

# SaC v1.4

new in v1.4:

- support for non-recursive structs
- support for gpukernel pragmas
- support for tensor comprehensions

## 1 Program Structure

*prg* ⇒ [( *module* | *class* )] [*interface* ]\*  
          [*structdef* ]\* [*typedef* ]\* [*objectdef* ]\*  
          [*function* ]\*

## 2 Module Declarations

*module* ⇒ **module** *id* [**deprecated** *str* ] ;  
*class* ⇒ **class** *id* [**deprecated** *str* ] ; *classtype*  
*classtype* ⇒ **classtype** *type* ;  
          | **extern classtype** ; [*interface\_pragma* ]\*

## 3 Import / Export

*interface* ⇒ ( **import** / **use** ) *id* : *symbolset* ;  
          | ( **export** / **provide** ) *symbolset* ;  
*symbolset* ⇒ **all** [**except** { *ext\_id* [, *ext\_id* ] } ]  
          | { *ext\_id* [, *ext\_id* ] }

## 4 Structure Definitions

*structdef* ⇒ **struct** *id* { [*type id* [, *id* ]\* ; ]\* } ;

## 5 Type Definitions

*typedef* ⇒ *loctypedef*  
          | *exttypedef*  
*loctypedef* ⇒ **typedef** *type id* ;  
*exttypedef* ⇒ **external typedef** *id* ; [*interface\_pragma* ]\*

## 6 Object Definitions

*objectdef* ⇒ ( *locobjdef* | *extobjdef* )  
*locobjdef* ⇒ **objdef** *type id = funcall* ;  
*extobjdef* ⇒ **external objdef** *type id ; [interface\_pragma ]\**

## 7 Function Declarations and Definitions

*function* ⇒ *extfundec* [( *interface\_pragma* | *function\_pragma* ) ]\*  
| *specfundec* [*function\_pragma* ]\*  
| *fundef*  
| *main*  
*extfundec* ⇒ **external** *varsignature* ;  
*specfundec* ⇒ **specialize** *fixsignature* ;  
*fundef* ⇒ [( **inline** | **noinline** ) ] *fixsignature* [*function\_pragma* ]\* *body*  
*fixsignature* ⇒ *fixrets ext\_id ( fixargs )*  
| *operator\_sig*  
*varsignature* ⇒ *varrets ext\_id ( varargs )*  
| *operator\_sig*  
*operator\_sig* ⇒ *type ( ext\_op ) ( arg )*  
| *type ( ext\_op ) ( arg , arg )*  
*fixargs* ⇒ ( *arg* [, *arg* ]\* | [**void**] )  
*varargs* ⇒ *fixargs*  
| *arg* [, *arg* ]\* , ...  
*arg* ⇒ *type* [&] *id*  
*fixrets* ⇒ ( *type* [, *type* ]\* | [**void**] )  
*varrets* ⇒ *fixrets*  
| *type* [, *type* ]\* , ...  
*main* ⇒ **int main** ( [**void**] ) *body*

## 8 Function Bodies

<i>body</i>	⇒	{ [ <i>catchsim_pragma</i> ] [ <i>vardec</i> ]* [ <i>statement</i> ]* [ <i>return</i> ] }
<i>vardec</i>	⇒	<i>type id</i> [, <i>id</i> ]* ;
<i>statement</i>	⇒	;   <i>assignment</i> ;   <i>funcall</i> ;   <i>withloop</i> ;   <i>cond</i>   <i>doloop</i>   <i>whileloop</i>   <i>forloop</i>
<i>return</i>	⇒	<b>return</b> [ <i>expr</i> ] ;   <b>return</b> ( [ <i>exprs</i> ] ) ;
<i>assignment</i>	⇒	<i>assign_lhs</i> [, <i>assign_lhs</i> ]* <i>assign_op</i> <i>expr</i>   <i>assign_lhs</i> ( ++   - )
<i>assign_lhs</i>	⇒	<i>id</i>   <i>assign_lhs</i> . <i>id</i>   <i>assign_lhs</i> [ <i>exprs</i> ]
<i>assign_op</i>	⇒	( =   +=   -=   *=   /=   %= )
<i>cond</i>	⇒	<b>if</b> ( <i>expr</i> ) <i>statementblock</i> [ <b>else</b> <i>statementblock</i> ]
<i>doloop</i>	⇒	<b>do</b> <i>statementblock</i> <b>while</b> ( <i>expr</i> ) ;
<i>whileloop</i>	⇒	<b>while</b> ( <i>expr</i> ) <i>statementblock</i>
<i>forloop</i>	⇒	<b>for</b> ( <i>assignment</i> [, <i>assignment</i> ]* ; <i>expr</i> ; <i>assignment</i> [, <i>assignment</i> ]* ) <i>statementblock</i>
<i>statementblock</i>	⇒	{ [ <i>catchsim_pragma</i> ] [ <i>statement</i> ]* }   <i>statement</i>

## 9 Expressions

<i>exprs</i>	$\Rightarrow$	<i>expr</i> [ , <i>expr</i> ]*
<i>expr_or_dot</i>	$\Rightarrow$	( <i>expr</i>   . )
<i>expr_or_mdots</i>	$\Rightarrow$	( <i>expr</i>   .   ... )
<i>expr_or_ass</i>	$\Rightarrow$	( <i>expr</i>   . <u>id</u> = <i>expr</i> )
<i>expr</i>	$\Rightarrow$	<i>const</i>   <i>qual_ext_id</i>   <i>funcall</i>   <i>withloop</i>   <i>tensor_comp</i>   <i>array</i>   <i>struct</i>   <i>expr</i>    <i>expr</i>   <i>expr</i> && <i>expr</i>   <i>expr</i> ? <i>expr</i> : <i>expr</i>   ( <i>type</i> ) <i>expr</i>   ( <i>expr</i> )
<i>array</i>	$\Rightarrow$	[ [ <i>exprs</i> ] ]   [ : <i>type</i> ]   <i>expr</i> [ [ <i>expr_or_mdots</i> [ , <i>expr_or_mdots</i> ]* ] ]
<i>struct</i>	$\Rightarrow$	( ( <i>structtype</i>   <i>usertype</i> ) ) { [ <i>expr_or_ass</i> [ , <i>expr_or_ass</i> ]* ] }   <i>expr</i> . <u>id</u>
<i>funcall</i>	$\Rightarrow$	<i>qual_ext_id</i> ( [ <i>exprs</i> ] )   <i>unary_prf</i> ( <i>expr</i> )   <i>qual_ext_op</i> <i>expr</i>   <i>binary_prf</i> ( <i>expr</i> , <i>expr</i> )   <i>expr</i> <i>qual_ext_op</i> <i>expr</i>   <i>ternary_prf</i> ( <i>expr</i> , <i>expr</i> , <i>expr</i> )
<i>tensor_comp</i>	$\Rightarrow$	{ <i>tc_def</i> [ ; <i>tc_def</i> ]* }
<i>tc_def</i>	$\Rightarrow$	<u>id</u> -> <i>expr</i> [   <i>tc_constraint</i> ]   [ [ <i>id_or_mdots</i> [ , <i>id_or_mdots</i> ]* ] ] -> <i>expr</i> [   <i>tc_constraint</i> ]
<i>tc_constraint</i>	$\Rightarrow$	<i>expr</i> ( <   <= ) ( <u>id</u>   <i>id_vec</i> ) [ <b>step</b> <i>expr</i> [ <b>width</b> <i>expr</i> ] ]   ( <u>id</u>   <i>id_vec</i> ) ( <   <= ) <i>expr</i> [ <b>step</b> <i>expr</i> [ <b>width</b> <i>expr</i> ] ]   <i>expr</i> ( <   <= ) ( <u>id</u>   <i>id_vec</i> ) ( <   <= ) <i>expr</i>   [ <b>step</b> <i>expr</i> [ <b>width</b> <i>expr</i> ] ]

## 10 With-Loops

<i>withloop</i>	⇒	<b>with</b> [ <i>generators</i> ] : <i>operations</i>
<i>generators</i>	⇒	{ [ <i>withloop_pragma</i> ] [ <i>generator</i> ]* }
<i>generator</i>	⇒	( <i>index_set</i> ) [ <i>generator_pragma</i> ] [{ [ <i>statement</i> ]* } ] : <i>gen_exprs</i> ;
<i>index_set</i>	⇒	<i>expr_or_dot</i> (<   <=) <i>index_vars</i> (<   <=) <i>expr_or_dot</i> [ <b>step</b> <i>expr</i> [ <b>width</b> <i>expr</i> ]]
<i>index_vars</i>	⇒	<i>id</i> [= <i>id_vec</i> ]   <i>id_vec</i>
<i>id_vec</i>	⇒	[ [ <i>id</i> [, <i>id</i> ]* ] ]
<i>gen_exprs</i>	⇒	<b>void</b>   <i>expr</i>   ( <i>expr</i> [, <i>expr</i> ]* )
<i>operations</i>	⇒	<b>void</b>   <i>operation</i>   ( <i>operation</i> [, <i>operation</i> ]* )
<i>operation</i>	⇒	<b>genarray</b> ( <i>expr</i> [, <i>expr</i> ] )   <b>modarray</b> ( <i>expr</i> )   <b>fold</b> ( ( <i>qual_ext_id</i>   <i>qual_ext_op</i> ) [( <i>exprs</i> ) ] , <i>expr</i> )   <b>foldfix</b> ( ( <i>qual_ext_id</i>   <i>qual_ext_op</i> ) [( <i>exprs</i> ) ] , <i>expr</i> , <i>expr</i> )   <b>propagate</b> ( <i>id</i> )

## 11 Types

*type* ⇒ *basetype* [*shape\_spec* ]  
*shape\_spec* ⇒ [ \* ]  
| [ + ]  
| [ [ . [ , . ]\* ] ]  
| [ *nums* ]  
*basetype* ⇒ *simpletype*  
| *usertype*  
| *structtype*  
*simpletype* ⇒ **byte**  
| **short**  
| **int**  
| **long**  
| **longlong**  
| **ubyte**  
| **ushort**  
| **uint**  
| **ulong**  
| **ulonglong**  
| **float**  
| **bool**  
| **char**  
| **double**  
*structtype* ⇒ [*id* :: ] **struct** *id*  
*usertype* ⇒ [*id* :: ] *id*

## 12 Identifiers

*id\_or\_mdots* ⇒ ( id / . / ... )  
*qual\_ext\_id* ⇒ [ id :: ] *ext\_id*  
*ext\_id* ⇒ ( id / *reservedid* )  
*reservedid* ⇒ **genarray**  
| **modarray**  
| **fold**  
| **foldfix**  
| **propagate**  
| **all**  
| **except**  
*qual\_ext\_op* ⇒ [ id :: ] *ext\_op*  
*ext\_op* ⇒ ( op / *reservedop* )  
*reservedop* ⇒ &  
| &&  
| ||  
| !  
| ~  
+
\*
/
%
<=
<
>=
>
»
«
^
++
-

## 13 Constants

*const* ⇒ *numbyte*  
| *numshort*  
| *numint*  
| *numlong*  
| *numlonglong*  
| *numubyte*  
| *numushort*  
| *numuint*  
| *numulong*  
| *numulonglong*  
| *num*  
| *float*  
| *double*  
| *char*  
| [*str*]<sup>+</sup>  
| **true**  
| **false**  
*nums* ⇒ [*num* [, *num*]<sup>\*</sup> ]



## 14 Builtin Operations

```

unary_prf ⇒ ( _tob_S_ / _tos_S_ / _toi_S_ / _tol_S_ / _toll_S_ )
            | ( _toub_S_ / _tous_S_ / _toui_S_ / _toul_S_ / _toull_S_ )
            | _tof_S_
            | _tod_S_
            | _toc_S_
            | _tobool_S_
            | ( _not_S_ / _not_V_ )
            | ( _neg_S_ / _neg_V_ )
            | ( _abs_S_ / _abs_V_ )
            | _dim_A_
            | _shape_A_

ternary_prf ⇒ _modarray_AxVxS_

binary_prf ⇒ ( _add_SxS_ / _add_SxV_ / _add_VxS_ / _add_VxV_ )
            | ( _sub_SxS_ / _sub_SxV_ / _sub_VxS_ / _sub_VxV_ )
            | ( _mul_SxS_ / _mul_SxV_ / _mul_VxS_ / _mul_VxV_ )
            | ( _div_SxS_ / _div_SxV_ / _div_VxS_ / _div_VxV_ )
            | ( _mod_SxS_ / _mod_SxV_ / _mod_VxS_ / _mod_VxV_ )
            | ( _min_SxS_ / _min_SxV_ / _min_VxS_ / _min_VxV_ )
            | ( _max_SxS_ / _max_SxV_ / _max_VxS_ / _max_VxV_ )
            | ( _eq_SxS_ / _eq_SxV_ / _eq_VxS_ / _eq_VxV_ )
            | ( _neq_SxS_ / _neq_SxV_ / _neq_VxS_ / _neq_VxV_ )
            | ( _le_SxS_ / _le_SxV_ / _le_VxS_ / _le_VxV_ )
            | ( _lt_SxS_ / _lt_SxV_ / _lt_VxS_ / _lt_VxV_ )
            | ( _ge_SxS_ / _ge_SxV_ / _ge_VxS_ / _ge_VxV_ )
            | ( _gt_SxS_ / _gt_SxV_ / _gt_VxS_ / _gt_VxV_ )
            | ( _and_SxS_ / _and_SxV_ / _and_VxS_ / _and_VxV_ )
            | ( _or_SxS_ / _or_SxV_ / _or_VxS_ / _or_VxV_ )
            | _reshape_VxA_
            | _sel_VxA_
            | _take_SxV_
            | _drop_SxV_
            | _cat_VxV_

```

## 15 Pragma

```

interface_pragma⇒ # pragma linkname str
                  | # pragma header str
                  | # pragma linkwith [str]+
                  | # pragma linkobj [str]+
                  | # pragma copyfun str
                  | # pragma freefun str
                  | # pragma linksign [ nums ]
                  | # pragma refcounting [ nums ]
                  | # pragma effect qual_ext_id [, qual_ext_id ]*
withloop_pragma⇒ # pragma wloop wc_funcall
                  | # pragma nocuda
generator_pragma⇒ # pragma gpukernel GridBlock ( num , gk_funcall )
wc_funcall      ⇒ Default
                  | All ( )
                  | Cubes ( )
                  | ConstSegs ( [[ nums ] , [ nums ] , ]+ wc_funcall )
                  | NoBlocking ( wc_funcall )
                  | BvL0 ( [[ nums ] , ]+ wc_funcall )
                  | BvL1 ( [[ nums ] , ]+ wc_funcall )
                  | BvL2 ( [[ nums ] , ]+ wc_funcall )
                  | Ubv ( [[ nums ] , ]+ wc_funcall )
                  | Scheduling ( sched_param , wc_funcall )
                  | Taskset ( tsel_param , wc_funcall )
gk_funcall      ⇒ Gen
                  | ShiftLB ( gk_funcall )
                  | CompressGrid ( [ nums ] , gk_funcall )
                  | Permute ( [ nums ] , gk_funcall )
                  | FoldLast2 ( gk_funcall )
                  | SplitLast ( num , gk_funcall )
                  | PadLast ( num , gk_funcall )
cachesim_pragma⇒ # pragma cachesim [str]*
function_pragma⇒ # pragma recountdots
                  | # pragma noline

```