SIMPLESEM (Operationale Semantik)

Einfacher Assembler zur Spezifikation der Semantik

Statische Struktur: Befehlszeiger (instruction pointer, ip), Befehlsspeicher (code memory, C), Datenspeicher (D)

Befehle: set Ziel, Quelle
set 10, D[20]
set 15, read
set write, D[50]
set 99, D[15]+D[33∗C[41]]−8
set D[10], D[20]
jump 47
jumpt 47, D[3]>D[8]
jump D[13]

SIMPLESEM und Sprachen

Statische Sprachen:
Speicher vor Ausführung allokiert
Beispiele: Frühe Versionen von FORTRAN und COBOL
Wir beschreiben die Sprachen C1, C2 und C2′

Stack-basierte Sprachen:
Speicher zur Laufzeit von einem Stack (LIFO) allokiert
Beispiele: ALGOL 60, . . .
Wir beschreiben die Sprachen C3 und C4

Dynamische Sprachen:
Speicher zur Laufzeit von einem Heap allokiert
Beispiele: die meisten heutigen Sprachen
Wir beschreiben die Sprache C5

C1: Sprache mit einfachen Statements

```
main()
{
   int i, j;
   get(i, j);
   while (i != j)
      if (i > j)
         i -= j;
      else
         j -= i;
   print(i);
}
```


C2: Einfache Routinen

```
int i=1, j=2, k=3;

alpha()
{
   int i=4, l=5;
   ...
   i += k + l;
   ...
};

beta()
{
   int k=6;
   ...
   i = j + k;
   alpha();
   ...
};

main()
{
   ...
   beta();
   ...
}
```

C2': Getrennte Übersetzung

```
/* file 1 */

int i=1, j=2, k=3;
extern beta();
main()
{
   ...
   beta();
   ...
}
alpha()
```

```
/* file 2 */

extern int k;
alpha()
{
   int i=4, l=5;
   ...
   i += k + l;
   ...
}
beta()
```

```
/* file 3 */

extern int i,j;
extern alpha();
beta()
{
   int k = 6;
   ...
   i = j + k;
   alpha();
   ...
}
```


C3: Rekursion und Ergebnis-Werte

```
int n;
int fact()
{
   int loc;
   if (n > 1) {
      loc = n--;
      return loc * fact();
   }
   else
      return 1;
}
```

```
main()
{
   get(n);
   if (n >= 0)
      print(fact());
   else
      print("input error");
}
```


Aufruf einer Routine

set 1, D[1]+1	allokiere Platz für Ergebnis
set D[1], ip+4	setze "return pointer"
set D[1]+1, D[0]	setze "dynamic link"
set 0, D[1]	setze "CURRENT"
set 1, D[1]+AR	setze "FREE"
jump addr	springe zur Routine

Rückkehr von einem Aufruf

set 1, D[0]	setze "FREE"
set 0, D[D[1]+1]	setze "CURRENT"
jump D[D[1]]	springe an die Rückkehradresse

Static Link

Jede Referenz ist statisch an ein Paar $\langle d, o \rangle$ gebunden;
d ist die Distanz, o der Offset

Berechnung des *frame pointers*:

$$\mathrm{fp}(d) \;=\; \text{if } d{=}0 \text{ then } D[0] \text{ else } D[\mathrm{fp}(d-1)+2]$$

Beispiele: $\mathrm{fp}(0) = D[0]$,
$\mathrm{fp}(1) = D[D[0]+2]$,
$\mathrm{fp}(2) = D[D[D[0]+2]+2]$

Die Adresse einer Variable gebunden an $\langle d, o \rangle$ ist $D[\mathrm{fp}(d)+o]$

Aufruf einer Routine in C4”

set 1, D[1]+1	allokiere Platz für Ergebnis
set D[1], ip+5	setze “return pointer”
set D[1]+1, D[0]	setze “dynamic link”
set D[1]+2, fp(d)	setze “static link”
set 0, D[1]	setze “CURRENT”
set 1, D[1]+AR	setze “FREE”
jump addr	springe zur Routine

C5’: Dynamische Arrays

Beispiel: **type** VECTOR **is array** (INTEGER **range** <>);
A: VECTOR (1 .. N); -- N nicht statisch bekannt

Zur Übersetzungszeit wird im activation record Platz für einen Descriptor reserviert. Dieser enthält einen Zeiger auf den Speicher für das Array sowie je eine Zelle für die Indexgrenzen.

Bei Ausführung der Deklaration eines dynamischen Arrays wird der activation record um Platz für das Array erweitert. Der Zeiger im Descriptor wird entsprechend gesetzt.

Beispiel für Zugriff A(I) := 0;

set[D[D[0]+m] + D[D[0]+s]], 0

(m = Offset des Descriptors von A; s = Offset von I)

Parameterübergabekonventionen

Call by reference: l-Wert des aktuellen Param. übergeben

Call by copy: Parameter sind lokale Variablen

- **Call by value:** übergebener r-Wert in Speicherbereich der aufgerufenen Routine kopiert
- **Call by result:** Rückgabewert in l-Wert einer Variablen des Aufrufers kopiert
- **Call by value-result:** r-Wert einer Variablen des Aufrufers in Speicherbereich der Routine kopiert; am Ende Inhalt des Speicherbereichs in l-Wert der Variablen kopiert

Call by name: In aufgerufener Routine jedes Vorkommen des formalen durch aktuellen Parameter ersetzt

Implementierung von Call by Reference

Activation record der aufgerufenen Routine um eine Zelle pro Parameter erweitert (Offset off)

Übergabe einer Variablen $\langle d, o\rangle$:

set D[0]+off, fp(d)+o

Übergabe eines call-by-reference-Parameters $\langle d, o\rangle$:

set D[0]+off, D[fp(d)+o]

Zuweisung an call-by-reference-Parameter x = 0;

set D[D[0]+off], 0

Call by Reference / Value-Result

Diese Konventionen unterscheiden sich wenn

- zwei formale Parameter aliases sind;

  Beispiel: aktuelle Parameter: a[i] und a[j]
  formale Parameter: x und y
  Rumpf der Routine: $x = 0;\ \ y{+}{+};$

- ein formaler Parameter und eine Variable, die sowohl für Aufrufer als auch Aufgerufenen sichtbar ist, aliases sind.

  Beispiel: aktueller Parameter: a (Variable)
  formaler Parameter: x
  Rumpf der Routine: $a = 1;\ \ x = x + a;$

Call by Name

```
                          |   int c; /* globale Variable */
swap(int a, b) {          |   swap(int a, b) {
   int temp = a;          |      int temp = a; a = b;
   a = b;                 |      b = temp; c++;
   b = temp;              |   };
};                        |   y() {
   ...                    |     int c, d;
   swap(i, a[i]);         |     swap(c, d);
   ...                    |   };

temp = i;                 |   temp = c; c = d;
i = a[i];                 |   d = temp;
a[i] = temp;              |   c++;              !!
```


Routinen als Parameter

```
int u, v;
a()
{
   int y;
   ...
};
```

```
b(routine x)
{
   int u, v, y;
   c() {
      ...
      y = ...;
      ...
   };
   x();
   b(c);
   ...
}
```

```
main()
{
   b(a);
};
```