Милан Чабаркапа

РЕКУРЗИВНЕ ФУНКЦИЈЕ

Једна од најтежих области за разумевање у програмирању је рекурзија. Скоро свака књига из програмирања се дотиче рекурзија, али најчешће илустровањем рекурзије на примеру израчунавања факторијела и без дубљег понирања у механизам њеног функционисања. Није случајно да је Bjorn Stroutstrap, творац C^{++}-а, нашао за потребно да, у књизи у којој излаже овај данас најраспрострањенији програмски језик у свету професионалног програмирања, цитира непознатог аутора који каже: „Итерација је људска, рекурзија је божанска“.

Рекурзија је начин организације процеса рачунања, у коме се функција у току извршавања њених оператора обраћа сама себи, директно или индиректно. Рекурзија је моћно средство за креирање програма, којим се огроман број операција може записати компактним програмским кодом. Шаљива дефиниција рекурзије би се могла дати као енциклопедијско тумачење: *Рекурзија* је – види под рекурзија.

Иако рекурзивна форма организације алгоритма обично даје компактнији програмски код него друга решења неког проблема, она троши више меморије за регистровање података и времена за организацију рекурзивних позива функције.

Анализирајмо прост пример директне рекурзивне функције:

```
void main()
  {
  printf("!");
  main();
  printf("?");
  }
```

Извршавањем првог оператора функције `main()` штампа се знак: !. Другим оператором функција `main()` се обраћа самој себи. Значи, функција `main()` се извршава поново, па се опет штампа знак !. Теоретски, ове се акције понављају бесконачно и трећи оператор се никада не извршава. Програм се прекида једино „насилно“, притиском на `Ctrl/Break` или искључивањем рачунара. Овде смо се суочили са једним од битних питања при реализацији рекурзија: када и како обезбедити излаз из рекурзије? Зато, већ у првој фази размишљања при креирању рекурзивног алгоритма, треба – обезбедити излаз из рекурзије!

Сада ћемо написати функцију која исписује бројеве $1, 2, \ldots, n$ прво у инверзном, а затим у директном поретку:

```
/* Ispis prvih N prirodnih brojeva u inverznom pa direktnom poretku */
void xxx(int n)
  {
  printf("%d",n); /* 1 */
  if (n>1) xxx(n-1); /* 2 - rekurzivni pozivi se prekidaju za n=1*/
  printf("%d ",n); /* 3 */
  }
/* Test funkcije */
void main()
  {
  int n;
  scanf("%d",&n);
  xxx(n);
  }
```

Овде можемо уочити да се рекурзивни позиви прекидају када n добије вредност 1. Пратићемо које се акције реализују за разне вредности променљиве n.

Ако је $n = 1$, реализује се позив `xxx(1)` у коме се извршавају само први и трећи оператор, тако да се исписује

11

Обраћање `xxx(n-1)` (оператор 2) се неће реализовати, јер услов `(n>1)` није задовољен. Дакле, за $n = 1$ нема рекурзивног позива.

За $n = 2$ прво се изврши оператор 1 (испис броја 2), а затим, пошто је `(n>1)` реализује поновно обраћање функцији – `xxx(1)`, али са другим стварним параметром. Према томе, извршавање `xxx(2)` се реализује као:

```
printf("%d",n); // posto je n=2 ispisuje se 2
if (n>1) xxx(n-1); // poziva se xxx(1) u kome se ispisuje 11,
printf("%d",n); // ispisuje 2
```

и исписује се 2112.

Сваки позив функције, без обзира да ли је рекурзивна или не, значи да се на stack-у креира копија вредности њених формалних параметара и локалних променљивих, а затим се управљање предаје првом извршивом оператору функције. Шта је *stack*? То је меморијски сегмент за привремено чување података који ради по LIFO (Last In First Out) принципу. То значи да се податак који је последњи постављен у stack први узима из stack-а.

Према томе, податак који се први уписује у stack биће на његовом дну, а податак који се уписује последњи, биће на врху stack-а. Над stack-ом се примењују две операције: "push" – постави у stack, и "pop" – узми из stack-а. Када се позове функција, њени формални параметри и локалне променљиве се уписују у stack, а када се заврши, „скидају" се са stack-а.

Промене на stack-у и редослед извршавања оператора анализираћемо пратећи извршавање претходне функције за $n = 3$, тј. `xxx(3)`. Коментари на слици 1 указују на редослед извршавања оператора. Иако позив функције у рекурзивној функцији не значи креирање у меморији нове копије функције, већ само њених параметара и локалних променљивих, на слици 1 ради очигледности уз сваки позив функције рећи ћемо да се креира примерак функције (само фиктивно, ради лакше анализе) и копија њених параметара и локалних променљивих.

Слика 1

Првим позивом функције са `xxx(3);` реализују се следеће акције:

- креирање првог примерка функције и копирање вредности формалног параметра ($n = 3$) на stack;
- испис вредности n (3);
- позив, пошто је $n > 1$, функције `xxx(2)`.

Другим позивом функције `xxx(2);` реализују се следеће акције:

- креирање другог примерка функције и копирање вредности формалног параметра ($n = 2$) на stack;
- испис вредности n (2);
- позив, пошто је $n > 1$, функције `xxx(1)`.

Трећим позивом функције `xxx(1);` реализују се следеће акције:

- креирање примерка функције и копирање вредности формалног параметра ($n = 1$) на stack;
- испис вредности n (1), по првом `printf()` оператору;
- испис вредности n (1), по другом `printf()` оператору;
- напуштање трећег примерка функције и скидање њених параметара и локалних променљивих са stack-а (овим се губи сваки траг о параметрима и локалним променљивим трећег примерка функције).

Излазом из трећег примерка функције и повратком на оператор иза оператора позива – `printf()`, у другом примерку функције реализује се:

- испис текуће вредности формалног параметра n (2);
- напуштање другог примерка функције и скидање њених параметара и локалних променљивих са stack-а (затамњени податак на слици постаје недоступан);
- ...

Овај поступак се аналогно понавља до повратка на оператор који се налази иза оператора првог позива рекурзивне функције.

Сваки пажљив читалац ће рекурзивно решавање оваквог и сличних проблема сматрати непотребним компликовањем нечега што се лако решава без рекурзије. А такав је и у књигама већ класичан пример за илустровање рекурзије – израчунавање факторијела. Сврха оваквих решења (а биће их још) је да се на једноставним примерима читалац навикне на рекурзивни начин размишљања, тако да му природа и начин функционисања рекурзије не представљају сметњу у овладавању сложенијим алгоритмима (комбинаторни, backtracking, итд.) и структурама (бинарна стабла) који су по својој природи рекурзивни и који се тешко решавају другим путем.

Следећи проблем се тешко решава нерекурзивно, с обзиром да дужина низа знакова који се учитавају није унапред позната.

Пример 1. Нека се са улаза учитава низ знакова чија дужина није унапред позната, а обележје краја низа је: $*$. Написати функцију која обезбеђује испис низа у инверзном поретку. На пример, ако су улазни подаци

*abcde**

рачунар треба да испише

**edcba*

Написаћемо програм са рекурзивном функцијом `inverz()`:

```
#include <stdio.h>
void main()
  {
  void inverz();
  inverz();
  }
void inverz()
  {
  char x;
  if ((x=getchar())!='*') inverz();
  putchar(x);
  }
```

Рекурзивна функција `inverz()` позива саму себе док се не учита `'*'`. Након тога се, позивом `putchar()`, исписују учитани знаци али у обрнутом поретку. После сваког исписа излази се из текуће функције и контрола предаје функцији из које је био позив – непосредно иза оператора којим је позвана функција. Функција `inverz()` нема параметара. Број обраћања функцији зависи од дужине низа и капацитета stack-а. Анализираћемо неке случајеве.

Ако се низ састоји из једног знака и то: $*$ (не треба заборавити и тај случај):

- први оператор функције прочита вредност и додели x;
- извршавањем `if` се не позива функција, јер је x једнако `'*'`, већ се прелази на оператор `putchar(x)` којим се исписује: $*$;
- прекида се извршавање функције.

Ако се низ састоји из два знака (први различит од знака: $*$, а други њему једнак), то се прво променљивој x додељује вредност различита од: $*$. Услов је задовољен, па се поново извршава функција `inverz()`. У другом примерку функције се прочита следећи знак, и додели променљивој x. Пошто је то: $*$, трећим оператором се исписује тај знак (дакле исписује се: $*$). У току процеса рачунања реализује се повратак у први примерак функције, где се извршава оператор `putchar(x)` – тј. штампа вредност првог податка, који је додељен променљивој x уочи обраћања другом примерку функције.

Није тешко приметити да низу од n знакова одговара n примерака функције `inverz()`. Пошто је оператор `putchar();` иза оператора обраћања истој функцији, обезбеђује се испис улазних података у инверзном поретку.

Пример 2. Написати рекурзивну функцију за израчунавање суме првих n природних бројева: $1 + 2 + 3 + \cdots + n$.

Проблем ћемо дефинисати рекурзивно. Ако суму првих n природних бројева означимо са s_n, онда је $s_0 = 0$, $s_i = i + s_{i-1}$.

Ову рекурентну везу можемо проверити за конкретну вредност, на пример 4.

$$s_4 = 4 + s_3 = 4 + 3 + s_2 = 4 + 3 + 2 + s_1$$
$$= 4 + 3 + 2 + 1 + s_0 = 4 + 3 + 2 + 1 + 0 = 10.$$

На основу рекурзивне дефиниције није тешко саставити функцију која решава проблем:

```
int suma(int n)
  {
  if (n==0)
    return 0;
  else return n+suma(n-1);
  }
```

Сл. 2

Ток извршавања функције при позиву `suma(3)` илуструје слика 2.

Пример 3. Написати рекурзивну функцију за израчунавање факторијела броја n.

Из формуле:

$$n! = \begin{cases} 1, & \text{ако је } n = 1, \\ (n-1)! \cdot n, & \text{ако је } n > 1, \end{cases}$$

добија се рекурентна формула за рачунање факторијела:

$$\text{fakt}(n) = \begin{cases} 1, & \text{ако је } n = 1, \\ n \cdot \text{fakt}(n-1), & \text{ако је } n > 1. \end{cases}$$

Сада се лако описује одговарајућа функција `fakt()`:

```
long fakt(int n)
  {
  if (n!=0)
    return (n*fakt(n-1));
  else return ((long)1);
  }
```

Пример 4. Солитер од n спратова треба да се кречи под следећим условима:

- сваки спрат се кречи или бело, или плаво;
- не смеју бити 2 плава спрата један изнад другог.

На колико начина се може окречити једна n-тоспратница?

Нека је k_n број различитих кречења n-тоспратнице под датим условима. Тада је k_n једнак збиру: броја кречења у којим је први спрат бео – b_n, и броја кречења у којим је први спрат плав – p_n. То значи: $k_n = b_n + p_n$. Боје изнад белог првог спрата могу бити бела или плава, па је $b_n = b_{n-1} + p_{n-1} = k_{n-1}$. Изнад плавог спрата може бити само бео спрат, па је $p_n = b_{n-1} = k_{n-2}$. Дакле,

$$k_n = b_n + p_n = (b_{n-1} + p_{n-1}) + b_{n-1} = k_{n-1} + k_{n-2}, \quad k_1 = 2, \quad k_2 = 3, \quad n \geqslant 3.$$

Сада је лако написати одговарајућу рекурзивну функцију:

```
int k(int n)
  {
  if (n==1) return 2;
  if (n==2) return 3;
  return k(n-1)+k(n-2);
  }
```

Рекурзија је моћно средство за решавање многих задатака, али је треба користити врло обазриво. Има ли смисла да се користи за израчунавање суме или факторијела као у претходним примерима? Сигурно не, јер су итеративна решења у оваквим случајевима несумњиво много простија, ефикаснија и, што је најважније, природнија. Треба имати на уму да у рекурзивним решењима сваки позив функције значи одвајање на stack-у простора за параметре и локалне променљиве, па се таквим решењима троши више меморије и успорава извршавање програма. Према томе, рекурзивну функцију за израчунавање суме или факторијела треба посматрати само као илустрацију рекурзивне технике на простим и очигледним примерима. Када треба применити рекурзију? Само када је природа проблема рекурзивна тако да се тешко трансформише у итеративни еквивалент или када је ефикасност рекурзивног решења задовољавајућа. Ако мислите да је рекурзија компликовање једноставних решења пробајте да решите нерекурзивно следећи проблем.

Пример 5. „Ханојска кула“. На плочи се налазе три стуба. На левом стубу се налази n дискова чији полупречници опадају идући ка врху стуба. Написати програм којим се исписују кораци потребни да се дискови са левог стуба пребаце на десни тако да се у једном кораку може пребацити само један диск. Забрањено је ставити већи диск преко мањег. (Ово је игра коју су играли браманси свештеници са 64 златна диска верујући да када се игра заврши то ће бити и крај света.)

Сл. 3

Стубове можемо нумерисати са 1, 2, 3 као на слици 3.

Тривијалан случај за $n = 1$ се решава пребацивањем диска са стуба 1 на стуб 3.

Ако треба пребацити 64 диска са стуба 1 на стуб 3, проблем се разбија на три потпроблема:

$$\text{prebaci}(64,1,3) \leftrightarrow \begin{cases} \text{prebaci}(63,1,2) \\ \text{prebaci disk sa 1 na 3} \\ \text{prebaci}(63,2,3) \end{cases}$$

где се други (пребаци диск са 1 на 3) решава једноставно, а први и трећи су исте природе као полазни – само је број дискова за један мањи и разликују се стубови *са* којих и *на* које се врши пребацивање. Дакле, овде се рекурзија намеће као природно решење, па настављајући истим поступком решавање добијених потпроблема број дискова се смањује до тривијалног случаја. Решивши га, а затим се враћајући у тачке позива решавају се потпроблеми од простијих (са мањим бројем дискова) ка сложенијим.

За конструисање општег алгоритма потребно је указати на стуб који се користи као помоћни:

$$\text{prebaci}(n,\text{sa},\text{na},\text{pom}) \leftrightarrow \begin{cases} \text{prebaci}(n-1,\text{sa},\text{pom},\text{na}) \\ \text{sa} \rightarrow \text{na} \\ \text{prebaci}(n-1,\text{pom},\text{na},\text{sa}) \end{cases}$$

На основу ове рекурзивне дефиниције може се саставити одговарајућа функција:

```
#include <stdio.h>
void prebaci(int n,int sa,int na,int pom) /* Hanojska kula */
  {
  if (n>0)    /* ili (n) */
    {
    prebaci(n-1,sa,pom,na);
    printf("%d -> %d\ n",sa,na);
    prebaci(n-1,pom,na,sa);
    }
  }
void main()
  {
  int broj_diskova;
  printf("Unesi broj diskova:");scanf("%d",&broj_diskova);
  prebaci(broj_diskova,1,3,2);
  }
```

Индуктивно се лако показује да је број корака потребан да се пребаци n дискова са једног стуба на други $2^n - 1$. Према томе, разлога за бригу нема, јер – ако бисмо претпоставили да су брамански свештеници, играјући са 64 златна диска, у стању да у секунди пребаци један диск са стуба на стуб – крај света није тако близу (у игри су милијарде година).

Пример 6. Са тастатуре се учитава формула (без грешке) следећег облика:

```
<formula>:=<cifra>|(<formula><znak><formula>)
<znak>::=+|-|*
<cifra>::=0|1|2|3|4|5|6|7|8|9
```

Написати програм који израчунава вредност учитаног израза. На пример,

`5 -> 5, ((2-4)*6) -> -12.`

Пошто је опис формуле рекурзивне природе, то се проблем најједноставније решава коришћењем рекурзивне функције која прати наведени синтаксни опис:

```
#include <stdio.h>
int f()
  {
  char c,op;
  int x,y,s;
  c=getchar();
  if ('0'<=c && c<='9') return (c-'0');
  // c je otvorena zagrada
  x=f();
  op=getchar(); // cita operacijski znak
  y=f();
  switch (op)
    {
    case '+':s=x+y;break;
    case '-':s=x-y;break;
    case '*':s=x*y;break;
    }
  c=getchar(); // cita zatvorenu zagradu
  return (s);
  }
void main()
  {
  printf("%d",f());
  }
```

Пример 7. *Pick-up-stones* (узми каменчић). На гомили се налази n каменчића. Играч који је на потезу са гомиле узима 1, 2 или 3 каменчића. Победник је играч који последњи одигра, тј. покупи преостале каменчиће.

а) Написати функцију $f(i)$ која на основу i – тренутни број каменчића, враћа:

0 – ако је позиција *губитничка*;

k – ако је позиција *добитничка*, где је k број каменчића које играч у добитничкој позицији треба да узме да би противника довео у губитничку позицију.

Позиција је губитничка за играча који је на потезу, ако више нема каменчића, или шта год да одигра позицију преводи у добитничку за играча који треба да одигра следећи потез.

Позиција је добитничка за играча који је на потезу, ако је могуће да узимањем одређеног броја каменчића позицију преведе у губитничку за играча који треба да одигра следећи потез.

б) Написати програм који симулира игру човека против рачунара, дајући предност човеку тиме што му омогућава да одигра први потез.

Решење. а) Према опису услова који детерминишу губитничку, односно добитничку позицију може се написати рекурентна формула:

$$f(i) = \begin{cases} 0, & i = 0, \\ i, & i \in \{1,2,3\}, \\ k, & (\exists k \in \{1,2,3\})\, f(i-k) = 0, \\ 0, & (\forall k \in \{1,2,3\})\, f(i-k) \neq 0, \end{cases}$$

по којој ћемо креирати тражену рекурзивну функцију:

```
int f(int i)
  {
  if (i==0) return 0;     // gubitnicka pozicija
  if (i==1 || i==2 || i==3) return i; // dobitnicka pozicija
  if (f(i-1)==0) return 1;  // dobitnicka pozicija, treba uzeti 1
  if (f(i-2)==0) return 2;  // dobitnicka pozicija, treba uzeti 2
  if (f(i-3)==0) return 3;  // dobitnicka pozicija, treba uzeti 3
  // nijedan od pokusaja dovodjenja na gubitnicku poziciju nije uspeo
  return 0;
  }
```

б)

```
#include <iostream.h>
int f(int i)
  {
  Videti pod a)
  }
void main()
  {
  int n,k,p;
  cout << "Unesi broj kamencica -->"; cin >> n;
  do
    {
    cout << "Na gomili je " << n << " kamencica, koliko uzimas?  ";
    cin >> k;
    n-=k;
```

```
    if (n==0)
      cout << "Bravo pobedio si !";
    else
      if (f(n)==0)
        {// gubitnicka pozicija za racunar, pa uzima po 1, nadajuci se gresci igraca
        cout << "Od preostalih " << n << " uzimam 1!" << endl;
        n--;
        }
      else
        {// dobitnicka pozicija za racunar
        p=f(n); //uzimajuci p prevodi poziciju u gubitnicku za protivnika
        cout << "Od preostalih " << n << " uzimam " << p << endl;
        n=n-p;
        if (n==0) cout << "Zao mi je izgubio si!" << endl;
        }
    }
  while (n!=0);
  }
```

ОБАВЕШТЕЊЕ

МИНИСТАРСТВО ПРОСВЕТЕ И СПОРТА РЕПУБЛИКЕ СРБИЈЕ
ДРУШТВО МАТЕМАТИЧАРА СРБИЈЕ

РЕПУБЛИЧКИ СЕМИНАР 2003.

о настави математике и рачунарства
у основној школи, у средњим школама,
на вишим школама и на факултетима

Обавештавамо вас да ће се Републички семинар 2003. који организују Министарство просвете и спорта Србије и Друштво математичара Србије одржати 14. и 15. фебруара 2003. године на Електронском факултету у Нишу, Београдска 14, и биће посвећен актуелним питањима наставе математике и рачунарства.

Семинар ће радити у четири секције:

I секција – настава математике у старијим разредима основне школе;

II секција – настава математике у средњим школама;

III секција – настава математике на вишим школама и факултетима;

IV секција – настава рачунарства.

Семинар почиње пленарним састанком 14.02.2003. године у 10.30 часова у Великој сали Дома Војске Југославије у Нишу, Синђелићев трг.

У оквиру пленарног састанка биће одржан Округли сто на тему: **Да ли је $9+3=8+4$?**

Министар просвете и спорта Републике Србије
проф. др Гашо Кнежевић, с.р.
Председник Друштва математичара Србије
проф. др Раде Дорословачки, с.р.