

InfoEducație 2010

Aer

Un microlimbaj de programare
© 2010, Vlad Dumitru.

Atenție! Acest document nu este în faza finală!
Aștept sugestii pe deveah@gmail.com

Prezentare

Aer este un limbaj de programare minimal, construit cu scopul de a studia modul în care funcționează compilatoarele. Având în minte acest lucru, limbajul a fost proiectat pentru a fi posibil să se scrie un compilator în el, astfel dovedind potența limbajului.

Limbajul este bazat pe o stivă, fiind principala structură de date disponibilă. Există și un suport pentru variabile, în caz că nu este de ajuns stiva.

Limbajul nu aduce nimic nou din punct de vedere sintactic – din acest punct de vedere se poate așeza limbajul în familia FORTH¹.

Sintaxa este liberă – un program este doar o înșiruire de funcții. De exemplu, programul „Salut, lume!” arată cam așa:

```
:main "Salut, lume!\n" @printf $0 @exit
```

Putem acum să luăm pe rând exemplul și să detaliăm ce face fiecare funcție:

<code>:main</code>	definește punctul de intrare (main pe Linux)
<code>"Salut, lume!\n"</code>	definește un șir de caractere și adaugă pointerul acestuia în stivă
<code>@printf</code>	execută funcția <code>printf</code> , care afișează șirul de caractere adăugat mai devreme în stivă
<code>\$0</code>	adaugă în stivă numărul întreg 0
<code>@exit</code>	apelează funcția <code>exit</code> , care termină imediat execuția programului.

¹ Din Wikipedia: „FORTH este un limbaj de programare structurat, imperativ, procedural și bazat pe o stivă.”

Prin `{ ... }` se intră în modul assembly - tot ce există înăuntrul blocului este copiat direct în fișierul compilat. De exemplu, putem incrementa ultimul element adăugat în stivă astfel:

```
{popl %eax; incl %eax; pushl %eax}
```

Prin `(...)` se scrie un comentariu - peste acesta se sare la timpul compilării.

Prin `[...]` se definește un bloc executat condiționat - tot ce este înăuntrul acestuia se execută numai dacă s-a executat înainte o operație de testare a adevărului, iar rezultatul acesteia a fost adevărat. Un simplu exemplu:

```
:main $1 $1 = (dacă 1=1) [ "Da, 1=1.\n" @printf ] $0 @exit
```

Operații de testare

Operațiile de testare iau primele două elemente din stivă și le compară:

- `=` verifică dacă sunt egale
- `>` verifică dacă primul este mai mare
- `<` verifică dacă primul este mai mic

Dacă elementele verifică relația, blocurile executate condiționat se vor executa normal.

Comanda `!` inversează rezultatul operației precedente. De exemplu, `$1 $1 = !` va fi fals.

Controlul fluxului

Pentru acesta, **Aer** suportă etichete:

- `: etichetă` definește o etichetă
- `; etichetă sare` la acea etichetă

O importantă calitate a limbajului este aceea că suportă funcțiile **libc** din sistemul de operare - asta înseamnă că poate apela oricând la orice funcție din librăria de funcții C, unde e disponibilă (Linux fiind scris în C, nu există nicio dificultate în folosirea libc).

De exemplu, apelăm la `getchar` în felul următor: `@getchar`.

Următorul exemplu repetă intrările utilizatorului, și iese când utilizatorul a introdus litera `q` (ASCII 113):

```
:main
    .ceva
:11
    @getchar ^ceva
    ,ceva $113 =
    [ $0 @exit ]
    ,ceva @putchar
;11
    $0 @exit
```

Folosind etichete, am creat un bloc repetitiv din care se iese cu o anumită condiție (aici dacă utilizatorul a introdus caracterul corespunzător).

Variabile

Aer suportă variabile, pentru a ușura munca programatorului:

- `.variabilă` definește o variabilă
- `,variabilă` adaugă acea variabilă în stivă
- `^variabilă` mută valoarea acumulatorului în acea variabilă.

Funcții pentru modificarea variabilelor:

- `+variabilă` incrementează variabila
- `-variabilă` decrementează variabila

De asemenea, există și funcția `~număr`, care atribuie acumulatorului acea valoare.

Un scurt exemplu:

```
:main
    .v1
    ~1 ^v1 ( atribuim lui v1 valoarea 1 )
    ,v1 "%i\n" @printf

    +v1 ,v1 "%i\n" @printf
    -v1 ,v1 "%i\n" @printf

    ( verificăm dacă v1 = 1 )
    ,v1 $1 =
    [ "Da, v1 = 1.\n" @printf ]

    $0 @exit
```

Rezultatul va fi:

```
1
2
1
Da, v1 = 1.
```

Compilatorul

Compilatorul (`ac - aer compiler`) are următoarea sintaxă:

```
ac [intrare] [ieșire]
```

- Prin [intrare] se înțelege fișierul care conține un program în limbajul aer.
- Prin [ieșire] se înțelege un nume de fișier în care să se stocheze codul sursă compilat în limbaj assembly.

De exemplu, dacă vrem să compilăm fișierul `test.txt`, pașii sunt următorii:

```
$ ac text.txt test.s
```

```
$ gcc -x assembler test.s -o test
```

De ce self-hosting?

Self-hosting (pe românește „care se poate compila singur”) pentru că limbajul este destul de puternic să creeze compilatorul pentru sine, un compilator care produce cu exactitate același fișier precum compilatorul original, scris în ANSI C.

Ca să dovedim acest lucru:

1. Construim compilatorul original:

```
$ gcc ac.c -o ac
```

2. Construim compilatorul scris în Aer:

```
$ ./ac a2.txt a2.s
```

```
$ gcc -x assembler a2.s -o a2
```

3. Compilatorul scris în Aer citește fișierul sursă din `in.txt`, așa că acum copiem fișierul `a2.txt`, compilatorul scris în [Aer](#), în `in.txt`:

```
$ cp a2.txt in.txt
```

4. Executăm `a2`

```
$ ./a2
```

5. `diff` este un program care afișează diferențele dintre două fișiere. Acum verificăm diferențele dintre `a2.s`, compilat de programul inițial, scris în ANSI C, și `out.s`, compilat de programul scris în [Aer](#):

```
$ diff out.s a2.s
```

```
$
```

Nu există nicio diferență, asta dovedind că am produs același compilator atât în ANSI C, cât și în [Aer](#). De aici self-hosting.

Mulțumesc colegilor pentru susținerea puternică.
Mihai Viteazul e cel mai tare.

Aici se încheie prezentarea proiectului ce sper să-mi aducă un premiu,
(dacă nu mai multe ;) – [Aer, un limbaj de programare și compilatorul său self-hosting](#).

©2010, Vlad Dumitru.

Poate-poate scos sub licența GNU GPL.