Jeff Morton
Data Structures and Algorithms 2
Project 4
Functional Decomposition

User-defined data structures used as parameters in the functions:

```
typedef struct Item *ItemP;
typedef struct Item
{
        float size;
        ItemP next;
} Item;


typedef struct Bin
{
        float sizeUsed, sizeFree;
        int numItems;
        ItemP first;
}Bin, *BinP;


typedef struct ItemArray
{
        ItemP *items;
        int size;
}ItemArray, *ItemArrayP;


typedef struct BinArray
{
        BinP *bins;
        int binsTotal, binsUsed;
}BinArray, *BinArrayP;

typedef struct Node *NodeP;
typedef struct Node
{
        double priority;
        void *info;
} Node;
```

```
typedef struct Heap
{
        int size;
        int maxSize;
        Node *array;
} Heap;
```

Files and Functions in the Program:

```
/*************************************************************
 * Student Name: Jeff Morton
 * File Name: BinPacker.c
 * Assignment Number: 4
 * Date Due: April 5, 2016
 *
 *  Created on: Mar 30, 2016
 *      Author: Jeff Morton
 *************************************************************/
```

BinPacker.c (Use for BinPacker.h as well, same functions)

```
/**
 * adds the item passed to the last item in the linkedlist
 * @param {ItemP} list - the item containing the linkedList
 * @param {ItemP} item - the item to add
 */
void addItemToList( ItemP list, ItemP item );

/**
 * prints the linkedList of items passed (the bin)
 * @param {ItemP} item - the linkedList to print
 */
void printBin( ItemP item );

/**
 * Clones the itemArray passed and sets all next pointers to NULL
 * @param {ItemArrayP} srcItems - the array of items to clone
 * @param {ItemArrayP} newItems - the new array of cloned items
 * @return ItemArrayP - the new array of items
 */
ItemArrayP cloneItemArray( ItemArrayP srcItems, ItemArrayP newItems );

/**
```

```
 * Resets the array of bins so that they can be repacked.
 * @param {BinArrayP} bins - the array of bins to reset
 */
void resetBins( BinArrayP bins );

/**
 * randomly permutes the ItemArray passed
 * @param {ItemArrayP} items - the itemarray to permute
 */
void permuteItems( ItemArrayP items );

/**
 * Allocates memory for a new bin and initializes it
 * @return BinP the newly created bin
 */
BinP newBin();

/**
 * allocates and initializes a new item
 * @param {float} size - the size of the item
 * @return ItemP the item created
 */
ItemP newItem( float size );

/**
 * reads items in from items.txt, creates items for them, stores them in an array, then
returns that array
 * @return ItemArrayP the array of items
 */
ItemArrayP readItems();

/**
 * creates an array of size number of bins
 * @param {int} size - the number of bins to create
 * @return BinArrayP the array of bins
 */
BinArrayP getBins( int size );

/**
 * First fit method of packing the bins
 * @param {BinArrayP} binArray - the array of bins to be packed
 * @param {ItemArrayP} itemArray - the array of items to be packed
 */
```

```
void firstFit( BinArrayP binArray, ItemArrayP itemArray );

/**
 * Next fit method of packing the bins
 * @param {BinArrayP} binArray - the array of bins to be packed
 * @param {ItemArrayP} itemArray - the array of items to be packed
 */
void nextFit( BinArrayP binArray, ItemArrayP itemArray );

/**
 * Best fit method of packing the bins
 * @param {BinArrayP} binArray - the array of bins to be packed
 * @param {ItemArrayP} itemArray - the array of items to be packed
 */
void bestFit( BinArrayP binArray, ItemArrayP itemArray );

/**
 * sorts items in the ItemArray passed in descending order
 * @param {ItemArrayP} itemArray - the array of items to sort
 */
void sortItems( ItemArrayP itemArray );

/**
 * prints the item array passed
 * @param {ItemArrayP} itemArray - the item array to print
 */
void printArray( ItemArrayP itemArray );

/**
 * prints the bin array passed
 * @param {BinArrayP} binArray - the item array to print
 */
void printBins( BinArrayP binArray );

/**
 * executes a search based genetic algorithm that calculates 50 permutations of the item
array passed
 * keeps the best permutation (determined by the number of bins resulting from best fit
algorithm)
 * returns the best bin packing achieved
 * @param {ItemArrayP} items - the array of items to pack
 * @return BinArrayP - the best packed array of bins after 50 permutations
 */
```

```
        BinArrayP searchBasedPacking( ItemArrayP items );

        /**
         * Sets the next pointer of every item in the array to null
         * @param {ItemArrayP} items - the array of items to reset
         */
        void resetItems( ItemArrayP items );

/*************************************************************
 * Student Name: Jeff Morton
 * File Name: DynamicHeap.c
 * Assignment Number: 4
 * Date Due: April 5, 2016
 *
 *  Created on: Feb 14, 2016
 *      Author: Jeff Morton (jhm14@students.uwf.edu)
 *************************************************************/
```

DynamicHeap.c (Use for DynamicHeap.h as well, same functions)

```
        /**
         * newHeap
         * malloc's a new Heap struct and returns a StructP pointer to it.
         * Remember, this pointer must be type-casted by the program utilizing the ADT
         * @param {int} size - the size of the heap to create
         * @return HeapP pointer to the new Heap
         */
        HeapP newHeap( int size );

        /**
         * Inserts a new element into the heap with the priority given
         */
        void heapInsert(HeapP heap, double priority, void *info);

        /**
         * Extracts an element from the heap with the lowest priority.
         * returns the element as a void pointer
         */
        void * heapExtract(HeapP heap);

        /**
         * returns an element from the heap with the lowest priority.
         */
        void * findMin(HeapP heap);
```

```
	/**
	 * returns true if the heap is full, otherwise false.
	 */
	int isFull(HeapP heap);

	/**
	 * returns true if empty, otherwise false
	 */
	int isEmpty(HeapP heap);

	/**
	 * returns the size of the heap
	 */
	int getSize(HeapP heap);

	/**
	 * returns the max size of the heap
	 */
	int getMaxSize(HeapP heap);

	/**
	 * prints the entire heap
	 */
	void printHeap(HeapP heap);

/*****************************************************************
 * Student Name: Jeff Morton
 * File Name: Main.c
 * Assignment Number: 4
 * Date Due: April 5, 2016
 *
 *  Created on: Mar 29, 2016
 *      Author: Jeff Morton
 *****************************************************************/
```

Main.c

```
	/**
	 * the main function
	 */
	int main(int argc, char *argv[]);
```