CKPMcc Syntax

The compiler is a sub-set of the programming language C.
The compiler consist at the Moment of three major parts:

- PreProcessor
- Sanner
- Parser

User Symbols which are use to define the EBNF:
Definition ...::=
StopSymbole … .
Logic or … |
Logic and … &
Option (0-1)… […]
Option (0-n)…{…}
Gruppierung … (...)
Anführungszeichen, 1. Variante … " ... " Zeichenketten

Keywords

Used Keywords:
if, elsif, else, while, struct, sizeof, return, break, continue, char, int, void, static, extern,

Nicht benutzte Keywords:
for, short, long, float, double, signed, unsigned, auto, register, typedef, union, const, volatile, goto, switch, case, default, do,

write, open, close, read, printf

Formale Definition:

CharSet	::= { ‘0x01’ .. ‘0x09’, ‘0x0B’, ‘0x0C’, ‘0x0E’.. ‘0x21’, ‘0x23’ .. ’0xFF’ }
Identifier	::= [Letter | "_"]{Letter | Digit | "_"}.
String	::= "\"" CharSet "\"".
Constant	::= Number | "'" Letter "'" | String.
SignedConstant	::= [“+” | ”-“] Number | "'" Letter "'" | String.
Number	::= Digit {Digit}.
Letter	::= "A" ... "Z" | "a" .. "z".
Digit	::= "0" ... "9".

Expressions

Program	::= {Declaration}.
Declaration	::= DataDeclaration | FunctionDeclaration.
DataDeclaration	::= [“extern”] [“static”] (SimpleDeclaration | StructDeclaration).

MemberDeclaration ::= TypeName [Pointer] Identifier ["[" Number "]"].

SimpleDeclaration ::= MemberDeclaration [Init] ";".

TypeName ::= SimpleType | StructDeclaration

SimpeType ::= “void” | "char" | "int" | Enumeration.

Enumeration ::= BooleanEnumaration.

BooleanEnumaration ::= “enum” “{“ “false” [“=” “0”] “,”
“true” [“=” “1”] “}” “boolean".

Pointer ::= “*” { Pointer }.

Init ::= "=" SignedConstant
| "{" SignedConstant { "," SignedConstant } "}".

StructDeclaration ::= "struct" Identifier "{" MemberDeclaration ";"
{ MemberDeclaration ";" } "}” “;" .

FunctionDeclaration ::= TypeName [Pointer] Identifier "(" NameList ")"
(Block | “;”).

NameList ::= TypeName [Pointer] Identifier {"," TypeName [Pointer] Identifier}.

Block ::= "{"{ DataDeclaration }{ Statement }"}".

Statement ::= Block | AssignmentExpression ";"
| "if” “(" AssignmentExpression ")" Statement { "elsif” “(" AssignmentExpression ")" Statement } ["else" Statement]
| "while” “(" AssignmentExpression ")" Statement
| "break” “;"
| "continue” “;"
| "return” [AssignmentExpression] “;" .

AssignmentExpression ::= { [Pointer] ListValue { ("." | "->") ListValue } "=" } [TypeCastExpression] Expression ";"

TypeCastExpression ::= “(“ TypeName [Pointer] “)”.

Expression ::= LogicalAndExpression { “||” LogicalAndExpression }.

LogicalAndExpression ::= ConditionalExpression { “&&” ConditionalExpression }.

ConditionalExpression ::= SimpleExpression [("==" | "!=" | "<" | "<=" | ">=" | ">") SimpleExpression] .

SimpleExpression ::= ["+" | "-" | "!" | “&” | “*”] Term { ("+" | "-" | "|") Term}.

Term	::= Factor { ("*" | "/" | "%" | “&") Factor }.
Factor	::= "sizeof" “(“ TypeName “)” | Constant | Identifier | "(" Expression ")" | Identifier "(" [Parameter] ")" | ListValue { ("." | “->”) ListValue }].
ListValue	::= Identifier ["[" Expression "]"].
Parameter	::= Expression {"," Expression }.

Offen:

- Problematik bei Strukturen mit der Deklaration: Referenz vs. Struktur
- → wenn nur Referenzen möglich sind, wie soll dann sizeof() funktionieren?
- → etwa so: sizeof(*pointer-type) ???

Examples Syntax for the CKPMcc-Compiler
Preprozessor:

```
#include
#define
#if
#ifdef
#ifndef
#else
#eslif
#endif
```

`Library Functions:`

open(2), close(2), write(2), read(2),
snprintf(3), malloc(3), free(3), atoi(3),

Compiler:

```
# <line-number> “<file-name>”
```

`Typen:`

```
char, int, long     ... alle signed!
boolean als:   enum { false = 0, true = 1}  boolean;
```

`Strukturen:`

```
struct <name> { int v1; long v2; };

struct <name> *<variable>;
```

```
typedef struct <name> <typname>;

sizeof()
```

Arrays / Referenzen:

```
X[j] = Y[g];

char *x = "das ist ein String.";

int x = 10;
int *y = &x;
```

Type Casts:

```
char *x = …;
*(int*)x = 17;

int x = (int) ‘x’ ;
char y = (char)x ;
int z = (int) y ;

int x,y,z ;
x = y = z = 0;

int x;    x = (int) ‘x’ ;
char y;   y = (char)x ;
int z;    z = (int) y ;

char *s;  s = (char*) malloc (18);
struct lala* l;
l = (struct lala*) malloc ( sizeof (struct lala) );


extern int errno;
static int geheim1;
static int geheim2 = 3;
int fuer_alle1;
int fuer_alle2 = 18;

void funktion1 ();
void funktion2 (int a);
void funktion2 (int a, long b, char* c);


int main (int argc, char** argv) {
   printf(“Hallo Welt!\n“);
    return 0;
}
```

Kontrollstrukturen:

```
int i=0;
while (i < 18) {
   printf(“Zahl ist %d\n”,i);
   if (errno == EPERM) continue;
   if (errno == 0) break;
   i++;
}

if (lala == 9) { machwas(1); } elseif (lala == 10) {
machauchwas(8); } else { nochwas(); }


if (errno != 0) {        /* ‚{‘ und ‚}’ müssen immer
geschrieben werden! */
     return 8;
}

if ( (lala = 19) > 0) {       /* →Zuweisungen in
Kontrollstrukturen */
                         /* !!! NICHT IMPLEMENTIERN → geht
                    aber !!! */
   machwas („....“);
}

x = y * ( a + 7);


struct meins { int a; int b; };
meins v1;                /* → geht zur Zeit noch. */
meins *v2;

v1.a = 8;                      /* → geht zur Zeit noch. */
v1.b = 9;                      /* → geht zur Zeit noch. */

v2 = (struct meins*)malloc(sizeof(struct meins));
v2->a = 1;
v2->b = 3;


Operatoren:

+ - * / %
++ --
&& ||
& | !
<< >>
== != > >= < <=
=
Nicht:    ^ ~ += -= *= /= %= <<= >>= ^= &= |=
```