ДА ЛИ ЈЕ *C* ДОБАР ПРОГРАМСКИ ЈЕЗИК?

Драган Машуловић, Зоран Будимац
Нови Сад

Последњих неколико година смо сведоци велике популарности и експанзије програмског језика *C*. Као његове основне предности се најчешће помињу велика ефикасност извршног кода, велика преносивост изворног кода, као и то да се "у *C*-у може све". *C* је такође постао и језик на коме је врло "модерно" програмирати. Осим тога, "сви прави програмери програмирају на *C*-у", "то је језик за професионалце", ... Чак и програмери почетници почињу да програмирају у *C*-у, никад не упознајући предности и мане других програмских језика и тако остају лишени могућности да упознају и лоше стране *C*-а.

У овом тексту се питамо да ли је популарност *C*-а заслужена, да ли је *C* заиста "добар" програмски језик и колико је лако у њему писати велике програме. Да бисмо боље илустровали неке особине и мане *C*-а, те да бисмо пружили и добру алтернативу, често ћемо *C* поредити са програмским језиком Modula-2. Од многих особина који карактеришу добар програмски језик, овде ћемо се задржати само на најзанимљивијима. Читаоце које више занима ова проблематика, упућујемо на литературу наведену на крају чланка (при чему је [3] послужио као основа за овај чланак).

Кратка историја

Програмски језик *C* су креирали Dennis Ritchie и Brian Kernighan 1974. године за потребе једног конкретног пројекта: реализације оперативног система Unix на рачунару PDP-11. Modulu-2 је 1982. године креирао Niklaus Wirth са идејом да поправи неке недостатке Pascal-а (Wirth-овог претходног језика), те да направи језик који би имао и могућности системског програмирања. У Moduli-2 је убрзо после тога направљен оперативни систем за рачунар Lilith.

Модуларност

За *C* се често може чути да је модуларан програмски језик. Модуларност међутим подразумева: (1) могућност да се програм разбије у више модула (програмских фајлова), (2) могућност засебног превођења модула (енгл. *separate compilation*), (3) сакривање информација (енгл. *information hiding*), и (4) инкременталност (после измене модула M, аутоматски се преводе сви модули који зависе од M).

Modula-2 јесте модуларан језик. Од горе наведених особина, *C* у потпуности омогућава само разбијање програма у више модула. *C* не преводи модуле (програмске фајлове) *засебно* него *независно* (енгл. *independent compilation*), а то је један слабији концепт који подразумева да су сва имена која су декларисана у другим модулима исправно декларисана. Даље, *C* модули (тачније речено програмски фајлови) не могу од других модула да "сакрију" дефиниције типова података. И најзад, *C* преводилац после измене неког модула неће аутоматски превести све модуле који од тог модула зависе (за те потребе постоје посебни програми којима је искључиво уз помоћ програмера могуће постићи сличан ефекат).

Ево, за пример, једног *C* модула који није добар:

```
/* mylib.h -- definicija modula */
long Fib(long n);

/* mylib.c -- implementacija modula */
#include <stdio.h>
long Fib(long n1, long n2) { return n1 + n2; }
```

У `.h` датотеци пише да се модул састоји од једне функције која има само један аргумент. Но, у `.c` датотеци пише да та функција има два аргумента. Ето грешке! Посматрајмо сада програм који користи модул `mylib`:

```
/* test.c */
#include <stdio.h>
#include "mylib.h"
void main(void)
{ long i;
  for(i = 1; i <= 20; i++) printf("%ld\n", Fib(i)); }
```

Грешка (тзв. инконзистентност дефиниције и имплементације) ће *можда* бити откривена тек у току повезивања програма ("линковање"), а то је неопростиво! Неки *C* преводиоци неће уопште реаговати, док мало боље организовани *C* преводиоци уписују додатне информације у објектни код како би бар у току повезивања програма открили грешке овог типа.

У језику Modula-2 овакве грешке се откривају још у току превођења. Превођење сваког модула подразумева проверу конзистентности дефиниције и имплементације тог модула.

Размотримо и следећи пример. Претпоставимо да имамо два модула `P` и `Q`, и да се у оба јавља променљива `x`. Претпоставимо такође да у главном програму треба користити обе променљиве `x`. У следећем *C* програму:

```
#include "P.h"
#include "Q.h"
void main(void) { ... x ... }
```

ће `x` бити једно од два могућа `x`-а (из `P` или `Q`) што зависи од реализације *C* преводиоца. Такође у зависности од реализације, *C* преводилац ће опоменути програмера о могућем проблему (генерисаће *"warning"*) или неће, али неће прекинути компилацију.

Одговарајући пример у Moduli-2 изгледа овако:

```
MODULE Test;
IMPORT P, Q;
BEGIN ...P.x ... Q.x ...
END Test.
```

`P.x` значи "`x` из модула `P`". Ова једноставна и природна конвенција омогућује да се обе променљиве `x` користе равноправно и без забуне.

C дакле, насупрот раширеном мишљењу, *није* модуларан језик.

Може се поставити питање због чега инсистирати на оваквим "ситницама". Посматрано на малим примерима то и јесу ситнице које се могу решити крајње једноставно. Но, када се узме у обзир да се ови програмски језици користе за писање *огромних* програма, као и то да се велики програми углавном пишу тимски, поменути недостаци у дизајну *C* језика више нису ситнице.

Поузданост

Под поузданошћу програмског језика ћемо овде подразумевати (не)могућност прављења грешака које се јављају током извршавања програма, а манифестују се неочекиваним понашањем програма током извршавања ("заглављивањем", "ресетовањем" рачунара и сл.). У истраживању из [2] је утврђено да се програмирањем у *C*-у прави три пута више грешака него програмирањем у Moduli-2. Тачније речено Modula-2 преводилац открије током превођења три пута више грешака (које ће се касније јавити током извршавања програма), него *C* преводилац.

Непостојање строге провере типова је основни узрок великог броја грешака у *C* програмима. У наредбама *C* програма је могуће мешати "бабе и жабе" (на пример, сабирати меморијске адресе, бројеве и логичке вредности) што врло често доводи до каснијих проблема. Искуство једног од аутора овог чланка казује да позив једне функције са погрешним бројем аргумената (већ смо видели да је то могуће) може да доведе до врло неконзистентног понашања програма и *веома* тешког проналажења узрока такве грешке.

Читкост

Иако читкост програма може бити субјективна категорија којој и (не)дисциплина програмера може да одмогне (или помогне), сматрамо да је *C* много подеснији за писање нечитких програма. На пример, свакоме ко се бар мало разуме у програмирање или у енглески језик јасно је шта представља следећи низ декларација у Moduli-2:

```
TYPE P1 = POINTER TO PROCEDURE(CHAR): INTEGER;
     P2 = POINTER TO PROCEDURE(INTEGER): P1;
     A  = ARRAY [0 .. 99] OF INTEGER;
```

док је у аналогним дефиницијама у *C*-у готово немогуће пронаћи некакво интуитивно значење без претходног тренинга:

```
typedef int (*P1)(char);
typedef P1  (*P2)(int);
typedef int A[100];
```

Слично важи и за декларацију `static int i`, која (у ствари) означава скривену целобројну променљиву `i`, или за декларацију `signed char c` која (у ствари) означава променљиву типа "кратки цео број" (у Moduli-2 би ова последња декларација гласила: `VAR c: SHORTINT`).

Да бисмо боље истакли "могућности" које *C* пружа у писању нечитких програма, на крају текста се налази неколико задатака. Покушајте да закључите шта поједини *C* програми раде (неће вам бити лако!)

Програмирање ниског нивоа

И Modula-2 и *C* имају могућност програмирања ниског нивоа: директно приступање меморији и портовима рачунара, манипулисање појединачним битовима, адресна аритметика, програмирање прекида итд. (иако се често тврди да такве могућности има једино *C*).

Већина директног манипулисања битовима се у Moduli-2 обавља скуповним операцијама. Постоји тип података који се зове BITSET и који најчешће представља једну машинску реч рачунара. Да би се "укључио" или "искључио" неки бит користе се одговарајуће скуповне операције: убацивање редног броја бита у скуп и избацивање броја из скупа. Поред тога Modula-2 има и операторе за померање садржаја меморијске речи, а за потребе програмирања ниског нивоа, могуће је и "ослабити" (контролисано!) правила строге провере типова.

У *C*-у се манипулација битовима врши искључиво операторима за померање садржине меморијске речи и логичким операторима.

У следећим примерима се укључује трећи бит 16-битне меморијске речи на адреси B800:0000, у *C*-у:

```
long *s
...
s = (long *)0xb80000;
*s |= 0x8;
```

па затим у Moduli-2 (сматрамо и овде да је Modula-2 код знатно читљивији):

```
VAR s [0B800H:0000H] : BITSET;
...
INCL(s, 3);   (* ili s := s + {3} *)
```

Поред тога, Modula-2 независно од оперативног система, подржава рад са процесима, корутинама и прекидима (што значи да је то брига преводиоца језика, а не програмера). У *C*-у се слични ефекти могу постићи позивима функција оперативног система, али таква решења захтевају детаљније познавање оперативног система и такви програми углавном нису преносиви, о чему говори следећи одељак.

Преносивост

Као једна од главних предности *C*-а истиче се преносивост његових програма са платформе на платформу (рачунар, оперативни систем и/или преводилац програмског језика). Да ли је заиста тако?

Следећи део *C* програма

```
int x;
x &= 0177770;
```

ће доња 3 бита у интерној репрезентацији целог броја поставити на нулу (на рачунару PDP-11). На рачунару VAX ће, међутим, поред тога "очистити" све из горње полу-речи. Даље, посматрајмо нешто компликованији део *C* програма:

```
int i, *ip; char *cp; char memblock [] = {"abcdefghijk"};
                      /* memblock je "poravnat" */
cp = memblock; cp++;
ip = (int *) cp;
i = *ip;
```

Пођимо од реалне претпоставке да `memblock` почиње на парној адреси (што обично и јесте случај). Након `cp = memblock`, `cp` за своју вредност добија адресу одговарајућег блока меморије (што је, по претпоставци, паран број). После `ip = (int *) cp`, `ip` садржи исту адресу као и `cp` (непаран број). Тада наредба `i = *ip` покушава да "добави" два бајта са непарне адресе, што ће на различитим рачунарима произвести различите ефекте.

Видимо да *C* програми нису нарочито преносиви (може се чак рећи да су мање преносиви од Modula-2 програма, као што смо наговестили у претходном одељку). Ако су *C* програми преносивији од програма писаних у неким другим програмсикм језицима, то је само због тога што су библиотеке функција које се могу позивати из *C* програма стандардизоване и што су релизоване на свим познатијим платформама. Уз стандардизоване библиотеке међутим, програми сваког програмског језика постају преносивији од *C* програма (једна таква библиотека за Modulu-2 се управо креира на Институту за математику у Новом Саду).

Ефикасност генерисаног кода

Генерисани код и *C* и Modula-2 програма је ефикасан. Стандардни пример *C* кода којим се демонстрира ефикасност генерисаног кода:

```
while(*p++ == *q++);
```

у Moduli-2 изгледа овако:

```
WHILE p^ = q^ DO IncAddr(p, 1); IncAddr(q, 1)
END
```

и ефикасност њиховог извршавања је на савременим рачунарима приближно једнака. (Наведени *C* програм би се (само) на PDP-11 извршавао значајно брже јер је конструкција `*p++` директан аналогон одговарајуће

машинске инструкције. PDP рачунари међутим углавном више нису у употреби.) Напоменимо још да за конструкције сличног типа у Moduli-2 најчешће нема потребе—на пример, структура (низ, слог, итд.) `b` се у структуру `a` копира на следећи (најефикаснији могући) начин: `a := b`.

Генерално, ефикасност *C* и Modula-2 кода је једнака. Напоменимо ипак да је по тестовима произвођача TopSpeed Modula-2 преводиоца од пре неколико година, тај преводилац (верзија 1) производио далеко најбржи код на PC рачунарима: неколико пута бржи од Turbo Pascal-a и 50% бржи од најбржег *C* преводиоца. Од тада ниједан произвођач ниједног преводиоца ни за један програмски језик на PC рачунарима није оспорио ове резултате. Ови резултати су потврђени и на интерним тестовима урађеним на Институту за математику у Новом Саду.

Закључак

Програмски језик *C* *није* модуларан, *не* производи ефикаснији код од неких других језика, *нема* највеће и најбоље могућности ниског нивоа (напротив и оне које има су често непреносиве), *C* програми *нису* читки (напротив), *C* програми *нису* преносивији од програма у неким другим програмским језицима (а ако и јесу, то је због "спољних" фактора) и (што је можда и најбитније) у *C* програмима се греши више и лакше него у другим програмским језицима. При свему томе у чланку нисмо говорили о многим другим (лошим) особинама програмског језика *C* (на пример, немогућности да се дефинишу локалне функције, немогућности да се аргументи функција по избору преносе по вредности или по "референци", итд.).

C дакле *није* добар програмски језик (у ствари, то је један од најлошијих програмских језика икада створених). Како је онда могуће да је постао толико популаран? На то питање нема једноставног и кратког одговора. Рецимо само да се сличне ствари у индустрији дешавају често: на пример, VHS систем је у видео индустрији далеко популарнији и раширенији од BETA система, иако је овај други неупоредиво квалитетнији.

Литература

[1] Angele J., Kupper D.: "Modula-2 an Alternative C?", SIGPLAN Notices, vol. 27, no. 4, 1992, pp. 17–26.

[2] Griffits L. K., "Modula-2 is three times less error prone than C", Proceedings of 2nd International Modula-2 Conference, Loughborough, England, 1991.

[3] Mašulović D., Budimac, Z., "C ili Modula-2?", Zbornik radova 9. konferencije ETRAN-a, Zlatibor, Sveska III, str. 183–186

Задаци

Шта раде следећи *C* програми?

1.
```
int x;
x =+ 10;
x += 10;
```

2.
```
long *ptr;
*ptr=*ptr++;
*ptr++=*ptr;
```

3.
```
/* Duff's device */
register n = (count+7)/8;
/* count > 0 */
switch (count % 8)
{
case 0: do{*to = *from++;
case 7:    *to = *from++;
case 6:    *to = *from++;
case 5:    *to = *from++;
case 4:    *to = *from++;
case 3:    *to = *from++;
case 2:    *to = *from++;
case 1:    *to = *from++;
       } while (--n > 0);
}
```

4.
```
/* by Brian Westley, 1988 */
#define _ -F<00||--F-00--;
int F=00,00=00;
void main(void)
{F_00();
 printf("%1.3f\n",4.*-F/00/00);}

void F_00(void)
{
              _-_-_-_
          _-_-_-_-_-_-_-_-_
       _-_-_-_-_-_-_-_-_-_-_-_
     _-_-_-_-_-_-_-_-_-_-_-_-_-_
   _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
   _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
     _-_-_-_-_-_-_-_-_-_-_-_-_-_
       _-_-_-_-_-_-_-_-_-_-_-_
          _-_-_-_-_-_-_-_-_
              _-_-_-_
}
```

Статијата прв пат е објавена во списанието Тангента на ДМ на Србија во 1997/98 година