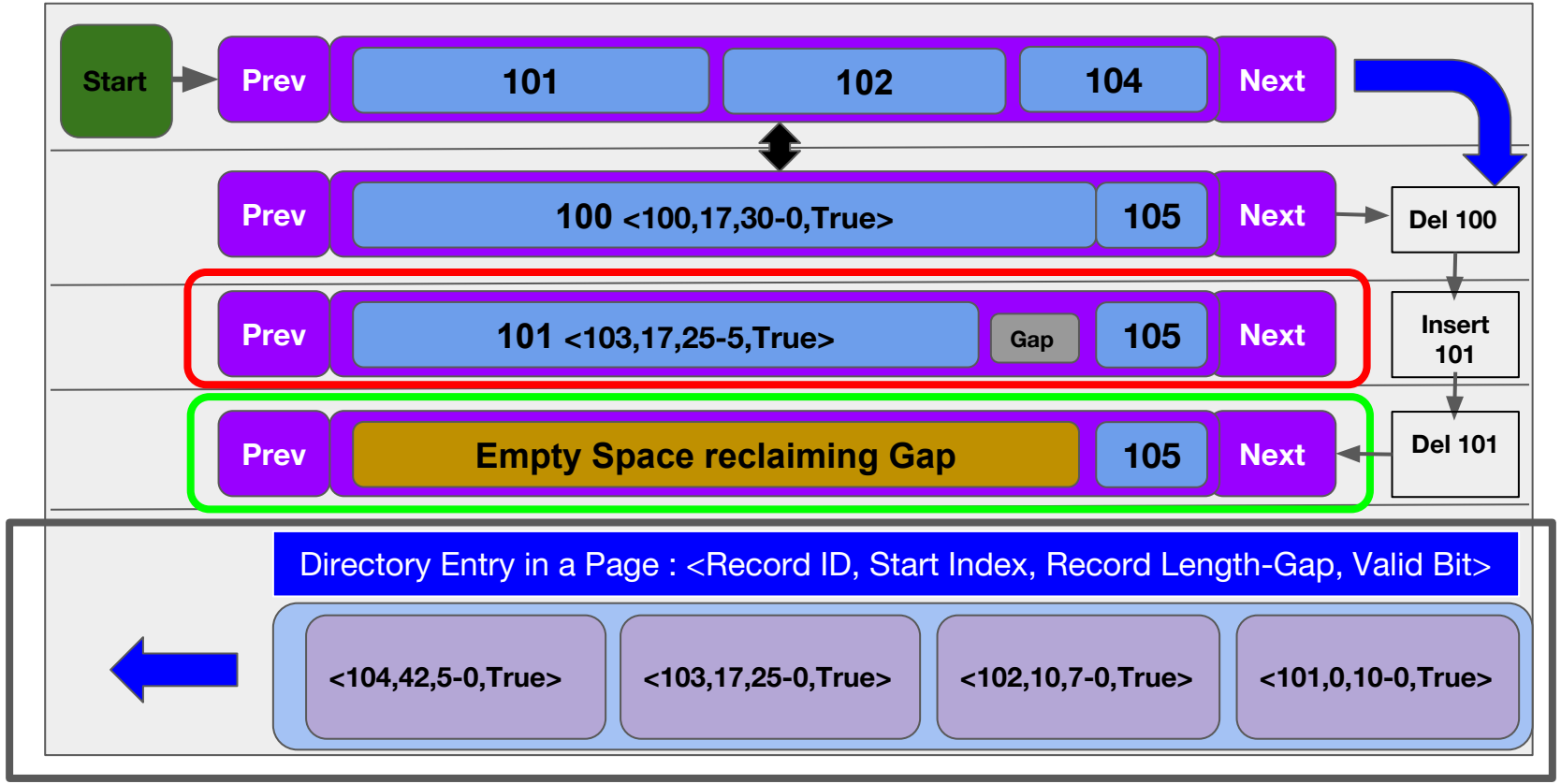
Implement Heap File with Variable Length Records

Anasua Mitra

Hosted in :- **https://github.com/sonaidgr8/HeapFile_Assignment**

DiskFile Structure

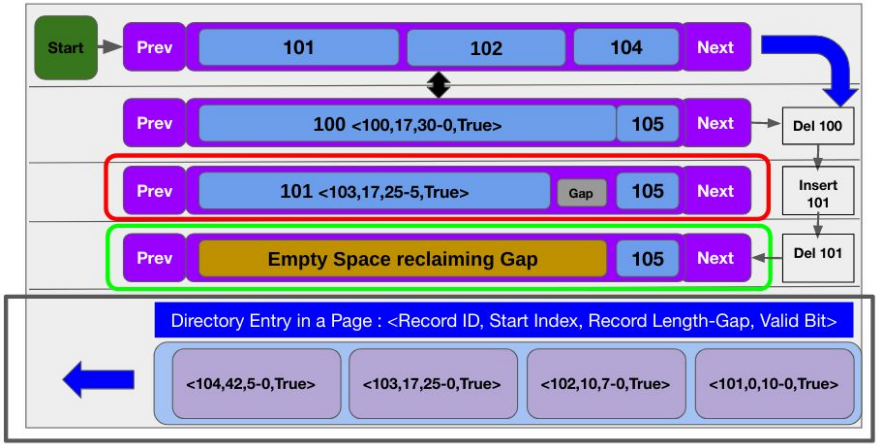


DiskFile Structure

Refer to files **DiskFile.cpp, DiskFile.h**

Data

1. struct Node {Page data; struct Node* next; struct Node* prev; };
2. struct Node * nodePointer;
3. int totalPages;



Member Functions

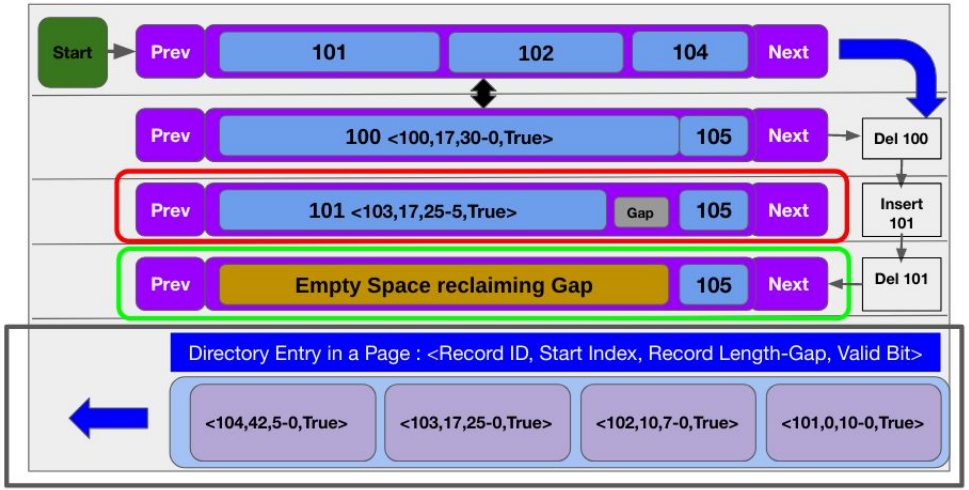
- DiskFile(){ }
- DiskFile(int n, bool * create_diskFile)
- void appendPages(struct Node** head_ref, Page new_data);
- void printPages(struct Node* node);
- void insertRecord(struct Node** head_ref, int rec_id, int rec_length, bool * inserted_records);
- void deleteRecord(struct Node** head_ref, int rec_id);

Common Variables

Refer to files **Common.h**

Variables

1. #define DISK_FILE_SIZE 1000
2. #define DISK_PAGE_SIZE 100
3. #define DIR_ENTRY_LENGTH 13

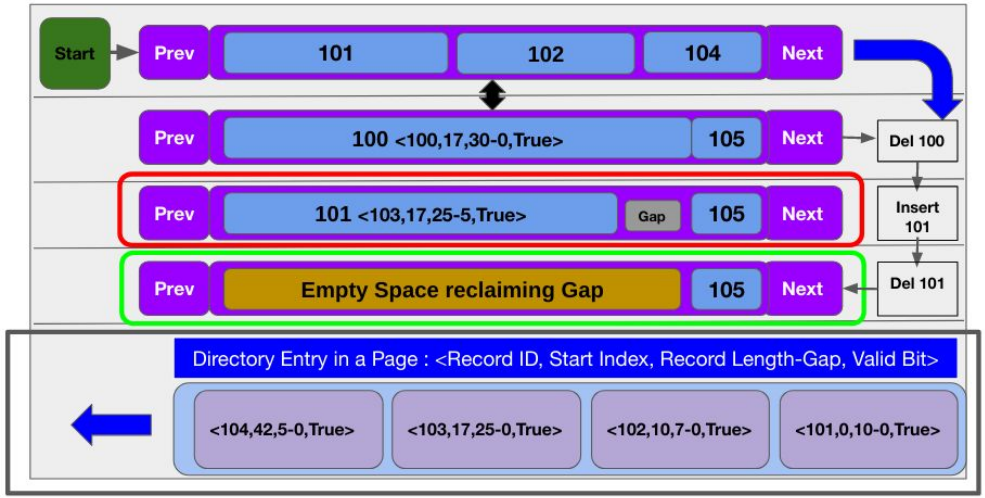


Page Structure

Refer to files **Page.h**

Variables & Functions

1. struct DirectoryEntry {
   int id;
   int length;
   int start;
   bool valid;
   - DirectoryEntry()
   - DirectoryEntry(int id, int length, int start, bool valid)}
2. vector<DirectoryEntry> arr;
3. int spaceLeft;
4. int dirSlotCount;
5. Page();



Task 1: Create Initial DiskFile

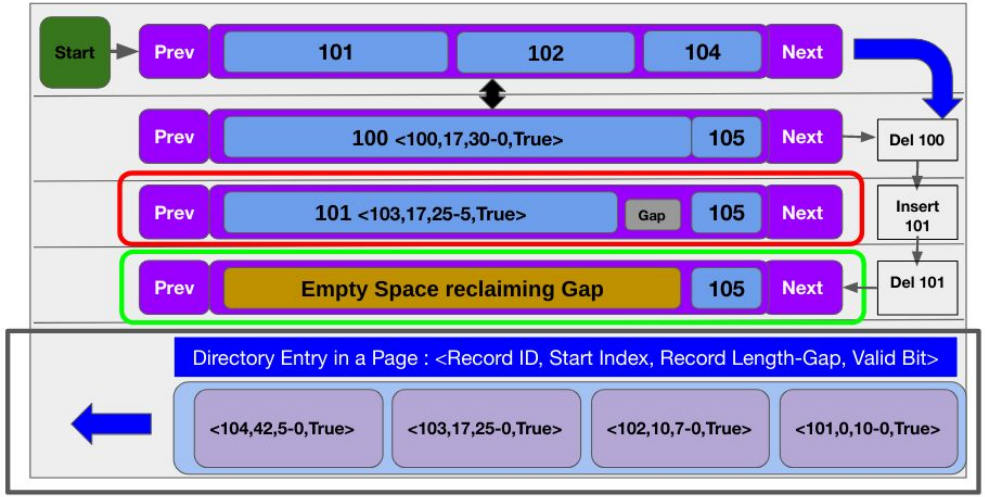
Refer to files **Main.cpp, DiskFile.h, DiskFile.cpp**

Enter :- 1:Create New DiskFile, 2:Insert Record, 3:Delete Record, 4:Display DiskFile Structure, -1:Exit

Steps

Input : Provide initial number of pages
Invokes : d = DiskFile(n, &create_diskFile);

1. Checks whether they can be accommodated in DiskFile.
2. appendPages(&this->nodePointer, Page()); // n number of times
3. printPages(this->nodePointer);
4. this->totalPages = n;
5. *create_diskFile = true;



Task 2: Insert Record

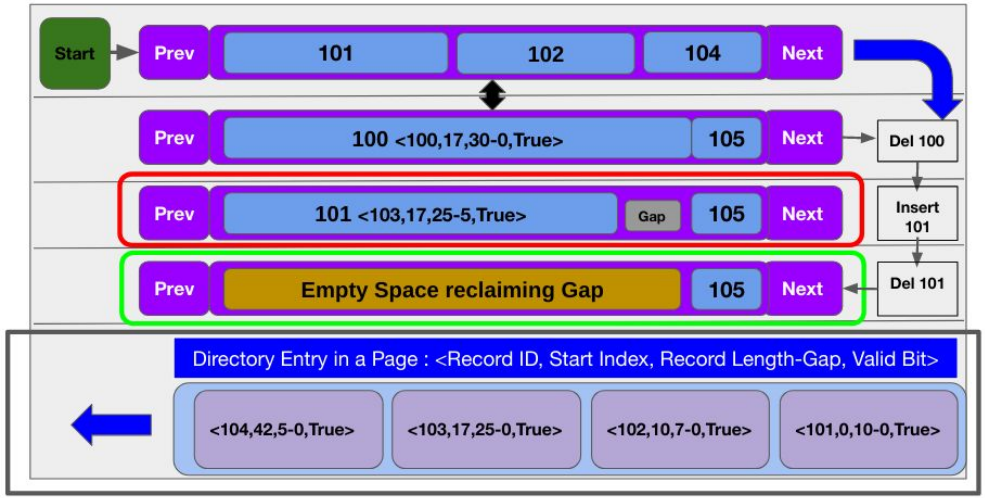
Refer to files **Main.cpp, DiskFile.h, DiskFile.cpp**

```
Enter :-  1:Create New DiskFile,  2:Insert Record,  3:Delete Record,  4:Display DiskFile Structure,  -1:Exit
```

Steps

Input : Provide record length & identifier
Invokes :
d.insertRecord(&d.nodePointer,id,l,&inserted_records);

1. Checks whether initial DiskFile has already been created.
2. d.insertRecord(&d.nodePointer,id,l,&inserted_records);
3. Various cases for insertion.



Task 2: Insert Record

Refer to files **Main.cpp, DiskFile.h, DiskFile.cpp**

```
Enter :-  1:Create New DiskFile,  2:Insert Record,  3:Delete Record,  4:Display DiskFile Structure,  -1:Exit
```

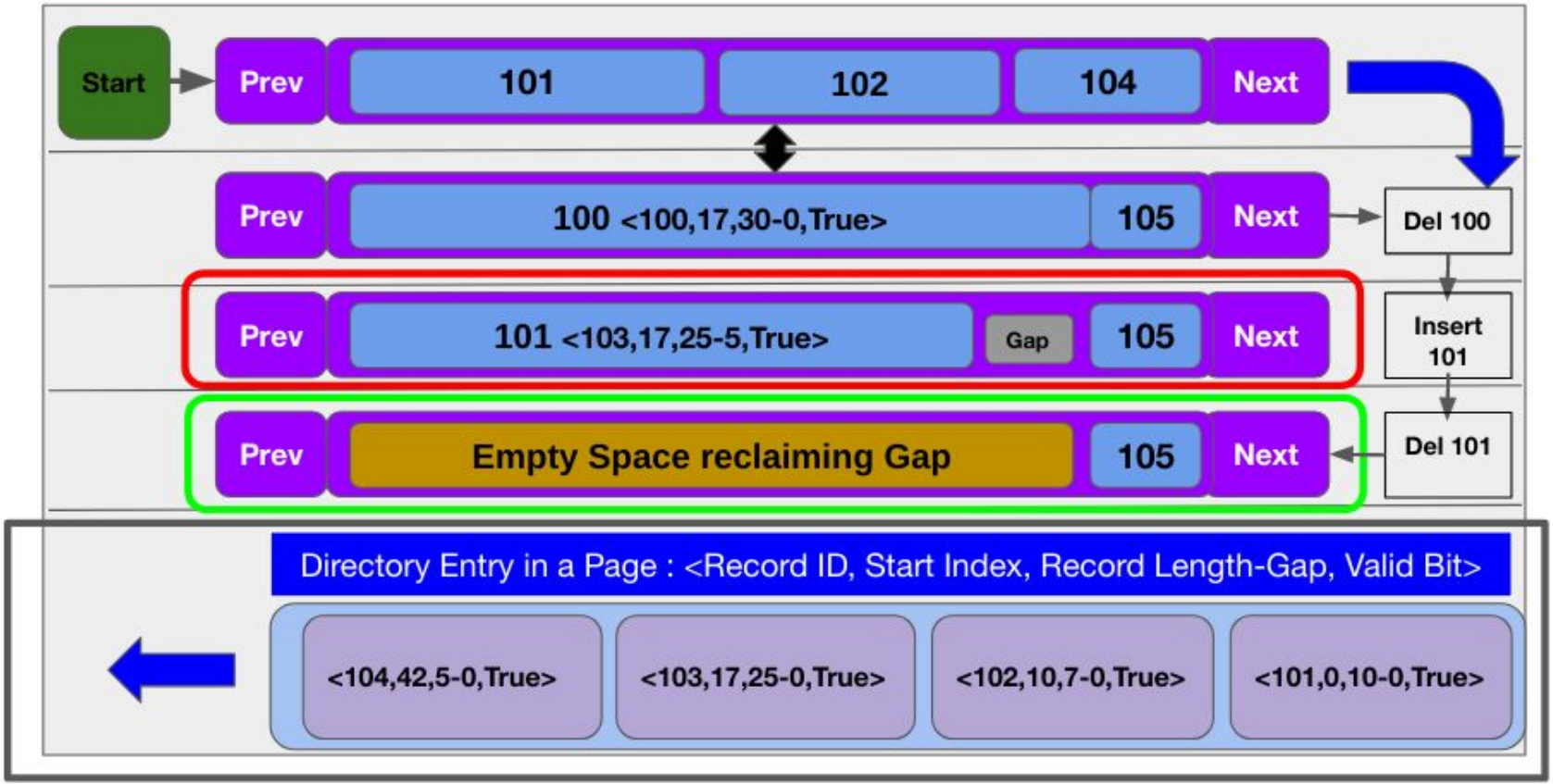
Cases for Insertion

1. Case-1: When DataPages are empty at the beginning
2. Case-2: When an empty slot is available from deletion to hold the Record

```
bool space_available = (rec_length <= (last->data.arr[i].start + last->data.arr[i].length));
```

3. Case-3: When no empty slot is available and if existing Page can accommodate the data at the end
4. Case-4: When no empty slot is available and if existing Page can not accommodate the data at the end, then go to next page to check availability
5. Case-5: When no empty slot is available and no existing Page can accommodate the data, then append a new page if maximum allowable DiskFile size is not exhausted.

Cases for Insertion & Deletion



Task 3: Delete Record

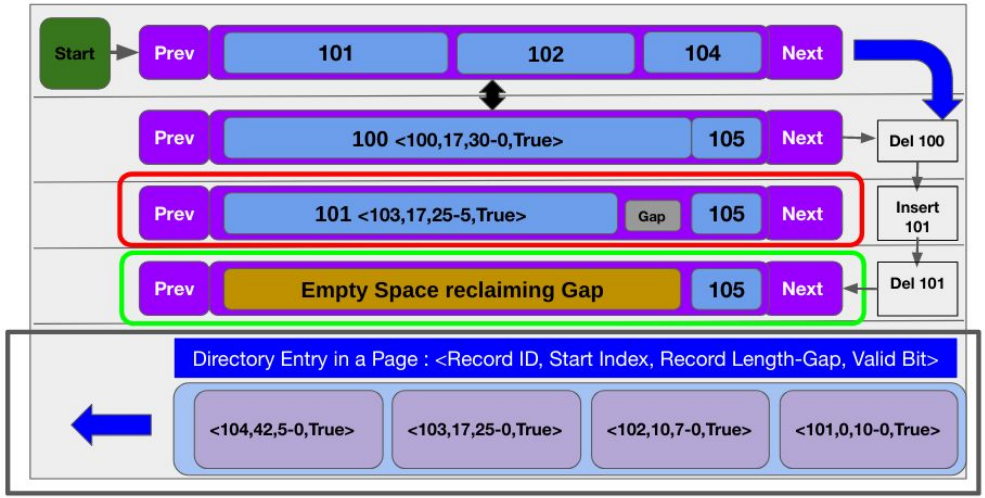
Refer to files **Main.cpp, DiskFile.h, DiskFile.cpp**

```
Enter :-  1:Create New DiskFile,  2:Insert Record,  3:Delete Record,  4:Display DiskFile Structure,  -1:Exit
```

Steps

Input : Provide record identifier
Invokes : d.deleteRecord(&d.nodePointer,id);

1. Checks whether initial DiskFile has already been created and few records are inserted.
2. d.deleteRecord(&d.nodePointer,id);
3. Various cases for deletion.



Task 3: Delete Record

Refer to files **Main.cpp, DiskFile.h, DiskFile.cpp**

```
Enter :-  1:Create New DiskFile,  2:Insert Record,  3:Delete Record,  4:Display DiskFile Structure,  -1:Exit
```

Cases for Deletion

1. Case-1: When Record is not present in any of the pages.
2. Case-2: To check for duplicate entries. It deletes all duplicate entries by traversing all the pages.
3. Case-3: Claim the left-over space through gap, restores original slot's length but not defined for last Record of a Page.

```
last->data.arr[i].id = 0;
int gap = (i == last->data.arr.size()-1) ? 0 : last->data.arr[i+1].start - (last->data.arr[i].start + last->data.arr[i].length);
last->data.arr[i].length = last->data.arr[i].length + gap;
last->data.arr[i].valid = false;
```

Task 4: Display DiskFile Structure

Refer to files **Main.cpp, DiskFile.h, DiskFile.cpp**

Enter :- 1:Create New DiskFile, 2:Insert Record, 3:Delete Record, 4:Display DiskFile Structure, -1:Exit

Steps

Input : N/A
Invokes : d.printPages(d.nodePointer);

1. Checks whether initial DiskFile has already been created.
2. d.printPages(d.nodePointer);
3. **Prints : < ID, Start, Length - Gap, Valid >**