Pilt: Ken Thompson, Dennis Ritchie ja Bill Clinton USA teadusmedeli kaelariputamisel.

C on üldotstarbeline, protseduurorienteeritud programmeerimiskeel ─ kui peame silmas programmeerimiskeelte klassifikatsiooni. Keele peamine autor on *Dennis Ritchie* ja *C*-keele loomine on orgaaniliselt seotud nii *PDP-11* arhitektuuri kui ka *UNIX*-operatsioonisüsteemiga. Keel konstrueeriti puhtalt pragmaatilistel eesmärkidel: kanda üle *PDP*-7[1] jaoks assembleris programmeeritud operatsioonisüsteem *UNIX* (autor Ken Thompson) üle *PDP*-7-ga ühildamatule *PDP-11*-le. Jätkem meelde: *C* tehti programmeerijate (*Ritchie* ja *Thompson*) poolt nende endi ülesande võimalikult lihtsamaks lahendamiseks (nagu muide ka *FORTRAN* ja *FORTH*). Tulemus meeldis professionaalidele ning *C* on tänapäeval konkurentsitult eelistatavaim süsteemide programmeerimise keel[2] ning üks kasutatavaimatest rakendus-

[1] *PDP*-7 oli 18-bitine masin, mis oli ühildatav *PDP*-4 j a *PDP*-9-ga (valmis 1965), ent mitte *PDP*-11ga. Ken Thompson kirjutas *PDP*-7-le opsüsteemi „*Unics*" (nimi oli inspireeritud mängu „*Space Travel*" toetava opsüsteemi nimest „*Multics*", mis pakkus ohtralt võimalusi graafika programmeerimiseks), ent peagi muutis ta oma süsteemi nime: *UNIX*. *PDP*-7 pilt on pärit Oslo arvutimuuseumi saidilt.

C-keel sai oma nime *Thompson*i keele *B* järgi, mida loodeti ülekandeks kasutada, ent kuivõrd see ei suutnud kasutada *PDP*-11 uusi võimalusi, siis tehti sellest uus versioon. Nimi *B* pärineb omakorda eeskujuks olnud keele *BCPL* nime esitähest

[2] Suur osa *Windows*ist on kirjutatud just tavalises *C*-s. Süsteemiga kaasnevaid rakenduspakette on programmeeritud ka muudes keeltes, näiteks *Internet Explorer* on kirjutatud keeles *C*++.

programmeerimise keeltest. Samas, *C* pole kunagi meeldinud akadeemilistele ringkondadele[3] ning pole palju selliseid aktsepteeritavaid kõrgkoole, kus seda õpetataks (vt. [wiki/C]).

Joonis KjaT. Ken Thompson ja Dennis M Ritchie.

.

C loodi 1969. aastal ning arendustöö kestis 1972. aastani. *UNIX*i tuum *PDP-11*-le kirjutati esialgu assembleris, hiljem aga asendati ka see *C*-keeles programmeerituga (aastal 1973).

PDP-7.

[3] *N. Wirth* ütles ([Wirth 30]), et „Tänapäeval kasutavad programmeerijad üldjuhul keelt C…C esindab väga madalat abstraktsioonitaset. See keel väldib neid andmestruktuure, staatilisi andmetüüpe ja staatiliselt kontrollitavate liidestega moodulite fundamentaalseid mõisteid, mis annavad keelele matemaatilise selguse. .. Tegelikult esitab C assembleri koodi, mis on peidetud ilmetu süntaksi varju ning mis on täis tipitud igasugu salamärke.”

Näiteprogramm

Esitame allpool ka C-keelse versiooni „*Wilson*i näitest": sisestada *n* (*n<100*) ja *n*-elemendiline reaalarvude vektor, leida elementide aritmeetiline keskmine ning trükkida keskväärtust ületavate elementide arv (vt [wils, lk. 51]).

```
main(){
     float a[100],mean,sum;
     int n,i,number;
     scanf(„%d",&n);
     for(i=0;i<n;i++) scanf(„%f",&a[i]);
     sum=0.0;
     for(i=0;i<n;i++) sum=sum+a[i]:
     mean=sum/n;
     number=0;
     for(i=0;i<n;i++){
          if(a[i]>mean) number++;
          }
     printf(„MEAN= %f\n",mean);
     printf(„NUMBER OVER MEAN= %d\n",number);
     }
```

Andmed.

Elementaartüübid.

C-keeles on neli baas-andmetüüpi:

char ─ üks bait (tavaliselt ühe sümboli hoidmiseks);

int ─ *integer*-tüüpi arv;

float ─ ujupunktarv;

double ─ topeltpikkusega ujupunktarv.

Arvuvälja pikkus sõltub protsessorist ning need pikkused on fikseeritud päisefailis *limits.h* (näiteks, 16-bitisel masinal on konstandi *INT_MIN* väärtus −32767 ja *UINT_MAX* 65535).

Integer-tüüpi võib kitsendada sõnadega *short, long* või *unsigned.* „Need kolm tüüpi ei pea aga igas realisatsioonis tingimata esitama erineva pikkusega täisarve ─ pikkused sõltuvad konkreetsest arvutist. Üldnõue on siiski, et *short* ei oleks kunagi pikem kui *int* ja *int* pikem kui *long."* [Kelder, 61]

Char-tüüpi saab edukalt kasutada ka *int*-tüüpi väikeste arvude jaoks, näiteks määrangut
char n;
võime interpreteerida kui arvu *n* kirjeldust, kusjuures $127 \geq n \geq -128$. Kirjutades aga
unsigned char n;
on selle muutuja võimalikud väärtused vahemikus 0..255.

Lisaks on keeles *viidatüüp*, mille deklareerimiseks pole omaette võtmesõna, ent on vahendid viitade tekitamiseks ja kasutamiseks (sümbolite * ja & abil).

Konstandid (vt. KR, lk. 37 jj) võivad esineda deklareerimata kujul literaalidena aritmeetilistes avaldistes ja olla deklareeritud makroga *#define,* näiteks
#define HTL 79 (*int*-konstant) või
#define pii 3.14 (*float*-konstant: väärtus sisaldab kümnendpunkti).

Silmas pidades C protsessori-taseme võimalusi pole üllatav, et saame defineerida nii 10-ndsüsteemi (vaikimisi), kaheksandsüsteemi (´\oiii') kui ka kuueteistkümnendsüsteemi (´\xiii') ühebaidiseid konstante. Nii näiteks järgmiselt defineeritud konstandi
#define BELL ´\007' saatmine konsoolile annab lühikese vile, võrdväärne definitsioon on *#define BELL ´*\x7'.

„Sisse ehitatud" on üle kümne 1-baidise konstandi, näiteks
\n ─ reavahetus;
\t ─ horisontaalne tabulatsioon või
\? ─ küsimärk.

Agregeeritud tüübid.

Keele tasemel on kasutatavad staatilised *vektor* (so, ühemõõtmeline *massiiv*) ja mitmemõõtmeline *massiiv*, näiteks:
char a[22]; kirjeldab 22-baidise vektori (juurdepääs näit. *a[i]*) ja
int B[4][10][2]; kirjeldab kolmemõõtmelise massiivi (juurdepääs näit. *B[i][j][1]*).

Dünaamilisele vektorile ja massiivile tuleb mälu küsida programselt. Selline vektor on just samamoodi indekseeritav kui staatilinegi, ent kahe- ja enamamõõtmelise massiivi indekseerimist tuleb programmeerijal endal modelleerida. Osas **dynma** esitame näite ühest võimalikust variandist selle töö tegemiseks.

C toetab andmestruktuurist *tabel* tuntud *kirjet*[4] tüübiga *struktuur* (*structure*). Tabeli saame, kui defineerime *struktuur*-tüüpi massiivi.

Alltoodud näide illustreerib nii struktuuri, viidatüüpi kui ka rekursiooni: esitame algoritmi kahendpuu läbimiseks lõppjärjekorras (*postorder*), kusjuures „tegevus" seisneb tipu märgendi väljatrükis.

```
struct tipp{
     char label;
     struct tipp *vasak; //viit vasakule alluvale või tühi
     struct tipp *parem; //viit paremale alluvale või tühi
     };
void post(struct tipp *t){
     if(t->vasak != struct tipp *NULL) post(t->vasak);
     if(t->parem != struct tipp *NULL) post(t->parem);
     printf(„%c\n",t–>label);
     }
```

[4] Tabel on kirjetest koosnev andmestruktuur, kusjuures tabeli kirje koosneb loogilisel tasemel *võtmest* ja sellega seotud infost.

Teine andmetüüpide agregeerimise vahend on *ühend* (*union*), selle näite leiame osas „C ja protsessori-tase”.

Tehted.

Omistamine.

„Tavaline” omistamismärk on nagu algebras või *FORTRAN*is „=”, ent lisandub palju originaalset:

a Θ= b: *Algoli* notatsioonis oleks see *a:=aΘb;* kus *Θ* on kas +, −, *, /, %, ^,| , << või >>. Näiteks, *a*=b* tähendab *a:=a*b;* ja *a*=y+1,* tähendab *a=a*(y+1);*
Jagamine „/” annab *int*-tüübi korral resultaadiks jagatise täisosa ning „%” ─ jäägi (viimast tehet ei saa kasutada ujupunktarvude puhul). Tehe „|” tähistab loogilist bitikaupa liitmist (*OR*) ning „^” ─ loogilist bitikaupa välistavat liitmist (XOR). Märgid „<<” ja „>>” tähistavad vastavalt nihutamist vasakule ja paremale.

„Varjatud” omistamine seondub *PDP*-11 *increment*- ja *decrement*-režiimidega. Nii on avaldis *i++* samaväärne avaldisega *i=i+1* (või *i+=1*), ja *j −−* on sama, mis *j=j−1*. Nood režiimid on kasutatavad ka kujul *++i* ja *−−i*. Kernighan ja Ritchie toovad järgmise näite [KR, 46][5]:
Kui *n*=5, siis
x=n++; on samaväärne *x=n; n=n+1;* aga
x=++n; on samaväärne *n=n+1; x=n;*

Increment-režiimi näiteks esitame stringide[6] kopeerimise funktsiooni koodi [KR, 105]. Operandideks olevad stringid on esitatud viidatüübi abil.

```
/* strcpy: copy t to s; array subscript version */
void strcpy(char *s, char *t){
     while((*s++=*t++)!='\0');
     }
```

„Tingimuslik” omistamine: *z=(a>b) ? a : b;* Muutujale *z* omistatakse *a* väärtus, kui *a>b*, muidu aga *z=b*.

Aritmeetika.

C-keeles on 5 binaarset operaatorit: +, −, *, / ja %. Neli esimest on rakendatavad kõigi samatüübiliste operandide vahel, viimane aga ainult *int*-tüüpi operandidega. Nimelt „/” annab tulemuseks kahe *int*-tüüpi operandi jagatise täisosa ning „%” ─ jäägi. Näiteks:

a=5/2; annab resultaadiks 2 ning a=5%2; annab resultaadiks 1.

Aritmeetilistes avaldistes on prioriteetsemad korrutamine ja jagamine; tehete sooritamise järjekorda saab muuta sulgude abil.

---

[5] Asi on tegelikult keerulisem. Näiteks. kui a=3, siis avaldise 3+(++a) väärtus on 7, aga 3+a++ väärtus on 6, seejuures pärast avaldise väärtuse arvutamist mõlemal juhul a=4.
[6] *C* string on *char*-tüüpi vektor, kus lõputunnuseks on bait ‘\0’.

Võrdlustehted ja loogika.

C võrdlustehted on järgmised:
== : võrdub,
!= : ei võrdu,
> : suurem,
>= : suurem või võrdne,
< : väiksem,
<= : väiksem või võrdne.
Nende tehete prioriteet on madalam kui aritmeetikatehetel, näiteks [KR, 41]
i<lim−1; täidetakse nii, nagu oleks kirjutatud *i<(lim−1);*

Loogilistes tingimustes kasutatakse konjunktsiooni-märgina „&&" ja disjunktsiooni-märgina „||". Avaldistes kontrollitakse loogilisi tingimusi vasakult paremale ning see tegevus katkestatakse esimese *väära* tõeväärtuse tuvastamisel. Näiteks [KR,41] funktsioonis *get-line* on tsüklioperaator

```
for(i=0; i<lim-1 && (c=getchar()) != '\n' && c!=EOF; ++i)
     s[i]=c;
```

Selle operaatori täitmine katkestatakse kohe, kui *i ==lim−1*.

Bitiviisilised operaatorid.

Nende tehete operandideks võivad olla ainult *int*-tüüpi muutujad (so, *char, short, int* ja *long* ning nende *unsigned*-variandid) ja operaatorid on:
& : loogiline „ja" (*AND*), 110 & 011 = 010,
| : loogiline „või" (*OR*)[7], 110 | 011 = 111,
^ : loogiline mittesamaväärtustamine (*XOR*), 110 ^ 011 = 101,
<< : nihutamine vasakule, 011 << 1 = 110,
>> : nihutamine paremale, 011 >> 1 = 001.

Muud operaatorid.

Tingimuslik operaator on täiskujul *if(avaldis) operaator else operaator;* ja lühendatud kujul *if(avaldis) operaator;* Seejuures konstruktsioonis *else operaator* võib see *operaator* olla uus tingimuslik operaator.

Lüliti on kujul

```
     switch (avaldis){
          case konstantavaldis: operaator
          case konstantavaldis: operaator
          ...
          default: operaator
          }
```

Tavaliselt järgneb *case*-operaatorile operaatorsulg *endcase,* mis annab juhtimise lüliti tegevuspiirkonnast välja, ent on juhtumeid, kus on otstarbekas täita ka algvalikule järgnev operaator — sel juhul *endcase'*i ei kirjutata. Märkusena mainigem, et „avaldis" on ka muutuja

[7] Kui eksikombel kirjutada loogilises tingimuses && asemel & (või || asemel |), siis tulemus sõltub kompilaatorist. Mõned "annavad andeks", mõned mitte.

nimi ning „konstantavaldis" ─ konstant. „*Default*" on sundvalik: tegevus, mis käivitub, kui „avaldise" väärtus ei võrdu ühegi „konstantavaldise" väärtusega.

Tsüklioperaatoritest on kasutatavad nii **for**- kui ka **while**-variandid. Neist esimese kuju on *for(avaldis1;avaldis2;avaldis3) operaator,* mis vastab „klassikalisele" tsüklile: 1. avaldis alg-väärtustab tsükliindeksi, teine määrab lõputingimuse ning kolmas tsükli sammu[8]. Teise variandi kuju on *while (avaldis) operaator;* ─ kuni *avaldise* väärtus on tõene, korratakse *operaatorit*, kusjuures *avaldise* argumentide väärtusi muudetakse *operaatoris.*

Suunamine on harjumuspärane operaator kujul *goto <märgend>*. Kuivõrd *Kernighan* ja *Ritchie* [KR,65] kirjutasid oma raamatu „süstemaatilise programmeerimise" paradigma tippajal, siis nendivad nad, et *goto* pole kunagi hädavajalik ja selle kasutamist on tavaliselt lihtne vältida, ent et siiski on mõned situatsioonid, kus võiks toda konstruktsiooni kasutada. Üks näide on mitmekordne tsükkel tsüklis, kus *break* katkestab ainult kõige sisesema tsükli, aga tarvis on lõpetada kogu tsükliline töö.

Alamprogrammid.

„Minimaalne" C-programm on tekstifaili xxx.c kompileerimise resultaat xxx.exe. Seejuures saab selles tekstis kodeerida protseduure ja funktsioone, mida kasutavad hiljemdefineeritud alamprogrammid (viimati tuleb defineerida *main*-moodul)[9]. Niisiis, me saame kasutada ainult eelnevalt defineeritud muutujaid, konstante ja alamprogramme. Tavaliselt peab suvaline programm mingil moel saama sisendandmeid ja väljastama resultaate, so. kasutama *BIOS*i ja *DOS*i funktsioone (kui peame silmas PC-masinaid).

Niisiis, iga C-programm peab evima võimalust kasutada kompilaatori standardfunktsioone, mis on rühmitatud oma orientatsiooni järgi. Teek, mille funktsioone tahetakse kasutada, tuleb deklareerida C-teksti alguses, osas, kus tohib ja saab kirjutada C-makrosid. Näiteks, makro

#include <string.h>

toimel lisab C preprotsessor (mis genereerib makrokäskude makrolaiendeid) .c-tekstile päi-sefailist ümberkirjutatud funktsioonide kirjeldused, mis võimaldavad noid funktsioone kasutada.

Iga iseseisvalt täidetavat algoritmi esitav C-tekst sisaldab reeglina järgmisi osi:

- makrod *#include* ja *#define*;
- globaalsete muutujate kirjeldused;
- alamprogrammide (protseduurid ja funktsioonid) koodid;
- peamoodul nimega *main*.

Kompilaatori standardfunktsioonide teekide nimed tuleb paigutada nurksulgude „<..>" vahe-le. Kasutaja saab samuti kirjeldada oma privaatseid mooduleid ─ nende teekide nimed on „jutumärkide" vahel. Muuseas on see võimalus kasutada *C*-keeles masinorienteeritud keeltest ja *FORTRAN*ist tuntud ühisvälju: tuleb kirjutada muutujate ja alamprogrammide kirjeldusi

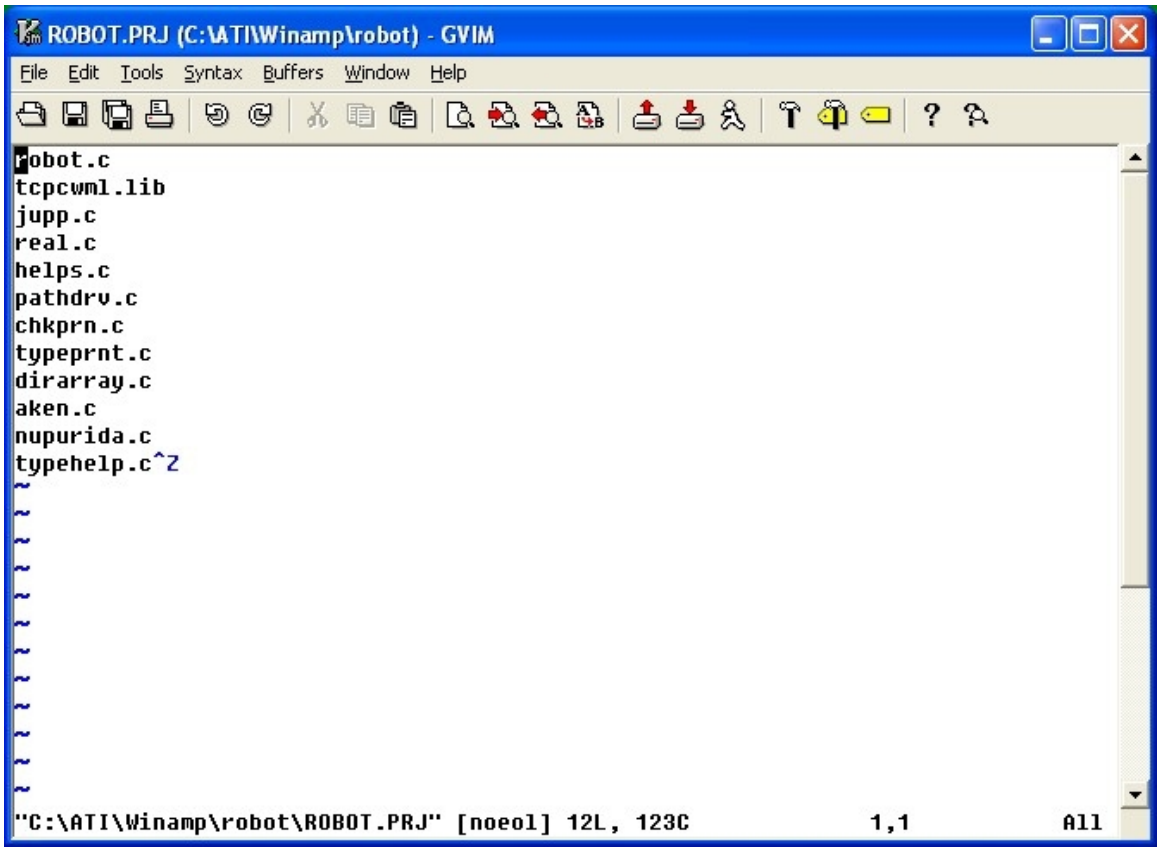
[8] Iga avaldis võib olla „tühi", ent eraldaja „;" on kohustuslik.

[9] Suvalises järjekorras võime alamprogrammide tekste kirjutada siis, kui pärast globaalsete muutujate kirjeldusi esitada nonde alamprogrammide kirjeldused, nt. `void ap(int A, char *b);`

sisaldav päisfail (nimelaiendiga *.h*) ja sellega ühilduv tegelikele muutujatele mälu reserveeriv samanimeline *.c*-fail[10]. Viimane tuleb lülida kõrgema taseme programmi projekti.

„Peaprogramm" võib kasutada suvalise arvu „komponentprogrammide" alamprogramme, kusjuures nondevaheline infovahetus võib toimuda ühisvälja(de) vahendusel. Kõik need „komponentprogrammid" vormistatakse kui iseseisvad *.c*-tekstifailid. Nonde „komponentprogrammide" (*x.c*) nimed tuleb (tavaliselt) paigutada komplekteerimiseeskirja esitava *.prj*- faili koosseisu. Allesitatud pildil on näha ühe sellise projektifaili tekst (*Turbo C*). Seejuures, põhiprogrammi sisaldav tekstifail peab olema esimesena deklareeritud ja sellest nimest tehakse ka *.exe*-faili nimi, näiteprojekti täitmine tekitab faili *robot.exe.*

C erinevad realisatsioonid kasutavad tavaliselt erinevaid „komplekteerimiseeskirju".



Joonis prj. Programmi *robot* projekt.

C-programmis saab käivitada suvalist *.exe*-faili analoogiliselt sellele, kuidas see toimub *MS-DOS*i keskkonnas käsurealt, näiteks kui (pn)=p3, siis järgmiste käskude toimel pöördutakse käsuga *system* programmi *cmp32* poole ─ just nii, nagu näha ka joonisel tts.

```
sprintf(MB,"cmp32 %s",pn);
system(MB);
```

Pärast *cmp32* töö lõppu antakse juhtimine *system*'ile järgnevale käsule.

[10] Vt. lisa xxx ja yyy (jupp.h ja jupp.c).

Joonis tts. Programmi käivitamine käsurealt.

Protseduurid ja funktsioonid.

Nii protseduuridel kui ka funktsioonidel on ühesugune päiseformaat:

tüüp nimi(tüüp parameeter-1, ...,tüüp parameeter-n), kusjuures

protseduuri tüüp on *void*[11] ning funktsioonil tagastatava väärtuse tüüp. Kui viimane on viidatüüp, siis funktsiooni nimele lisandub prefiks „*". Funktsioonist väljutakse käsuga

return(muutuja või konstant);

Standardprogrammide teekide arv varieerub sõltuvalt keele realisatsioonist, ent tavaliselt on nende hulgas järgmised (vt. [KR, lk. 241..258]):

<*assert.h*> — diagnostika;
<*ctype.h*> — sümbolitestid;
<*float.h*> — ujupunktarvude karakteristikud;
<*limits.h*> — *int*-arvude karakteristikud;
<*math.h*> — kogum matemaatilisi funktsioone;
<*signal.h*> — katkestuste käsitlemine;
<*stdarg.h*> — alamprogrammi muutuva väärtuste arvuga parameetri töötlemine;
<*stdio.h*> — sisend- ja väljundfunktsioonid;
<*stdlib.h*> — „kasulikud" (*utility*) funktsioonid, näit. tüübiteisendused, dünaamilise mälu haldamine, *system* käsurea rollis jmt.;
<*string.h*> — stringifunktsioonid;
<*time.h*> — kuupäev ja kellaaeg.

[11] Mõnikord on otstarbekas vormistada protseduur funktsioonina, mille tagastatavat väärtust ei kasutata — näiteks siis, kui protseduuril on mitu sõltumatut lõpetamispunkti; neis on lihtne kirjutada `return(n);` kus *n* on suvaline konstant.

Järgmises jaotises esitame näite, mis peaks illustreerima mitut ülalkäsitletud teemat: .c-faili üldkuju, päisefailide deklareerimine, muutuva väärtuste arvuga parameetri kasutamine, deskriptori tegemine ja kasutamine ning „normaalselt" indekseeritava dünaamilise *n*-mõõtmelise massiiviga operatsioonide kirjeldamine.

Dünaamiline massiiv: array.c.

```
/*   n-mõõtmeline dünaamiline int-massiiv. 20.04.00, VisualC++
6.0  */
/*   see fail ei võimalda kompileerida .exe-faili (main()
puudub, kood on lisatav .prj-faili abil muudele rakendustele
*/

#include <io.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <alloc.h>
#include <stdarg.h>

// massivi kirjeldus
struct descr{
     int *a;     // array: viit massiivile
     int dima;   // dimensioonide arv -- nt 3
     int *dim;   // nt 2 3 2; indeks[i]=1..dim[i]; i=1..dima
     int *DM;    // 0..dima-1: DM[i]=dim[i]*DM[i-1]; DM[0]=1
     };

/*  massiivi  kirjeldamine,  mälueraldus.  n=dimensioonide  arv,
k=dimensioonid.  */
     struct descr *array(int n,int k, ...){
     va_list ap;
     struct descr *D;
     int b,i;
     D=(struct   descr   *)malloc(sizeof   (struct   descr));
//dünaamiline mälueraldus
     memset(D,'\0',sizeof(struct descr));   //saadud      mälu
//täitmine nullidega
     va_start(ap,k);  // initsialiseerib argumentide listi
     D->dima=n;
     D->dim=(int *)malloc(n*sizeof(int));
     memset(D->dim,'\0',n*sizeof(int));
     D->DM=(int *)malloc(n*sizeof(int));
     memset(D->DM,'\0',n*sizeof(int));
     D->dim[0]=k;
     for(i=1;i<n;i++)  D->dim[i]=va_arg(ap,int);   /*  tsükkel
                          üle argumentide */
     va_end(ap);  /* argumentide listi kustutamine --
          süsteemsed parameetrid! */
```

```
     printf("rajad: ");
     for(i=0;i<n;i++) printf("%d ",D->dim[i]); printf("\n");
     D->DM[0]=1;
     for(i=1;i<n;i++) D->DM[i]=D->DM[i-1]*D->dim[i-1];
     printf("DM: ");
     for(i=0;i<n;i++) printf("%d ",D->DM[i]);
     b=1;
     for(i=0;i<n;i++) b=b*D->dim[i];
     D->a=(int *)malloc(b*sizeof(int));
     memset(D->a,'\0',b*sizeof(int));
     printf("\n");
     return(D);
}

/* massiivi D elemendi lugemine: k on indeksite loetelu
   NB! indeks[i]=1..dim[i] */

int *val(struct descr *D,int k, ...){
     va_list ap;
     int i,n;
     int x;
     va_start(ap,k);
     n=D->dima;
     x=k-1;
     for(i=1;i<n;i++) x=x+D->DM[i]*(va_arg(ap,int)-1);
     va_end(ap);
     return(D->a[x]);
}

/* massiivi D kirjutamine: v on väärtus, k on indeksite
loetelu */

void wri(int v,struct descr *D,int k, ...){
     va_list ap;
     int i,n;
     int x;
     va_start(ap,k);
     n=D->dima;
     x=k-1;
     for(i=1;i<n;i++) x=x+D->DM[i]*(va_arg(ap,int)-1);
     va_end(ap);
     D->a[x]=v;
     }
```

C ja protsessori-tase.

Selles jaotises vaatleme põgusalt *C*-keele neid nüansse, mis teevad temast süsteemiprogrammeerimise keele. Loodetavasti on järjekindel lugeja adunud, et näiteks operatsioonisüsteemi programmeerimiseks peab olema võimalik (kui keskenduda *Intel*-protsessorile) evida juurdepääsu:

- aparatuursetele registritele
- vaba mälu piiraadressidele ja nende muutmisele;
- kui tegemist on *MS-DOS*-süsteemiga, siis kõigile *DOS*- ja *BIOS*-katkestustele:
- võimalusele katkestusi genereerida.

Lisades biosh ja dosh on toodud vabavaralise *GNU-C*[12] päisefailide *BIOS.H* ja *DOS.H* tekstid, millega tutvumine peaks andma ettekujutuse C-keele võimalustest tööks „madalaimal tasemel".

Tuleb rõhutada, et tänu masinorienteeritusele erinevad erinevate arvutite *BIOS*id üksteisest detailides, mistõttu võib ühel platvormil kirjutatud ja *BIOS*i funktsioone kasutatavate programmide ülekandmine teisele platvormile nõuda mõningat „kohendamist"; näiteks *IBM*i *PC-BIOS* ja selle funktsioonid pole üksüheses vastavuses *GNU-C* oma(de)ga, ent kui sellega oskame arvestada, pole lisatöö kuigi märkimisväärne.

Mõned *BIOS*i funktsioonid on dubleeritud *C* funktsioonidega, näiteks *IBM PC* funktsioonid *putch()* või *getche()*, ent enamik pole ning nende kasutamiseks tuleb neid kasutada „ilmutatud kujul". *IBM PC* võimaldab seda funktsiooni *int86(INT,&inregs,&outregs)* abil. Parameetrid on järgmised:

- *INT* on *int*-tüüpi katkestuse number
- *&inregs* on registrite sisendväärtused (vaikimisi kehtivad väärtused)
- *&outregs* registrite tagastatud väärtused

Robert Lafore [lafore 327] toob järgmise näite (kommentaarid on meie tõlgitud):

```
/* memsize0.c */
/* trükib mälu mahu */
#define MEM 0x12    /* BIOSi katkestuse number */
main()
{
     struct WORDREGS    /* 16-bitised registrid */
          {
          unsigned int ax;
          unsigned int bx;
          unsigned int cx;
          unsigned int dx;
          unsigned int si;
          unsigned int di;
          unsigned int flags;
          };
     struct BYTEREGS    /* 8-bitised registrid */
          {
          unsigned char al,ah;
          unsigned char bl,bh;
          unsigned char cl,ch;
          unsigned char dl,dh;
          };
     union REGS          /* nii baidid kui ka sõnad */
          {
```

[12] Copyright © 1989, 1991 Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

```
          struct WORDERGS x;
          struct BYTEREGS h;
          };
     union REGS regs;
     unsigned int size;
     int86(MEM,&regs,&regs);
     /* „sisendregistreid” ei muudetud */
     size=regs.x.ax;          /* mälu maht AX-registrist */
     printf(„Memory size is %d Kbytes”,size);
}
```

Järgmises näites kasutatakse *IBM PC DOS*-funktsiooni peitmaks kursorit [lafore 331].

```
/* curoff.c */
#include <dos.h>    /* seal on defineeritud union REGS */
#define CURSIZE 1   /* teenus „set cursor size” */
#define VIDEO 0x10  /* BIOSi videokatkestuse number */
#define STOPBIT 0x20  /* see bitt lülitab kursori välja */
main()
{
     union REGS regs;
     regs.h.ch=STOPBIT;
     regs.h.ah=CURSIZE;
     int86(VIDEO,&regs,&regs):
     /* videokatkestuse väljakutse */
}
```

Meie õppevahendi piiratud maht ega ka eesmärgid ei võimalda esitada komplitseeritumaid näiteid; huviline võiks noist lihtsamatega tutvuda näiteks *Robert E. Marmelstein*i raamatu [games] abil.

Kokkuvõtteks.

Niisiis, *Ritchie, Thompson* jt. (eeskätt *Kernighan*) töötasid *C* välja utilitaarsetel eesmärkidel (kanda *UNIX* võimalikult kiiresti üle kardinaalselt uue arhitektuuriga masinale (*PDP-11*) . Polnud ei aega ega ka mõtet pöörata (liigset) tähelepanu keele „akadeemilistele aspektidele” nagu täitmisaegne turvalisus (näiteks indeksite ja dünaamilise mälu piiride kontroll, vaba juurdepääs protsessori-tasemele jmt), süntaksi selgus või kogu täidetava programmi lähteteksti tervikesitus. Tähtis oli funktsionaalsus ning efektiivsus. Ja tulemuseks oli keel, mis on tänaseks kõige populaarsem süsteemprogrammeerimise instrument ja mille leksika ning süntaks on olnud eeskujuks paljudele *C*-st sootuks erinevate orientatsioonidega keeltele (näiteks *Java* või *PHP*).
Joonisel *family* on näidatud, et *C* „esivanem” on *Algol-60.* Kahtlemata sarnaneb *C* rohkem *Algol*ile kui *FORTRAN*ile, *COBOL*ile või muudele kaasaegsetele teerajajatele (näit. *Lisp*ile, *APL*ile või *Snobol*ile, rääkimata juba *FORTH*ist). Johtuvalt *Thompson*i eelnevast kogemusest *Algol*-sarnaste keelte *BCPL*i ja *B*-ga oli normaalne, et mitmes mõttes sai just *Algol* uue keele prototüübiks — kui peame silmas ranget tüpiseerimist (kõik kasutatavad objektid, nii liht-

muutujad, agregaadid kui ka alamprogrammid tuleb enne kasutamist täpselt kirjeldada)[13], operaatoreid (avaldised, loogilised tingimused, *for*- ja *while*-tsüklid, *switch*-tüüpi lüliti, märgendid ja suunamine, rekursioon) ning alamprogrammide kirjeldamise võimalust põhiprogrammi tekstiga samas „failis".

Ent samas loobusid *C* autorid plokkstruktuurist, alamprogrammide parameetrite edastamisest *by name* jmt. ning tõid juurde masinorienteeritud keeltest tuntud võimalused: eraldi kompileeritavad alamprogrammid, (varjatud) ühisväljad, dünaamilised andmestruktuurid (*struct* ja *union*, ent paradoksaalselt ei toeta ta dünaamilisi massiive), viidatüübi ning juurdepääsu protsessoritasemele.

C on suhteliselt väike (kui võrrelda *COBOL*i, *PL\I* või *ADA*ga) ja lihtsalt realiseeritav keel, ent tööriistana sobib ta pigem kogenud kui algajatele programmeerijatele[14].

Ja veel „inimlikust aspektist" andis sellele keelele hinnangu umbes 25 aastat tagasi TRÜ ja Küberneetika Instituudi ühise suvekooli Moskvast tulnud noor lektor *Vladimir Serebrjakov*, kui talt küsisime (ise kogemust omamata), et kuidas *C* tundub. *Volodja* ütles: „Teate, see on midagi narkootikumi-sarnast.. Alul on vastik, aga pärast ei saa lahti."[15]

[13] Samas puudub täitmisaegne tüübikontroll, mis annab head võimalused vigaste programmide kirjutamiseks [Wilson, 50].
[14] See kehtib kõigi süsteemprogrammeerimise keelte kohta.
[15]« Знаете, это что-то вроде наркотика: сначала противно, а потом не отвыкнешь».