Jonas Franz

Snips Documentation & Manual

Contents

1. About SNIPS

2. Language Syntax

Type System

Primitives

Snips has a range of built-in primitive types. All of the types are 1 32-bit word large.

Void. The void type acts as a universal primitive type. Using this type requires caution, because of its unique property to match every other type. This is not only true for primitives, but for every possible type. This can be used to create "don't care" types. But, to improve type security, this should be avoided. Also, since the void type is 1 word large, assigning a expression to this type that is larger than 1 word can cause major problems. See Void Arrays for a workaround. A Void is declared with the type specifier $void$. For example:

```
void v = 20;
```

Integer. The integer represents the most common numeric type, with a range from $-2^{31}-1$ to 2^{31}. An Integer is declared with the type specifier int. For example:

```
int x = 10;
```

Boolean. The boolean type is mostly used for logic. Its value range is the exact same as the one of the Integer, but when interpreting the value, only the value 0 is interpreted as $false$, all other possible values are interpreted as $true$. A boolean is declared with the type specifier $bool$. For example:

```
bool b = true;
```

Character. The character type can be used to store UTF-8 encoded Characters. The value Range is, because of the 32-bit size, like the one of the Integer, but the usable Range is only from 0 to 255. The 0 value takes a special role. In a String, a char with the value 0 indicates the end of the String. A character is declared with the type specifier $char$. For example:

```
char c = 'a';
```

Enumerations. The enumeration type is not necessarily a primitive type, but is handled like one on the low level. An enumeration has to be declared, see Enum Typedef. After Declaration, an Enum value can be selected. An Enum is declared with the name of the Enum itself. For Example:

```
State s = State.Normal;
```

Composite Types

3. Built in Libraries

Data Structures

linked_list.sn

Full Path: lib/std/data/linked_list.sn
Usage: A data structure that can hold a variable number of elements.
Package Type: Base Package
Namespacing: The entire package is namespaced in 'List'.
Description:

Provides a data structure that can hold a variable number of items. Each element is capsuled in a own node. Using pointers, these nodes are chained together. The nodes are stored on the heap.

Struct Name	Struct Fields	Description
ListNode<T>	ListNode<T>* next T value	Capsules a single data element. Also contains a pointer to the next list node in the list. Is set to 0 if this is the last element in the list.
LinkedList<T>	ListNode<T>* head ListNode<T>* tail	Capsules two list node pointers, one pointing to the first, and one to the last element. Both pointers are initialized and set to 0 whenever the pointer has no target.

Function Header	Description
LinkedList<T>* create<T>()	Initializes a new linked list struct. Sets the pointers to 0. $O(1)$
void destroy(LinkedList<void>* l)	Destroys the given list and all of its contained elements by freeing every node and the list itself from the heap. $O(n)$
void add<T>(LinkedList<T>* lp, T x)	Adds given x to the list by encapsuling it in a new list node and appending the element at the end of the list. This uses the tail pointer of the list, so the complexity is $O(1)$.
T get<T>(LinkedList<T>* lp, int i)	Returns the value of the i-th element in the given list. $O(n)$
ListNode<T>* getNode<T>(LinkedList<T>* lp, int i)	Returns a pointer to the list node in the list at the i-th position in the list. $O(n)$
bool contains<T>(LinkedList<T>* lp, T x)	Checks wether given value is the value of one of the contained list nodes. Returns true if the value is found. $O(n)$
ListNode<T>* find<T>(LinkedList<T>* lp, T x)	Attempts to find the first list node in order in the list that has the given value x. Returns a pointer to this list node. $O(n)$
int size(LinkedList<void>* lp)	Returns the number of list nodes in the given list. $O(n)$

Function Header	Description
void remove(LinkedList<void>* lp, int i)	Returns the list node at the i-th position. Updates the pointers of the neighbours to fill the gap. Also frees the removed node. $O(n)$

Math

matrix.sn

Full Path: lib/std/math/matrix.sn
Usage: Contains utility for matrix operations.
Package Type: Base Package
Namespacing: The entire package is namespaced in 'Matrix'.
Description:

Contains matrix utility. Currently only matrix multiplication.

Function Headers	Description
int mult(int* m, int* n, int dim0, int dim1)	Multiplies the two given matrices. Stores the result in an integer array on the heap. Returns a pointer to the result. The first matrix should have dim0 columns, the second matrix should have dim1 columns. The resulting matrix will have the dimensions $dim0 \times dim1$. Does not check matrix size compatibility.

vector.sn

Full Path: lib/std/math/vector.sn
Usage: Contains utility for vectors.
Package Type: Base Package
Namespacing: The entire package is namespaced in 'Vector'.
Description:

Contains utility functions for vector operations, like the scalar product.

Function Headers	Description
int scalarProd(int* v, int* w, int l)	Calculates the scalar product of the two given vectors. Both vectors have to have the given length. Returns the result.

Utility

bits.sn

Full Path: lib/util/bits.sn
Usage: Used to manipulate single bits in 1-Word types
Package Type: Base Package
Namespacing: None
Description:

This library can be used to create something like bit-banks. Meaning a single Integer can hold 32 States for an automaton, flag-set etc. These bits can easily be modified with this library.

Function Headers	Description
bool isBitSet(int x, int i)	Checks wether the bit at the i-th place in given word x is a 1. The lowest bit has the index 0.
int setBit(int target, int i, bool val)	Sets the bit in the target word at the i-th place to given val. Returns the resulting data word.
int toggleBit(int target, int i)	Sets the bit in given target word at the i-th place to its complement. Returns the resulting word.
Int clearBit(int target, int i)	Sets the bit in given target word at the i-th place to 0. Returns the resulting word.