

<input type="checkbox"/> Gr. 1, Dr. H. Dobler	Name _____	Aufwand in h _____
<input type="checkbox"/> Gr. 2, Dr. G. Kronberger		
<input type="checkbox"/> Gr. 3, Dr. H. Dobler	Punkte _____	Übungsleiter _____

Im Moodle-Kurs zu dieser Lehrveranstaltung finden Sie in der Datei *Coco-2.zip* die .NET-Version von Coco-2 (in und für C#).

1. MiniC: Scanner und Parser mit Coco-2

(8 Punkte)

MiniC kennen Sie ja schon. Zur Wiederholung: Unten links ist ein einfaches Programm zur Berechnung des Satzes von Pythagoras dargestellt, rechts die Grammatik von MiniC (die Sie auch im Moodle-Kurs in der Datei *MiniC.syn* finden):

```
void main() {
    int a, b, cs;
    scanf(a);
    scanf(b);
    cs = (a * a) + (b * b);
    printf(cs);
}
```

```
MC =      "void" "main" "(" ")" "{"
        [ VarDecl ]
        StatSeq
        "}" .
VarDecl = "int" ident { "," ident } ";" .
StatSeq = Stat { Stat } .
Stat =    [ ident "=" Expr
          | "scanf" "(" ident ")"
          | "printf" "(" Expr ")"
          ] ";" .
Expr =    Term { ( "+" | "-" ) Term } .
Term =    Fact { ( "*" | "/" ) Fact } .
Fact =    ident | number | "(" Expr ")" .
```

Erzeugen Sie mit Coco-2 einen lexikalischen Analysator (*scanner*) und einen Syntaxanalysator (*parser*) für MiniC und bauen Sie daraus ein Programm für die Analyse von MiniC-Programmen.

2. MiniCpp: Scanner, Parser und ... mit Coco-2

(16 Punkte)

MiniCpp kennen Sie zwar auch schon, hier aber trotzdem das Beispiel zur Wiederholung:

```
void Sieve(int n); // declaration

void main() {
    int n;
    cout << "n > ";
    cin >> n;
    if (n > 2)
        Sieve(n);
} // main

void Sieve(int n) { // definition
    int col, i, j;
    bool *sieve = 0;
    sieve = new bool[n + 1];
    i = 2;
    while (i <= n) {
        sieve[i] = true;
        i++;
    } // while
```

```
cout << 2 << " ";
col = 1;
i = 3;
while (i <= n) {
    if (sieve[i]) {
        if (col == 10) {
            cout << endl;
            col = 0;
        } // if
        col++;
        cout << i << " ";
        j = i * i;
        while (j <= n) {
            sieve[j] = false;
            j = j + 2 * i;
        } // while
    } // if
    i = i + 2;
} // while
delete[] sieve;
} // Sieve
```

Hier die Grammatik für MiniCpp, die Sie auch im Moodle-Kurs in der Datei *MiniCpp.syn* finden:

```

MiniCpp =      { ConstDecl | VarDef | FuncDecl | FuncDef } .
ConstDecl =    'const' Type ident Init ';' .
Init =         '=' ( false | true | number ) .
VarDef =       Type [ '*' ] ident [ Init ]
               { ',' [ '*' ] ident [ Init ] } ';' .
FuncDecl =     FuncHead ';' .
FuncDef =      FuncHead Block .
FuncHead =     Type [ '*' ] ident '(' [ FormParList ] ')' .
FormParList =  ( 'void' |
                 Type [ '*' ] ident [ '[' ']' ]
                 { ',' Type [ '*' ] ident [ '[' ']' ] } ) .
Type =         'void' | 'bool' | 'int' .
Block =        '{' { ConstDecl | VarDef | Stat } '}' .
Stat =         ( IncStat | DecStat | AssignStat
                 | CallStat | IfStat
                 | WhileStat | BreakStat
                 | InputStat | OutputStat | DeleteStat | ReturnStat
                 | Block
                 | ';'
                 ) .
IncStat =      ident '++' ';' .
DecStat =      ident '--' ';' .
AssignStat =   ident [ '[' Expr ']' ] '=' Expr ';' .
CallStat =     ident '(' [ ActParList ] ')' ';' .
ActParList =   Expr { ',' Expr } .
IfStat =       'if' '(' Expr ')' Stat [ 'else' Stat ] .
WhileStat =    'while' '(' Expr ')' Stat .
BreakStat =    'break' ';' .
InputStat =    'cin' '>>' ident ';' .
OutputStat =   'cout' '<<' ( Expr | string | 'endl' )
               { '<<' ( Expr | string | 'endl' ) } ';' .
DeleteStat =   'delete' '[' ']' ident ';' .
ReturnStat =   'return' [ Expr ] ';' .
Expr =         OrExpr .
OrExpr =       AndExpr { '|' AndExpr } .
AndExpr =      RelExpr { '&&' RelExpr } .
RelExpr =      SimpleExpr
               [ ( '==' | '!=' | '<' | '<=' | '>' | '>=' )
                 SimpleExpr ] .
SimpleExpr =   [ '+' | '-' ]
               Term { ( '+' | '-' ) Term } .
Term =         NotFact { ( '*' | '/' | '%' ) NotFact } .
NotFact =      [ '!' ] Fact .
Fact =         'false' | 'true'
               | number
               | ident [ ( '[' Expr ']' )
                         | ( '(' [ ActParList ] ')' )
                         ]
               | 'new' Type '[' Expr ']'
               | '(' Expr ')' .

```

Erzeugen Sie mit Coco-2 einen lexikalischen Analysator (*scanner*) und einen Syntaxanalysator (*parser*) für MiniCpp und ... bauen Sie daraus ein Programm für die Analyse von MiniCpp-Programmen. Von diesem Werkzeug für die *statische Programmanalyse* sollen mindestens

- die Anzahl der Zeilen (*lines of code*, LOC),
- die Anzahl der Anweisungen (*statements*) und
- die Strukturkomplexität V nach McCabe ($V = 1 + \text{Anzahl der binären Verzweigungen}$)

berechnet werden. Natürlich wäre auch EV (die essentielle Strukturkomplexität = V nach Reduktion der D-Diagrammanteile) von Interesse, vielleicht schaffen Sie das ja auch noch.

3. Für besonders Interessierte, deshalb ...

(+ 4 Bonuspunkte)

M. H. Halstead hat Metriken vorgeschlagen, die nicht nur die Größe eines Algorithmus (in Form seiner "Länge" N , s. u.) und seine Struktur (wie McCabe) in Betracht ziehen, sondern vor allem die Umfang und Komplexität der darin durchgeführten Berechnungen berücksichtigen. Dafür müssen die vier in Tab. 0.1 definierten Werte $n1$, $N1$, $n2$ und $N2$ aus dem Quelltext eines Algorithmus ermittelt werden.

	Anzahl unterschiedlicher ...	Gesamtanzahl der verwendeten ...
... Operatoren	$n1$	$N1$
... Operanden	$n2$	$N2$

Tab. 0.1: Werte zur Berechnung der Halstead-Metriken

Dabei gelten als Operatoren nicht nur die mathematischen Operatoren (z. B. die arithmetischen $+$, $-$, $*$, $/$ und die relationalen $=$, $<$, $>$, \dots), sondern auch Symbole (z. B. $=$) sowie jene Schlüsselwörter (z. B. *if*, *else* und *while*), die Aktionen in einem Algorithmus auslösen. Als Operanden werden alle Elemente eines Algorithmus gewertet, die Daten repräsentieren (vor allem Literale, Konstanten, Variablen, aber auch Sprungmarken). Aus den vier Größen in Tab. 0.1 können die in Tab. 0.2 genannten fünf Halstead-Metriken berechnet werden.

Halstead-Metrik	Bezeichnung	Formel
Alphabet (engl. <i>vocabulary</i>)	n	$n = n1 + n2$
Länge (engl. <i>length</i>)	N	$N = N1 + N2$
Volumen (engl. <i>volume</i>)	V	$V = N \cdot \log_2(n)$
Schwierigkeit (engl. <i>difficulty</i>)	D	$D = (n1 \cdot N2) / (2 \cdot n2)$
Aufwand (engl. <i>effort</i>)	E	$E = D \cdot V$

Tab. 0.2: Halstead-Metriken und ihre Berechnung

Erweitern Sie Ihr Werkzeug zur statischen Analyse von MiniCpp-Programmen aus 2. um die Berechnung der oben erläuterten Halstead-Metriken.