// FILE: DPQueue.h

// CLASS PROVIDED: p\_queue (priority queue ADT)

//

// TYPEDEFS and MEMBER CONSTANTS for the p\_queue class:

// typedef \_\_\_\_\_ value\_type

// p\_queue::value\_type is the data type of the items in

// the p\_queue. It may be any of the C++ built-in types

// (int, char, etc.), or a class with a default constructor, a

// copy constructor, an assignment operator, and a less-than

// operator forming a strict weak ordering.

//

// typedef \_\_\_\_\_ size\_type

// p\_queue::size\_type is the data type considered best-suited

// for any variable meant for counting and sizing (as well as

// array-indexing) purposes; e.g.: it is the data type for a

// variable representing how many items are in the p\_queue.

// It is also the data type of the priority associated with

// each item in the p\_queue

//

// static const size\_type DEFAULT\_CAPACITY = \_\_\_\_\_

// p\_queue::DEFAULT\_CAPACITY is the default initial capacity of a

// p\_queue that is created by the default constructor.

//

// CONSTRUCTOR for the p\_queue class:

// p\_queue(size\_type initial\_capacity = DEFAULT\_CAPACITY)

// Pre: initial\_capacity > 0

// Post: The p\_queue has been initialized to an empty p\_queue.

// The push function will work efficiently (without allocating

// new memory) until this capacity is reached.

// Note: If Pre is not met, initial\_capacity will be adjusted to

// DEFAULT\_CAPACITY. I.e., when creating a p\_queue object,

// client can override initial\_capacity with something deemed

// more appropriate than DEFAULT\_CAPACITY; but if (in doing so)

// client mis-specifies 0 (NOTE: size\_type is unsigned, thus

// can't be negative) as the overriding size, DEFAULT\_CAPACITY

// remains as the value to be used for initial\_capacity (this

// is to ensure no attempt is made at allocating memory that's

// 0 in amount).

//

// MODIFICATION MEMBER FUNCTIONS for the p\_queue class:

// void push(const value\_type& entry, size\_type priority)

// Pre: (none)

// Post: A new copy of item with the specified data and priority

// has been added to the p\_queue.

//

// void pop()

// Pre: size() > 0.

// Post: The highest priority item has been removed from the

// p\_queue. (If several items have the equal priority,

// then the implementation may decide which one to remove.)

//

// CONSTANT MEMBER FUNCTIONS for the p\_queue class:

// size\_type size() const

// Pre: (none)

// Post: The return value is the total number of items in the

// p\_queue.

//

// value\_type front() const

// Pre: size() > 0.

// Post: The return value is the data of the highest priority

// item in the p\_queue, but the p\_queue is unchanged.

// (If several items have equal priority, then the

// implementation may decide which one to return.)

//

// bool empty() const

// Pre: (none)

// Post: The return value is true if the p\_queue is empty,

// otherwise false.

//

// VALUE SEMANTICS for the p\_queue class:

// Assignments and the copy constructor may be used with p\_queue

// objects.

#ifndef D\_P\_QUEUE\_H

#define D\_P\_QUEUE\_H

#include <cstdlib> // provides size\_t

namespace CS3358\_FA2019\_A7

{

class p\_queue

{

public:

// TYPEDEFS and MEMBER CONSTANTS

typedef int value\_type;

typedef size\_t size\_type;

static const size\_type DEFAULT\_CAPACITY = 1;

// CONSTRUCTORS AND DESTRUCTOR

p\_queue(size\_type initial\_capacity = DEFAULT\_CAPACITY);

p\_queue(const p\_queue& src);

~p\_queue();

// MODIFICATION MEMBER FUNCTIONS

p\_queue& operator=(const p\_queue& rhs);

void push(const value\_type& entry, size\_type priority);

void pop();

// CONSTANT MEMBER FUNCTIONS

size\_type size() const;

bool empty() const;

value\_type front() const;

// EXTRA CONSTANT MEMBER FUNCTION FOR DEBUG PRINTING

void print\_tree(const char message[] = "", size\_type i = 0) const;

void print\_array(const char message[] = "") const;

private:

// STRUCT to store information about one item in the p\_queue

struct ItemType

{

value\_type data;

size\_type priority;

};

// PRIVATE MEMBER VARIABLES

ItemType \*heap;

size\_type capacity;

size\_type used;

// HELPER FUNCTIONS

void resize(size\_type new\_capacity);

bool is\_leaf(size\_type i) const;

size\_type parent\_index(size\_type i) const;

size\_type parent\_priority(size\_type i) const;

size\_type big\_child\_index(size\_type i) const;

size\_type big\_child\_priority(size\_type i) const;

void swap\_with\_parent(size\_type i);

};

}

#endif