A Documentation of iplC

Still in works!

Mayank Jain

March 20, 2023

1 Overview

An incomplete documentation of iplC built using the ISO C standard and the provided grammar as reference.

1.1 Base Types

- int
- float
- void
- struct

1.2 Type modifiers

- T*
- T[n]

Only possible pointers are to base types and not to arrays.
The only possible method of constructing a pointer to an array is to use `&` operator on an array type.
This can create types of the form `int**(*)[2][3]` which is a pointer to an array [2][3] of int **.
note: "pointer to complete object type" means pointer that is not `void*`.
note: "pointers of compatible type" means that after degenerating first dimension of array (if array is on top) , type should match exactly. Few examples follow:

- `int**[3]` and `int***` are compatible
- `int**[3][4]` and `int***[4]` are incompatible
- `int**(*)[4]` and `int**[3][4]` are compatible
- `int[3][4]` and `int(*)[4]` are compatible

2 Operator constraints

(What operators are allowed, and on what operands):

2.1 Array sub-scripting

`A[b]`

- A is a pointer to a complete object type
- b is of type integer

2.2 Function calls

`A f(b, ...)`

- number of arguments should mach to those in the declaration.
- Passed parameters being assigned to the corresponding arguments should be allowed.

2.3 Structure member

`a.b`
`a->b`

- a should have struct type (for `.`) or pointer to struct type (for `->`) and b should name a member of that type.

2.4 Postfix increment

`a++`

- a should be a modifiable lvalue, and it should be of type int, float or a pointer (note: void* is fine).

note:the result is no longer an lvalue

2.5 Address

`&a`

- a should be an lvalue

2.6 Indirection

`*a`

- a should be a pointer to a complete type.

2.7 Unary minus

`-a`

- a should be int or float.

2.8 Negation

`!a`

- a should be of type int, float or a pointer.

2.9 Multiplicative Operators

`a*b`
`a/b`

- a and b both should be of type int or float.

2.10 Addition

`a+b`

- a and b both should be of type int or float, OR
- one of them int and other a pointer.

2.11 Subtraction

`a-b`

- a and b both should be of type int or float, OR
- both should be pointers of compatible type, OR
- a should be a pointer and b should be an integer

2.12 Relational Operators

`a<b`
`a>b`
`a<=b`
`a>=b`

- a and b both must be int or float, OR
- both must be pointers of compatible type.

2.13 Equality Operators

`a==b`
`a!=b`

- a and b both must be int or float, OR
- both must be pointers of compatible type, OR
- atleast one of them must be of type void*, OR
- or one of them is zero (nullptr) and the other is a pointer

2.14 Logical Operators

`a&&b`
`a||b`

- a and b both must be int, float or pointers.

2.15 Assignment

`a=b` a should be a modifiable lvalue, AND

- a and b are both are int or float, OR
- a and b are both same struct types, OR
- a and b are pointers to compatible types, OR
- a and b are pointers and atlease one of them is void*, OR
- a is a pointer and b is zero (nullptr).

3 Statement Contraints

(What is allowed to appear inside what statement):

3.1 Expression used as condition

In selection statement (for, while and if-else)
should be of type int, float or pointer.

3.2 return statement

Returning expression should be able to be assigned to type returned by function.

3.3 declarations

- Multiple definitions of same identifier are not allowed.
- Using an undefined function, struct or function is also disallowed.
- Functions should be allowed to be recursive.
- Structs may have pointers to their own types inside their definition.