

My Project

Generated by Doxygen 1.8.17

1 Class Index	1
1.1 Class List	1
2 Class Documentation	3
2.1 Packet Struct Reference	3
2.1.1 Detailed Description	3
2.2 PacketProcessor Class Reference	4
2.2.1 Detailed Description	4
2.2.2 Constructor & Destructor Documentation	5
2.2.2.1 PacketProcessor()	5
2.2.3 Member Data Documentation	5
2.2.3.1 _finish_time	5
2.2.3.2 _max_size	5
2.3 Queue< T > Class Template Reference	5
2.3.1 Detailed Description	6
2.3.2 Constructor & Destructor Documentation	7
2.3.2.1 Queue()	7
2.3.3 Member Function Documentation	7
2.3.3.1 back()	7
2.3.3.2 deque()	7
2.3.3.3 enqueue()	8
2.3.3.4 flush_input_to_output()	8
2.3.3.5 front()	8
2.4 Response Struct Reference	8
2.4.1 Detailed Description	9
2.5 SList< T > Class Template Reference	9
2.5.1 Detailed Description	10
2.5.2 Member Typedef Documentation	11
2.5.2.1 Ref	11
2.5.3 Constructor & Destructor Documentation	11
2.5.3.1 SList()	11
2.5.4 Member Function Documentation	11
2.5.4.1 create() [1/2]	11
2.5.4.2 create() [2/2]	11
2.5.4.3 curr()	12
2.5.4.4 current()	12
2.5.4.5 find()	12
2.5.4.6 find_next()	13
2.5.4.7 fold()	14
2.5.4.8 front()	14
2.5.4.9 goto_first()	14
2.5.4.10 goto_next()	15

2.5.4.11 has()	15
2.5.4.12 has_next()	15
2.5.4.13 head()	16
2.5.4.14 insert()	16
2.5.4.15 next()	16
2.5.4.16 pop_front()	17
2.5.4.17 push_front()	17
2.5.4.18 remove()	17
2.5.4.19 set_current()	18
2.6 SNode< T > Class Template Reference	18
2.6.1 Detailed Description	19
2.6.2 Member Typedef Documentation	19
2.6.2.1 Ref	19
2.6.3 Constructor & Destructor Documentation	19
2.6.3.1 SNode()	20
2.6.4 Member Function Documentation	20
2.6.4.1 create()	20
2.6.4.2 has_next()	20
2.7 Stack< T > Class Template Reference	21
2.7.1 Detailed Description	22
2.7.2 Member Typedef Documentation	22
2.7.2.1 Ref	22
2.7.3 Constructor & Destructor Documentation	22
2.7.3.1 Stack()	23
2.7.4 Member Function Documentation	23
2.7.4.1 create() [1/2]	23
2.7.4.2 create() [2/2]	23
2.7.4.3 fold()	23
2.7.4.4 pop()	24
2.7.4.5 push()	24
2.7.4.6 top()	24
Index	25

Chapter 1

Class Index

1.1 Class List

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

Packet	Models a Packet . We are only interested in the arrival time and how much time is spent to be processed	3
PacketProcessor	Models the packet processor	4
Queue< T >	ADT Queue . Models a queue of T	5
Response	Models the response to a incoming packet. The package can be processed at any given time or be dropped if the processor buffer is full at the time of arrival	8
SList< T >	ADT SList . Models a Single linked list[T]	9
SNode< T >	Single link node	18
Stack< T >	ADT Stack . Models a Stack using a single linked list*	21

Chapter 2

Class Documentation

2.1 Packet Struct Reference

Models a [Packet](#). We are only interested 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 interested 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:

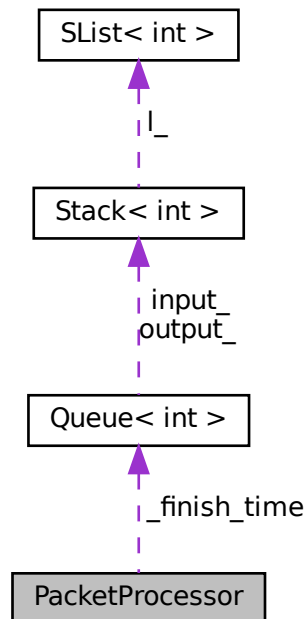
- packet_processor.hpp

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 cycle.

- `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 enqueue (const T &new_it)`
Insert a new item.
- `void dequeue ()`
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()`

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.

2.3.3 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 enqueue()

```
template<class T >
void Queue< T >::enqueue (
    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 incoming 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 incoming 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

<i>in</i>	is the input stream.
-----------	----------------------

Warning

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

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

<i>it</i>	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

<i>it</i>	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

<i>out</i>	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

<code>in</code>	<code><i>it</i></code>	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

<i>new_it</i>	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

<i>new_it</i>	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

<i>new_v</i>	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()

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.

2.6.4 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

<i>it</i>	the value save in the node.
<i>next</i>	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 cycle.

- `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\(\)](#)

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.

2.7.4 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

<code>out</code>	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

- [_finish_time](#)
 - [PacketProcessor, 5](#)
 - [_max_size](#)
 - [PacketProcessor, 5](#)
- [back](#)
 - [Queue< T >, 7](#)
- [create](#)
 - [SList< T >, 11](#)
 - [SNode< T >, 20](#)
 - [Stack< T >, 23](#)
- [curr](#)
 - [SList< T >, 12](#)
- [current](#)
 - [SList< T >, 12](#)
- [deque](#)
 - [Queue< T >, 7](#)
- [enqueue](#)
 - [Queue< T >, 7](#)
- [find](#)
 - [SList< T >, 12](#)
- [find_next](#)
 - [SList< T >, 13](#)
- [flush_input_to_output](#)
 - [Queue< T >, 8](#)
- [fold](#)
 - [SList< T >, 13](#)
 - [Stack< T >, 23](#)
- [front](#)
 - [Queue< T >, 8](#)
 - [SList< T >, 14](#)
- [goto_first](#)
 - [SList< T >, 14](#)
- [goto_next](#)
 - [SList< T >, 14](#)
- [has](#)
 - [SList< T >, 15](#)
- [has_next](#)
 - [SList< T >, 15](#)
 - [SNode< T >, 20](#)
- [head](#)
 - [SList< T >, 15](#)
- [insert](#)
 - [SList< T >, 16](#)

- [next](#)
 - [SList< T >, 16](#)
- [Packet, 3](#)
- [PacketProcessor, 4](#)
 - [_finish_time, 5](#)
 - [_max_size, 5](#)
 - [PacketProcessor, 5](#)
- [pop](#)
 - [Stack< T >, 24](#)
- [pop_front](#)
 - [SList< T >, 16](#)
- [push](#)
 - [Stack< T >, 24](#)
- [push_front](#)
 - [SList< T >, 17](#)
- [Queue](#)
 - [Queue< T >, 7](#)
- [Queue< T >, 5](#)
 - [back, 7](#)
 - [deque, 7](#)
 - [enqueue, 7](#)
 - [flush_input_to_output, 8](#)
 - [front, 8](#)
 - [Queue, 7](#)
- [Ref](#)
 - [SList< T >, 11](#)
 - [SNode< T >, 19](#)
 - [Stack< T >, 22](#)
- [remove](#)
 - [SList< T >, 17](#)
- [Response, 8](#)
- [set_current](#)
 - [SList< T >, 17](#)
- [SList](#)
 - [SList< T >, 11](#)
- [SList< T >, 9](#)
 - [create, 11](#)
 - [curr, 12](#)
 - [current, 12](#)
 - [find, 12](#)
 - [find_next, 13](#)
 - [fold, 13](#)
 - [front, 14](#)
 - [goto_first, 14](#)
 - [goto_next, 14](#)
 - [has, 15](#)

- has_next, [15](#)
- head, [15](#)
- insert, [16](#)
- next, [16](#)
- pop_front, [16](#)
- push_front, [17](#)
- Ref, [11](#)
- remove, [17](#)
- set_current, [17](#)
- SList, [11](#)
- SNode
 - SNode< T >, [19](#)
- SNode< T >, [18](#)
 - create, [20](#)
 - has_next, [20](#)
 - Ref, [19](#)
 - SNode, [19](#)
- Stack
 - Stack< T >, [22](#)
- Stack< T >, [21](#)
 - create, [23](#)
 - fold, [23](#)
 - pop, [24](#)
 - push, [24](#)
 - Ref, [22](#)
 - Stack, [22](#)
 - top, [24](#)
- top
 - Stack< T >, [24](#)