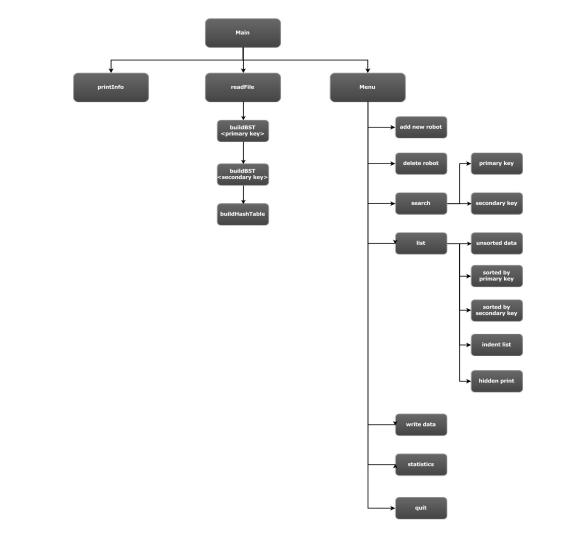


Program purpose:

- Store robot data into a text file
- Sort data by its serial number, reduce collision
- Functions for user to manage the database (insert, remove, search, print, etc)



Project Components

- Robot Class
- BST
- Queue
- Hashtable
- primeHelper
- basic file io
- Diagrams (DSD, SC, UML)

Robot class



Robot

- serialnumber: string

- model: string

- alias: string

- comment: string

- productiondate: string

//constructor

+ Robot(string, string, string, string, string)

+ Robot(const Robot &obj)

//destructor

+ getSerialNumber(): String

+ getModel(): String

+ getAlias(): String

+ getComment(): String

+ getProductionDate(): String

+ setAlias(string): void

+ setComment(string): void

+ setProductionDate(string): void

+ operator==(const Robot &r) const: bool

+ operator<(const Robot &r) const: bool

+ operator>(const Robot &r) const: bool

// friend

+ operator<<(ostream &, const Robot &): ostream &

Queue(Multiple uses)

• Temporary storage from file input

Display multiple robots with the same model key

• Remove multiple robots under the same model key



Trees and tree node

Extends

<templete ItemType> Binary Tree

- rootPtr: BinaryNode*<ItemType>
- count: int

//constructor

//destructor

- + isEmpty(): bool
- + size(): int
- + clear(): void
- + preOrder(void visit(ItemType&)): void
- + postOrder(void visit(ItemType&)): void
- + inOrder(void visit(ItemType&)): void
- + breadthOrder(void visit(ItemType&)): void
- + virtual insert(ItemType&): bool
- + virtual remove(ItemType&): bool
- + virtual getEntry(ItemType&): bool
- destroyTree(BinaryNode<ItemType>*): void
- _preOrder(void visit(ItemType&, BinaryNode<ItemType>*)): void
- _inOrder(void visit(ItemType&, BinaryNode<ItemType>*)): void
- _postOrder(void visit(ItemType&, BinaryNode<ItemType>*)): void
- _breadthOrder(void visit(ItemType&, BinaryNode<ItemType>*)): void

<templete ItemType> Binary Node

- item: ItemType
- leftPtr: BinaryNode<ItemType>*
- rightPtr: BinaryNode<ItemType>*

//constructor

//destructor

- + setItem(ItemType&): void
- + setLeftPtr(BinaryNode<ItemType>*): void
- $+\ setRightPtr(BinaryNode < ItemType >^*):\ void$
- + getItem(): ItemType
- + getLeftPtr(): BinaryNode<ItemType>*
- + getRightPtr(): BinaryNode<ItemType>*
- + isLeaf():bool

<templete ItemType> Binary Search Tree

- root: BinaryNode*
- count: int
- (*comp)(ItemType &, ItemType &): int

//constructor

BinarySearchTree(int compare(ItemType &, ItemType &))

//destructor

- + insert(ItemType&): bool
- + remove(ItemType&): bool
- + getEntry(const ItemType, &ItemType&): bool
- + getMinEntry(ItemType&): bool
- + getMaxEntry(ItemType&): bool
- _insert(BinaryNode*, BinaryNode*): BinaryNode*
- _remove(BinaryNode*, ItemType, bool): BinaryNode*
- deleteNode(BinaryNode*): BinaryNode*
- findNode(BinaryNode*, ItemType&): BinaryNode*

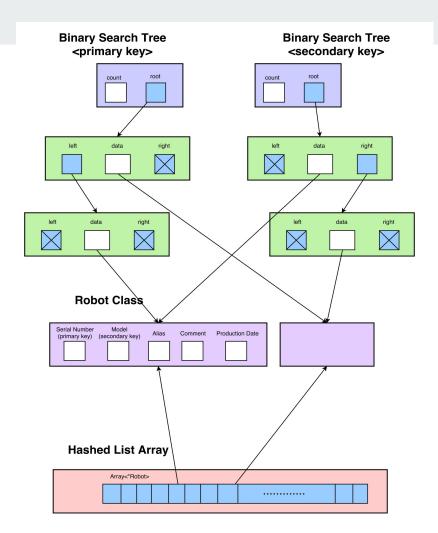
DATA STRUCTURES

hashed list array (contiguous memory)

- hash of primary key (must be string)
- hash with 2 functions ("good" and "bad")

2 binary search trees

- 1 bst for primary key
- 1 bst for secondary key



HashTable

A Good version and Bad version

Additional: primeHelper .h

To find its closest prime number to given input integer

<templete ItemType> HashTable

- table: ItemType**
- _sentinel: void*
- _size: int32_t
- _count: int32_t
- (*_hashFunction)(const ItemType &, const int32_t &): int32_t

//constructor

- + HashTable(int32_t, int32_t hashFunction(const T &, const int32_t &))
- //destructor
- + size(): int32_t
- + count(): int32 t
- + at(int32_t): ItemType*
- + print(): int
- + insert(ItemType &): int32_t
- + hash(const ItemType &): int32_t
- + find(const ItemType &): int32_t
- + remove(const int32_t &): ItemType*

Hash algorithm

```
algorithm robotHash (Robot robot, int tableSize)
     Int sum = 0
     Char *cPtr = robot.snr
     while(*cPtr)
           sum += (*cPtr)^3
           cPtr++
     End while
     Return sum % tableSize
End algorithm
```

Collisions Algorithm

will be resolved in "linear probe" like method

collision resolving method alternates between positive and negative index offsets from home address

```
if home is full

loop (offset = 1)

try home + offset

exit loop on success

if offset < 0

offset * -1 + 1
```

(offset + 1) * -1

Else

Thank you

