My Project

Generated by Doxygen 1.8.17

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

Class Documentation

2.1 Packet Struct Reference

Models a Packet. We are only interesed in the arrival time and how much time is spent to be processed.

`#include <packet_processor.hpp>`

Public Member Functions

- **Packet** (int arrival_time, int process_time)

Public Attributes

- int **arrival_time**
- int **process_time**

2.1.1 Detailed Description

Models a Packet. We are only interesed in the arrival time and how much time is spent to be processed.

CopyRight F. J. Madrid-Cuevas `fjmadrid@uco.es`

Sólo se permite el uso de este código en la docencia de las asignaturas sobre Estructuras de Datos de la Universidad de Córdoba.

Está prohibido su uso para cualquier otro objetivo.

The documentation for this struct was generated from the following file:

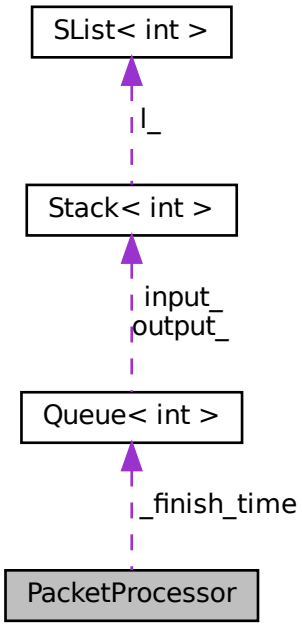
- packet_processor.hpp

2.2 PacketProcessor Class Reference

Models the packet processor.

`#include <packet_processor.hpp>`

Collaboration diagram for PacketProcessor:



Public Member Functions

- PacketProcessor (size_t size)

  Create a packet processor with a queue for at most size packets.
- Response process (const Packet &packet)

  generate the response when a packet arrives.

Protected Attributes

- size_t _max_size
- Queue< int > _finish_time

2.2.1 Detailed Description

Models the packet processor.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 PacketProcessor()

```
PacketProcessor::PacketProcessor (
            size_t size )
```

Create a packet processor with a queue for at most size packets.

CopyRight F. J. Madrid-Cuevas fjmadrid@uco.es

Sólo se permite el uso de este código en la docencia de las asignaturas sobre Estructuras de Datos de la Universidad de Córdoba.

Está prohibido su uso para cualquier otro objetivo.

2.2.3 Member Data Documentation

2.2.3.1 _finish_time

```
Queue<int> PacketProcessor::_finish_time  [protected]
```

Saves the finishing times for packets waiting to be processed

2.2.3.2 _max_size

```
size_t PacketProcessor::_max_size  [protected]
```

Max number of packets.

The documentation for this class was generated from the following files:

- packet_processor.hpp
- packet_processor.cpp

2.3 Queue< T > Class Template Reference

ADT Queue. Models a queue of T.

```
#include <queue.hpp>
```

Public Member Functions

Life cicle.

- Queue ()
  Create an empty Queue.
- ~Queue ()
  Destroy a Queue.

Observers

- bool is_empty () const
  is the list empty?.
- size_t size () const
  Get the number of items in the queue.
- T front () const
  get the front item (the oldest one).
- T back () const
  get the back item (the newest one).

Modifiers

- void enque (const T &new_it)
  Insert a new item.
- void deque ()
  Remove the front item.

Protected Member Functions

- void flush_input_to_output ()
  Flush input stack into the output stack.

Protected Attributes

- Stack< T >::Ref **input_**
- Stack< T >::Ref **output_**
- T **back_**

2.3.1 Detailed Description

template<class T>
class Queue< T >

ADT Queue. Models a queue of T.

CopyRight F. J. Madrid-Cuevas fjmadrid@uco.es

Sólo se permite el uso de este código en la docencia de las asignaturas sobre Estructuras de Datos de la Universidad de Córdoba.

Está prohibido su uso para cualquier otro objetivo.

2.3.2 Constructor & Destructor Documentation

2.3.2.1 Queue()

```
template<class T >
Queue< T >::Queue
```

Create an empty Queue.

Postcondition

is_empty()

2.3.3 Member Function Documentation

2.3.3.1 back()

```
template<class T >
T Queue< T >::back
```

get the back item (the newest one).

Precondition

not is_empty()

2.3.3.2 deque()

```
template<class T >
void Queue< T >::deque
```

Remove the front item.

Precondition

not is_empty()

Postcondition

size()==(old.size()-1)

2.3.3.3 enque()

```
template<class T >
void Queue< T >::enque (
            const T & new_it )
```

Insert a new item.

Postcondition

back() == new_it

size()==(old.size()+1)

2.3.3.4 flush_input_to_output()

```
template<class T >
void Queue< T >::flush_input_to_output  [protected]
```

Flush input stack into the output stack.

Precondition

!is_empty()

2.3.3.5 front()

```
template<class T >
T Queue< T >::front
```

get the front item (the oldest one).

Precondition

not is_empty()

The documentation for this class was generated from the following files:

- queue.hpp
- queue_imp.hpp

2.4 Response Struct Reference

Models the response to a incomming packet. The package can be processed at any given time or be dropped if the processor buffer is full at the time of arrival.

```
#include <packet_processor.hpp>
```

Public Member Functions

- **Response** (bool dropped, int start_time)

Public Attributes

- bool **dropped**
- int **start_time**

2.4.1 Detailed Description

Models the response to a incomming packet. The package can be processed at any given time or be dropped if the processor buffer is full at the time of arrival.

The documentation for this struct was generated from the following file:

- packet_processor.hpp

2.5 SList< T > Class Template Reference

ADT SList. Models a Single linked list[T].

`#include <slist.hpp>`

Public Types

- typedef std::shared_ptr< SList< T > > Ref

  Define a shared reference to a SNode. Manages the storage of a pointer, providing a limited garbage-collection facility, possibly sharing that management with other objects.

Public Member Functions

Observers

- bool is_empty () const

  is the list empty?.
- size_t size () const

  Get the number of items in the list.
- T front () const

  Get the head's item of the list.
- T current () const

  get the current item.
- bool has_next () const

  Is there a next item?
- T next () const

  Get the next item.
- bool has (T const &it) const

  Has it the item data?
- void fold (std::ostream &out) const

Fold to an output stream.

Modifiers

- void set_current (T const &new_v)
  Set a new value for current.
- void push_front (T const &new_it)
  insert an item as the new list's head.
- void insert (T const &new_it)
  insert a new item after current.
- void pop_front ()
  Remove the head. @prec !is_empty()
- void remove ()
  Remove current item.
- void goto_next ()
  Move the cursor to the next list's item.
- void goto_first ()
  Move the cursor to the list's head.
- bool find (T const &it)
  Move the cursor to the first occurrence of a value from the head of the list. If the item is not found, the cursor will be at the end of the list.
- bool find_next (T const &it)
  Move the cursor to the next occurrence of a value from current. If the item is not found, the cursor will be at the end of the list.

Protected Member Functions

- SNode< T >::Ref head () const
  Get a reference to the head node.
- SNode< T >::Ref curr () const
  Get a reference to the current node.

Life cicle.

- SList ()
  Create an empty Stack.
- ~SList ()
  Destroy a Stack.
- static SList< T >::Ref create ()
  Create a SList using dynamic memory.
- static SList< T >::Ref create (std::istream &in) noexcept(false)
  Create a SList unfoldig from an input stream.

2.5.1 Detailed Description

template<class T>
class SList< T >

ADT SList. Models a Single linked list[T].

2.5.2 Member Typedef Documentation

2.5.2.1 Ref

```
template<class T >
typedef std::shared_ptr< SList<T> > SList< T >::Ref
```

Define a shared reference to a SNode. Manages the storage of a pointer, providing a limited garbage-collection facility, possibly sharing that management with other objects.

See also

http://www.cplusplus.com/reference/memory/shared_ptr/

2.5.3 Constructor & Destructor Documentation

2.5.3.1 SList()

```
template<class T >
SList< T >::SList
```

Create an empty Stack.

Postcondition

is_empty()

2.5.4 Member Function Documentation

2.5.4.1 create() [1/2]

```
template<class T >
SList< T >::Ref SList< T >::create  [static]
```

Create a SList using dynamic memory.

Returns

a shared referente to the new slist.

2.5.4.2 create() [2/2]

```
template<class T >
SList< T >::Ref SList< T >::create (
            std::istream & in )  [static], [noexcept]
```

Create a SList unfoldig from an input stream.

The input format will be "[ ]" for the empty list or "[" item1 item2 ... item_n "]" where item1 is the head.

Parameters

in	is the input stream.

Warning

if the input format is not correct a std::runtime_error with what message "Wrong input format." will be reaised.

Returns

A shared referente to the new slist.

2.5.4.3 curr()

```
template<class T >
SNode<T>::Ref SList< T >::curr ( ) const  [protected]
```

Get a reference to the current node.

Returns

a reference to the current node.

2.5.4.4 current()

```
template<class T >
T SList< T >::current
```

get the current item.

Precondition

not is_empty()

2.5.4.5 find()

```
template<class T >
bool SList< T >::find (
            T const & it )
```

Move the cursor to the first occurrence of a value from the head of the list. If the item is not found, the cursor will be at the end of the list.

Parameters

it	is the value to be found.

Returns

true if it is found.

Precondition

!is_empty()

Postcondition

!ret_val || item()==it

ret_value || !has_next()

2.5.4.6 find_next()

```
template<class T >
bool SList< T >::find_next (
            T const & it )
```

Move the cursor to the next occurrence of a value from current. If the item is not found, the cursor will be at the end of the list.

Parameters

it	is the value to be found.

Returns

true if it is found.

Precondition

has_next()

Postcondition

!ret_val || item()==it

ret_value || !has_next()

2.5.4.7 fold()

```
template<class T >
void SList< T >::fold (
            std::ostream & out ) const
```

Fold to an output stream.

The format will be "[ ]" for the empty list or '[' item1 item2 item3 ... item_n ']' where item1 is the head.

Parameters

out	is the output stream.

Returns

the output stream.

2.5.4.8 front()

```
template<class T >
T SList< T >::front
```

Get the head's item of the list.

Returns

the item at the head.

Precondition

!is_empty()

2.5.4.9 goto_first()

```
template<class T >
void SList< T >::goto_first
```

Move the cursor to the list's head.

Precondition

!is_empty()

Postcondition

current()==front()

2.5.4.10 goto_next()

```
template<class T >
void SList< T >::goto_next
```

Move the cursor to the next list's item.

Precondition

has_next()

Postcondition

old.next()==current()

2.5.4.11 has()

```
template<class T >
bool SList< T >::has (
            T const & it ) const
```

Has it the item data?

Parameters

in	*it*	is the item to find.

Returns

true if the item is into the list.

2.5.4.12 has_next()

```
template<class T >
bool SList< T >::has_next
```

Is there a next item?

Returns

true if there is.

Precondition

!is_empty()

2.5.4.13 head()

```
template<class T >
SNode< T >::Ref SList< T >::head  [protected]
```

Get a reference to the head node.

Returns

a reference to the head node.

2.5.4.14 insert()

```
template<class T >
void SList< T >::insert (
            T const & new_it )
```

insert a new item after current.

Parameters

new_it	is the item to insert.

Postcondition

old.is_empty() implies front()==current()==new_it

!old.is_empty() implies current()==old.current() && has_next() && next()==new_it

size()==(old.size()+1)

2.5.4.15 next()

```
template<class T >
T SList< T >::next
```

Get the next item.

Returns

the next item data.

Precondition

has_next()

2.5.4.16 pop_front()

```
template<class T >
void SList< T >::pop_front
```

Remove the head. @prec !is_empty()

Postcondition

is_empty() || front() == "next of old.front()".

size()==(old.size()-1)

2.5.4.17 push_front()

```
template<class T >
void SList< T >::push_front (
            T const & new_it )
```

insert an item as the new list's head.

Parameters

new↩ _it	is the item to insert.

Postcondition

front()==new_it

size()==(old.size()+1)

2.5.4.18 remove()

```
template<class T >
void SList< T >::remove
```

Remove current item.

Precondition

!is_empty()

Postcondition

old.has_next() implies current()==old.next()

!old.has_next() implies is_empty() || current()=="old previous item."

size()==(old.size()-1)

2.5.4.19 set_current()

```
template<class T >
void SList< T >::set_current (
            T const & new_v )
```

Set a new value for current.

Parameters

new_v	is the new value.

Precondition

!is_empty()

Postcondition

item()==new_v

The documentation for this class was generated from the following files:

- slist.hpp
- slist_imp.hpp

2.6 SNode< T > Class Template Reference

a single link node.

`#include <slist.hpp>`

Public Types

- typedef std::shared_ptr< SNode< T > > Ref

  Define a shared reference to a SNode. Manages the storage of a pointer, providing a limited garbage-collection facility, possibly sharing that management with other objects.

Public Member Functions

Observers.

- T item () const

  Get the data item.
- bool has_next () const

  Has it a next node?.
- SNode< T >::Ref next () const

  Get the link to next element.

Modifiers.

- void set_item (const T &new_it)

  Set the data item.
- void set_next (SNode< T >::Ref next)

  Set the link to the next node.

Protected Attributes

- T **_item**
- SNode< T >::Ref **_next**

Life cicle.

- SNode (T const &it)

  Create a node.
- SNode (T const &it, SNode< T >::Ref &next)

  Create an empty Stack.
- ~SNode ()

  Destroy a SNode.
- static SNode< T >::Ref create (T const &it, SNode< T >::Ref next=nullptr)

  Create a SNode using dynamic memory.

2.6.1 Detailed Description

template<class T>
class SNode< T >

a single link node.

CopyRight F. J. Madrid-Cuevas fjmadrid@uco.es

Sólo se permite el uso de este código en la docencia de las asignaturas sobre Estructuras de Datos de la Universidad de Córdoba.

Está prohibido su uso para cualquier otro objetivo.

2.6.2 Member Typedef Documentation

2.6.2.1 Ref

```
template<class T >
typedef std::shared_ptr< SNode<T> > SNode< T >::Ref
```

Define a shared reference to a SNode. Manages the storage of a pointer, providing a limited garbage-collection facility, possibly sharing that management with other objects.

See also

http://www.cplusplus.com/reference/memory/shared_ptr/

2.6.3 Constructor & Destructor Documentation

2.6.3.1 SNode()

```
template<class T >
SNode< T >::SNode (
            T const & it )
```

Create a node.

Postcondition

!has_next()

2.6.4 Member Function Documentation

2.6.4.1 create()

```
template<class T >
SNode< T >::Ref SNode< T >::create (
            T const & it,
            SNode< T >::Ref next = nullptr )  [static]
```

Create a SNode using dynamic memory.

Parameters

it	the value save in the node.
next	link to the next node.

Returns

a shared referente to the new node.

2.6.4.2 has_next()

```
template<class T >
bool SNode< T >::has_next
```

Has it a next node?.

Returns

true if it has a next node.

The documentation for this class was generated from the following files:

- slist.hpp
- slist_imp.hpp

2.7 Stack< T > Class Template Reference

ADT Stack. Models a Stack using a single linked list*.

`#include <stack.hpp>`

Public Types

- typedef std::shared_ptr< Stack< T > > Ref

  Define a shared reference to a Stack. Manages the storage of a pointer, providing a limited garbage-collection facility, possibly sharing that management with other objects.

Public Member Functions

Observers

- bool is_empty () const

  is the list empty?.
- size_t size () const

  Get the number of items in the stack.
- T top () const

  get the top item.
- void fold (std::ostream &out) const

  Fold the stack to an output stream. The output format is like the slist.

Modifiers

- void push (const T &new_it)

  Insert a new item.
- void pop ()

Protected Attributes

- SList< T >::Ref **l_**

Life cicle.

- Stack ()

  Create an empty Stack.
- ~Stack ()

  Destroy a Stack.
- static Stack< T >::Ref create ()

  Create a Stack using dynamic memory.
- static Stack< T >::Ref create (std::istream &in) noexcept(false)

  Create a Stack from an input stream. The input format is the same of a single list.

2.7.1 Detailed Description

template<class T>
class Stack< T >

ADT Stack. Models a Stack using a single linked list*.

CopyRight F. J. Madrid-Cuevas fjmadrid@uco.es

Sólo se permite el uso de este código en la docencia de las asignaturas sobre Estructuras de Datos de la Universidad de Córdoba.

Está prohibido su uso para cualquier otro objetivo.

2.7.2 Member Typedef Documentation

2.7.2.1 Ref

```
template<class T >
typedef std::shared_ptr< Stack<T> > Stack< T >::Ref
```

Define a shared reference to a Stack. Manages the storage of a pointer, providing a limited garbage-collection facility, possibly sharing that management with other objects.

See also

http://www.cplusplus.com/reference/memory/shared_ptr/

2.7.3 Constructor & Destructor Documentation

2.7.3.1 Stack()

```
template<class T >
Stack< T >::Stack
```

Create an empty Stack.

Postcondition

is_empty()

2.7.4 Member Function Documentation

2.7.4.1 create() [1/2]

```
template<class T >
Stack< T >::Ref Stack< T >::create  [static]
```

Create a Stack using dynamic memory.

Returns

a shared referente to the new stack.

2.7.4.2 create() [2/2]

```
template<class T >
Stack< T >::Ref Stack< T >::create (
            std::istream & in )  [static], [noexcept]
```

Create a Stack from an input stream. The input format is the same of a single list.

Returns

a shared referente to the new stack.

Warning

throw std::runtime_error if wrong input format.

2.7.4.3 fold()

```
template<class T >
void Stack< T >::fold (
            std::ostream & out ) const
```

Fold the stack to an output stream. The output format is like the slist.

Parameters

out	is the output stream.

2.7.4.4 pop()

```
template<class T >
void Stack< T >::pop
```

Remove the top item.

Precondition

not is_empty()

Postcondition

size() = old.size()-1

2.7.4.5 push()

```
template<class T >
void Stack< T >::push (
            const T & new_it )
```

Insert a new item.

Postcondition

top() == new_it

size() = old.size()+1

2.7.4.6 top()

```
template<class T >
T Stack< T >::top
```

get the top item.

Precondition

not is_empty()

The documentation for this class was generated from the following files:

- stack.hpp
- stack_imp.hpp

Index