DSA - Seminar 2 –
ADT Bag (with Dynamic Array)

ADT Bag (also called MultiSet)

- A Bag is similar to a set, but the elements does not have to be unique.
- It is similar to a shopping bag/cart: I can buy multiple pieces of the same product and the order in which I buy the elements is not important.
- The order of the elements is not important
- There are no positions in a Bag:
  - There are no operations that take a position as parameter or that return a position.
  - The added elements are not neccesarily stored in the order in which they were added (they can be stored in this way, but there is no guarantee, and we should not make any assumptions regarding the order of the elements).

  - For example:
    - We add to an initially empty Bag the following elements: 1, 3, 2, 6, 2, 5, 2
    - If we print the content of the Bag, the elements can be printed in any order:
      - 1, 2, 2, 2, 3, 6, 5
      - 1, 3, 2, 6, 2, 5, 2
      - 1, 5, 6, 2, 3, 2, 2
      - Etc.

Domain: $\mathcal{B}$ = {b | b is a Bag with elements of the type TElem}

Interface (set of operations):

init(b)

pre : true

post: b ∈ $\mathcal{B}$, b is an empty Bag

add(b, e)

pre: b ∈ $\mathcal{B}$, e ∈ TElem

post: b' ∈ $\mathcal{B}$, b' = b U {e} (Telem *e* is added to the Bag)

remove(b, e)

pre: b ∈ $\mathcal{B}$, e ∈ TElem

post: b' ∈ $\mathcal{B}$, b' = b \ {e} (one ocurrence of e was removed from the Bag). If e is not in b, b is not changed.

search(b, e)

pre: b ∈ $\mathcal{B}$, e ∈ TElem

post: $search \leftarrow \begin{cases} true, if\ e \in \mathbf{B} \\ false, otherwise \end{cases}$

size(c)

pre: b ∈ $\mathcal{B}$

post: size ← $the\ number\ of\ elements\ from\ b$

destroy(b)

pre: b ∈ $\mathcal{B}$

post: b was destroyed

iterator(b, i)

pre: b ∈ $\mathcal{B}$

post: i ∈ $\mathcal{I}$, i is an iterator over b

ADT Iterator

- Has access to the interior structure of the Bag and it has a current element from the Bag.

Domain: $\mathcal{I}$ = {i | i is an iterator over b ∈ $\mathcal{B}$ }

Interface:

init(i, b)

pre: b ∈ $\mathcal{B}$

post: i ∈ $\mathcal{I}$, i is an iterator over b

valid(i)

pre: i ∈ $\mathcal{I}$

post: $valid \leftarrow \begin{cases} true, if\ the\ current\ element\ from\ i\ is\ a\ valid\ one \\ false, otherwise \end{cases}$

next(i)

pre: i ∈ $\mathcal{I}$, valid(i)

post: i' ∈ $\mathcal{I}$, the current element from i' refers to the next element from the bag b.

getCurrent(i, e)

pre: i ∈ $\mathcal{I}$, valid(i)

post: e ∈ TElem, e is the current element from i

Reprezentation:

1.

- A dynamic array of elements, where every element can appear multiple times.
- The iterator will have a current position from the dynamic array

1	3	2	6	2	5	2

2.

- A dynamic array of unique elements, and for each element its frequency (either two dynamic arrays, or one single dynamic array of pairs).
- The iterator will hve a current position and a current frequency for the element.

(1,1)	(3,1)	(2,3)	(5,1)	(6,1)

C++ implementation (some operations can have other correct implementations as well):

1. **Version 1**

File Bag.h

```
#ifndef _BAG_H
#define _BAG_H

	typedef int TElem;
	typedef struct _Iterator Iterator; //forward declaration for the iterator

	typedef struct _Bag{
		int len;
		int cap;
		TElem* elements;
	} Bag;

	Bag init();
	void destroy(Bag* b);
	void add(Bag* b, TElem elem);
	void remove(Bag* b, TElem elem);
	bool search(Bag* b, TElem elem);
	int size(Bag* b);
	Iterator iterator(Bag* b);
	//other possible operations for copying, assigning, etc.

#endif
```

File Bag.cpp

```
#include <stdlib.h>
#include "Bag.h"
#include "Iterator.h"

Bag init() {
	Bag b;
	b.len = 0;
	b.cap = 10;
	b.elements = (TElem*)malloc(b.cap * sizeof(TElem));
	return b;
}

void destroy(Bag* b) {
	b->len = 0;
	b->cap = 0;
	free(b->elements);
	b->elements = NULL;
}


void add(Bag* b, TElem elem) {
	if (b->len == b->cap) {
		b->cap = b->cap * 2;
		TElem* newElements = (TElem*)malloc(b->cap * sizeof(TElem));
		for (int i = 0; i < b->len; i++) {
			newElements[i] = b->elements[i];
		}
		free(b->elements);
		b->elements = newElements;
	}
	b->elements[b->len] = elem;
	b->len++;
}

void remove(Bag* b, TElem elem) {
	bool found = false;
	int pos = 0;

	while (found == false && pos < b->len) {
		if (b->elements[pos] == elem) {
			found = true;
		}
		else {
			pos = pos + 1;
		}
	}

	if (found) {
		for (int i = pos; i < b->len - 1; i++) {
			b->elements[i] = b->elements[i + 1];
		}
		b->len--;
	}
}
```

```
bool search(Bag* b, TElem elem) {
	bool found = false;
	int pos = 0;

	while (found == false && pos < b->len) {
		if (b->elements[pos] == elem) {
			found = true;
		}
		else {
			pos = pos + 1;
		}
	}
	return found;
}

int size(Bag* b) {
	return b->len;
}

Iterator iterator(Bag* b) {
	Iterator it =  init(b);
	return it;
   }
```

File Iterator.h

```
#ifndef __ITERATOR_H
#define __ITERATOR_H

	#include "Bag.h"

	typedef struct _Iterator{
		Bag* bag;
		int currentPos;
	} Iterator;


	Iterator init(Bag* b);
	TElem getCurrent(Iterator* it);
	bool valid(Iterator* it);
	void next(Iterator* it);
#endif
```

File Iterator.cpp

```
#include "Iterator.h"
#include "Bag.h"

Iterator init(Bag* b) {
	Iterator it;
	it.bag = b;
	it.currentPos = 0;
	return it;
}

TElem getCurrent(Iterator* it) {
	return it->bag->elements[it->currentPos];
}

bool valid(Iterator* it) {
	return it->currentPos < it->bag->len;
}

void next(Iterator* it) {
	it->currentPos++;
}
```

Test program:

```
#include <iostream>
#include "Bag.h"
#include "Iterator.h"

void addNumbersToBag(Bag* b) {
	add(b, 10);
	add(b, 11);
	add(b, 19);
	add(b, 13);
	add(b, 22);
	add(b, 13);
	add(b, 21);
	add(b, 19);
	add(b, 15);
	add(b, 10);
	add(b, 9);
	add(b, 10);
	add(b, 15);
}

void printBag(Bag* b) {
	Iterator it = iterator(b);
	while (valid(&it)) {
		TElem elem = getCurrent(&it);
		std::cout << elem << std::endl;
		next(&it);
	}
}

int main() {
	Bag b = init();
	remove(&b, 11);
	addNumbersToBag(&b);
	std::cout << size(&b) << std::endl;
	remove(&b, 13);
	remove(&b, 13);
	remove(&b, 13);
	printBag(&b);
	std::cout << size(&b) << std::endl;
	destroy(&b);
}
```

2. Verson 2

File BagF.h

```
#ifndef _BAGF_H
#define _BAGF_H

	typedef int TElem;
	typedef struct _IteratorF IteratorF;

	typedef struct _BagF {
		int len;
		int cap;
		TElem* elements;
		int* frequencies;
	} BagF;

	BagF initF();
	void destroy(BagF* b);
	void add(BagF* b, TElem elem);
	void remove(BagF* b, TElem elem);
	bool search(BagF* b, TElem elem);
	int size(BagF* b);
	IteratorF iterator(BagF* b);
	//other possible operations for copying, assigning, etc.


#endif
```

File BagF.cpp

```
#include <stdlib.h>
#include "BagF.h"
#include "IteratorF.h"

BagF initF() {
	BagF b;
	b.len = 0;
	b.cap = 10;
	b.elements = (TElem*)malloc(b.cap * sizeof(TElem));
	b.frequencies = (int*)malloc(b.cap * sizeof(int));
	return b;
}

void destroy(BagF* b) {
	b->len = 0;
	b->cap = 0;
	free(b->elements);
	free(b->frequencies);
	b->elements = NULL;
	b->frequencies = NULL;
}

void add(BagF* b, TElem elem) {
	int pos = 0;
	bool found = false;

	while (!found && pos < b->len) {
		if (b->elements[pos] == elem) {
			found = true;
		}
		else {
			pos++;
		}
	}
	if (found) {
		b->frequencies[pos]++;
	}
	else {
		if (b->len == b->cap) {
			b->cap = b->cap * 2;
			TElem* newElements = (TElem*)malloc(b->cap * sizeof(TElem));
			int* newFrequencies = (int*)malloc(b->cap * sizeof(int));
			for (int i = 0; i < b->len; i++) {
				newElements[i] = b->elements[i];
				newFrequencies[i] = b->frequencies[i];
			}
			free(b->elements);
			free(b->frequencies);
			b->elements = newElements;
			b->frequencies = newFrequencies;
		}

		b->elements[b->len] = elem;
		b->frequencies[b->len] = 1;
		b->len++;
```

```
	}
}

void remove(BagF* b, TElem elem) {
	bool found = false;
	int pos = 0;

	while (found == false && pos < b->len) {
		if (b->elements[pos] == elem) {
			found = true;
		}
		else {
			pos = pos + 1;
		}
	}

	if (found) {
		if (b->frequencies[pos] > 1) {
			b->frequencies[pos]--;
		}
		else {
			for (int i = pos; i < b->len - 1; i++) {
				b->elements[i] = b->elements[i + 1];
				b->frequencies[i] = b->frequencies[i + 1];
			}
			b->len--;
		}
	}
}

bool search(BagF* b, TElem elem) {
	bool found = false;
	int pos = 0;

	while (found == false && pos < b->len) {
		if (b->elements[pos] == elem) {
			found = true;
		}
		else {
			pos = pos + 1;
		}
	}
	return found;
}

int size(BagF* b) {
	int size = 0;
	for (int i = 0; i < b->len; i++) {
		size += b->frequencies[i];
	}
	return size;
}


IteratorF iterator(BagF* b) {
	IteratorF it = init(b);
	return it;
   }
```

File IteratorF.h

```
#ifndef __ITERATORF_H
#define __ITERATORF_H

	#include "BagF.h"

	typedef struct _IteratorF {
		BagF* bag;
		int currentPos;
		int currentFrecv;
	} IteratorF;

	IteratorF init(BagF* b);
	TElem getCurrent(IteratorF* it);
	bool valid(IteratorF* it);
	void next(IteratorF* it);

#endif
```

File IteratorF.cpp

```
#include "IteratorF.h"
#include "BagF.h"

IteratorF init(BagF* b) {
	IteratorF it;
	it.bag = b;
	it.currentPos = 0;
	it.currentFrecv = 1;
	return it;
}

TElem getCurrent(IteratorF* it) {
	return it->bag->elements[it->currentPos];
}

bool valid(IteratorF* it) {
	return it->currentPos < it->bag->len;
}

void next(IteratorF* it) {
	if (it->currentFrecv < it->bag->frequencies[it->currentPos]) {
		it->currentFrecv++;
	}
	else {
		it->currentPos++;
		it->currentFrecv = 1;
	}
}
```