Алгоритми

Рачунар је **програмабилна машина**.
У разним научним дисциплинама, али и у обичном животу често проблеме решавамо пратећи прецизно дефинисане **поступке**. На пример, док кувамо или правимо колаче обично пратимо рецепт који описује поступак који треба спровести да бисмо припремили укусно јело. У физици и математици користимо велики број формула. Често кажемо да су задаци који се могу решити праћењем неког поступка „шаблонски" и да онај ко научи шаблон тј. поступак решавања, тај може да реши било који задатак тог типа. Уместо речи поступак тј. шаблон, у математици и рачунарству се најчешће користи реч **алгоритам**.

У 9. веку арапски математичар Абу Абдулах Мухамед ибн Муса ал Хорезми (обично се памти само **Ал Хорезми**) написао је неколико књига о математици. Једна је говорила о поступцима за решавање једначина и из њеног латинизованог дела наслова „ал џабр" потекла је данашња реч алгебра. Друга књига, је у Европи постала позната под латинским називом Algoritmi de Numero Indorum, што је требало да значи „Ал Хорезми, о индијским бројевима". Међутим, заборавило се да је Algoritmi име аутора и усталио се превод „Поступци рачунања индијским бројевима". Од тада реч алгоритам означава произвољан, обично математички поступак и одомаћила се у области рачунарства.	 Ал Хорезми

Рачунари **нису** интелигентне машине. Једино у чему су рачунари добри је то што могу јако **брзо** да спроводе **прецизно описане поступке**. Додатно, поступци у рачунару увек говоре како се барата са бројевима (обично бинарно записаним) – подсетимо се, рачунари су дигитални и сви садржаји којима приступамо кодирани су помоћу бројева (градиво првог разреда). Самим тим, рачунарски алгоритми су по својој природи математички. **Да би рачунар могао да спроведе неки алгоритам, тај алгоритам мора бити описан на веома прецизном програмском језику, у облику рачунарског програма.**

Појам алгоритма

Неформално, можемо рећи да алгоритам дефинише **низ прецизно описаних елементарних корака чијим се доследним спровођењем долази до решења неког проблема**.
Наведимо пример алгоритма из свакодневног живота – **Рецепт за палачинке**:

1. Узми дубљу посуду!
2. Сипај у њу 200g брашна и 2 јаја!
3. Ако имаш млека, сипај 2 dl млека, у супротном сипај 2dl воде.
4. Мути све док смеса не буде без грудвица.
5. Додај 1dl уља.
6. Промешај!

- *„шпагети" програми*

Својства алгоритама

коначан, зауставља се, ефикасан, детерминистички

Сваки опис алгоритма мора да буде **коначан** тј. алгоритам мора бити описан помоћу коначног броја корака. Међутим то не гарантује да ће и извршавање алгоритма бити коначно тј. да ће се оно зауставити за било коју вредност улазних

параметара. На пример, алгоритам:

„Док је x различито од нуле, увећај y за 1 и смањуј x за 1, као резултат прикажи y.“

(x=3, y=5 и x=-3, y=4)

Пожељно је да се сваки алгоритам **зауставља** и враћа резултат за било које улазне вредности. То није увек лако остварити, а само питање испитивања заустављања датог алгоритма је комплексно математичко питање.

Да би алгоритам био практично употребљив он мора бити **ефикасан**, што подразумева да се може спровести у времену које је прихватљиво за корисника.

Да би рачунари могли да спроведу неки алгоритам он треба да буде **детерминистички**. То значи да сваки корак мора да буде такав да се једнозначно зна како се он извршава и шта се дешава након извршавања тог корака. Корак:

„Нека је x неки број између 1 и 10!”

би у алгоритам увео недетерминизам у алгоритам, јер се не би знало једнозначно коју вредност ће имати x након извршавања тог корака и како ће се извршавати наредни кораци чије понашање зависи од вредности. Рачунари су детерминистичке машине и они нису у стању да извршавају недетерминистичке кораке тј. недетерминистичке алгоритме.

Начини описивања алгоритама

Постоје разни начини да се алгоритми опишу. Основну поделу начина описа алгоритама начинићемо на основу тога ко тај алгоритам треба да спроведе, тј. да ли алгоритам описујете човеку који на основу вашег описа треба га да разуме и усвоји или машини (тј. рачунару) која тај алгоритам треба потпуно аутоматски да спроведе. Сваки опис алгоритма мора бити довољно прецизан да би онај ко треба да га спроведе могао да га спроведе.

Природно-језички описи

Алгоритми су у обичном животу намењени људима и зато се описују на природном језику (као што је то био случај са рецептом за палачинке). И многи математички алгоритми су такви да их обично спроводе људи, а не машине и зато се и они често описују на природном језику.
Када се алгоритми описују људима, описи су често релативно непрецизни и ослањају се на то да је публика довољно интелигентна да „попуни рупе”, тј. да разјасни себи оно што није баш потпуно прецизно описано и да евентуалне грешке у описима примети и исправи.

Алгоритми се често „описују” само тако што се наведе низ примера који илуструју њихов рад (на пример, учитељица ученике не учи сабирању бројева тако што им да опис алгоритма, већ тако што им покаже велики број примера сабирања). Иако су људима овакви описи блиски (јер су веома конкретни), они често подразумевају да је читалац прилично интелигентан и да ће на основу примера самостално успети да апстрахује, прецизира и усвоји поступак. Дешава се често да људи умеју исправно да спроведу неки поступак, али да не умеју да дају његов прецизан опис, па чак ни на природном језику. На пример, сви ученици умеју да сабирају декадно-записане бројеве, али када се од њих тражи да тај алгоритам опишу прецизно на говорном језику, само мали број ученика уме то да уради.

Много бољи од описа алгоритама само кроз примере су експлицитни описи агоритма на природном језику. На пример, такав је наредни опис алгоритма којим се одређује највећи од 5 бројева које нам неко говори.

Пошто је природни језик обично непрецизан, и овакви описи су често непрецизни и такође подразумевају одређену интелигенцију читаоца (самим тим, не долазе у обзир као облик саопштавања алгоритма рачунару).

Описи у облику псеудо-кода

Уобичајен начин описа алгоритама данас представљају опис у облику **псеудо-кода**. Такви описи представљају компромис између флексибилности коју дају неформални, природно-језички описи и потпуно прецизних описа алгоритама које рачунар може да извршава Наредни псеудо-кôд описује алгоритам за одређивање највећег броја и он је доста близак програмским кодовима које ћемо сретати током овог курса.

```
broj := unesi_broj
maksimum := broj
ponovi sledeće četiri puta
    broj := unesi_broj
    ako je broj > maksimum onda
          maksimum := broj
saopšti maksimum
```

Већ код описа овог алгоритма на природном језику било је јасно да је за његово спровођење потребно да се нешто запамти (меморише, запише). Ако је потребно да се запамти више података, пожељно је да сваки тај податак има јединствено име. За то се користе **променљиве** и скоро сви алгоритми са којима ћете се срести у рачунарству користе променљиве. У претходном псеудокоду користе се две променљиве (`maksimum` и `broj`).

На једном примеру разјаснимо наведени алгоритам и његов опис у псеудокоду. Претпоставимо да редом добијамо бројеве 1, 3, 2, 4 и 1.

- На почетку извршавања алгоритма наредбом `maksimum := unesi_broj` променљива `maksimum` поставља се на вредност 1 (први број који нам је саопштен).
- Након тога извршавају се следећа четири корака:
  - Наредбом `broj := unesi_broj` променљива `broj` поставља се на вредност 3 (други број). Пошто је то веће од текуће вредности променљиве `maksimum` (која је 1), наредбом `maksimum := broj` променљива `maksimum` се ажурира на вредност 3.
  - Наредбом `broj := unesi_broj` променљива `broj` поставља се на вредност 2 (трећи број). Пошто то није веће од текуће вредности променљиве `maksimum` (која је 3), променљива `maksimum` задржава своју вредност.
  - Наредбом `broj := unesi_broj` променљива `broj` поставља се на вредност 4 (четврти број). Пошто је то веће од текуће вредности променљиве `maksimum` (која је 3), наредбом `maksimum := broj` променљива `maksimum` се ажурира на вредност 4.
  - Наредбом `broj := unesi_broj` променљива `broj` се поставља на вредност 1 (пети број). Пошто то није веће од текуће вредности променљиве `maksimum` (која је 4), променљива `maksimum` задржава своју вредност.
- На крају се наредбом `saopšti maksimum` саопштава текућа вредност променљиве `maksimum`, која је 4, што заиста јесте највећи од добијених бројева.

Наредна табела приказује како се вредност променљивих мењала током спровођења овог алгоритма:

Broj	maksimum
-	1
3	3
2	3
4	4
1	4

Приметимо да псеудокод комбинује обичан, говорни језик и прецизну, математичку симболичку нотацију. Нагласимо и то да није усаглашено како псеудокод треба да изгледа, већ сваки аутор псеудокода бира језичке и симболичке конструкције које сматра одговарајућим (на пример, у наведеном псеудокоду претпоставили смо да се додела вредности променљивој означава симболом `:=`, да је језик за опис наредби српски, да се на основу увлачења линија кода (тзв. назубљивања) одређује које се наредбе понављају одређени број пута итд.).

Као и код описа у природном језику, псеудокод се састоји од појединачних наредби и веома је важно да се одреди које облике наредби разуме онај ко треба да спроведе тај алгоритам. У наведеном примеру коришћена је додела вредности

променљивој (тзв. наредба доделе) означена са `:=`, затим наредба гранања означена са `ako je ... onda` и петља `ponovi sledeće ... puta`. Тај облик петље не постоји у свим програмским језицима. Да би на основу оваквог псеудокода у таквим језицима могао да се напише прави програмски кôд, потребно је да се алгоритам преформулише. Користићемо петљу облика `dok je ... radi sledeće`, која је присутна у већини правих програмских језика, а тако написан псеудокод лакше се може пребацити у прави кôд него претходни. У овој верзији алгоритма користи се помоћна променљива `i` којом се броји колико је бројева обрађено до тог тренутка (њена вредност се поставља на 1 приликом учитавања првог броја и увећава се за 1 након читања сваког следећег броја). Петља престаје са извршавањем када вредност променљиве `i` достигне 5 (што значи да је обрађено 5 бројева). На основу претходног разматрања, алгоритам може да се формулише на следећи начин:

```
broj := unesi_broj
maksimum := broj
i := 1
dok je i < 5 radi sledeće
    broj := unesi_broj
    ako je broj > maksimum onda
          maksimum := broj
    i := i + 1
saopšti maksimum
```

Приликом описа сваке процедуре важно је питање који се елементарни кораци могу користити (на пример, претпоставили смо да постоји корак `unesi_broj` којим се добија следећи број, да постоји корак `saopšti` којом се саопштава неки резултат, да онај ко спроводи алгоритам уме да упореди два броја, што је означено симболом > итд.). Псеудокод оставља могућност ономе ко пише алгоритам да користи елементарне кораке које сматра адекватним. Обично се сматра да сиромашан скуп елементарних операција доводи до предугачког описа алгоритма и тешког разумевања основне идеје. С друге стране, ако има много операција које аутор алгоритма сматра елементарним, а које су комплексне за онога ко тај алгоритам касније треба да спроведе, тада је потребан додатни труд да се такав псеудокод трансформише у довољно прецизан опис који човек или рачунар може извршавати. На пример, у неком сложенијем алгоритму може се сматрати да је корак „нађи највећи од пет бројева” елементаран, али смо видели да је за његово извршавање потребно да се разради.
Разумевање описа датих у облику псеудокода (а касније и у облику програмског кода) важна је вештина и предуслов је за програмирање. Као вежбу, прикажимо још један пример алгоритма и покушајмо да закључимо шта тај алгоритам израчунава.

```
s := 1, i := 0
dok je i < n radi sledeće:
   s := s·x
   uvećaj i za 1
ispiši s
```

Овако описан алгоритам може да се спроведе корак по корак. Међутим, приметимо да променљиве `x` и `n` немају задату почетну вредност. Њих сматрамо улазним параметрима алгоритма и алгоритам се може спровести за разне вредности тих параметара. На пример, ако је `x` = 5 и `n` = 3, током спровођења алгоритма у тачки провере услова променљиве имају следеће вредности:

x	n	s	i
5	3	1	0
5	3	5	1
5	3	25	2
5	3	125	3

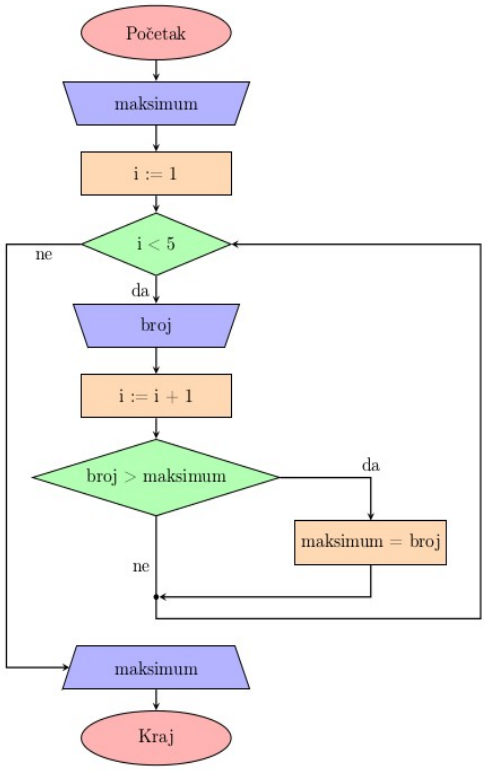
На крају извршавања алгоритма променљива `s` садржи вредности 5^3, тј. вредност x^n. Заиста, кренувши од вредности 1,

променљива `s` множи се `n` пута вредношћу променљиве `x`, тако да након извршавања петље садржи вредност x^n. Прецизније се може рећи да пре петље, током извршавања петље и након петље важи услов да је $s = x^i$, јер је након `i` корака вредност променљиве `s` једнака вредности почетној вредности `1`, која је `i` пута помножена вредношћу променљиве `x` (ако посматрамо вредности променљивих `s`, `x` и `i` током извршавања програма, примећујемо да је $1 = 5^0$, $5 = 5^1$, $25 = 5^2$, $125 = 5^3$). Пошто након завршетка петље важи да је $i = n$, следи да је $s = x^n$. Дакле, овим алгоритмом врши се степеновање броја `x` на ненегативан степен `n`.

Дијаграми тока програма

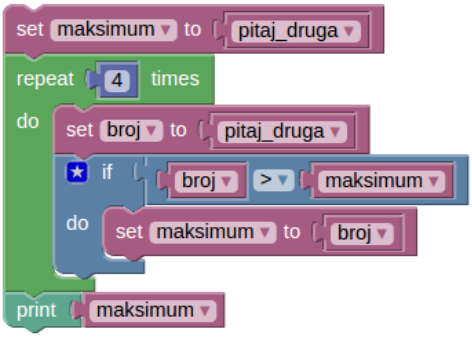
У другој половини 20. века веома популаран начин за опис алгоритама били су такозвани **дијаграми тока програма** (енгл. *control-flow graphs*, *flowcharts*) тј. **дијаграми кода** (енгл. *code-charts*). Иако се данас не користе као раније, ови дијаграми су важан део историје рачунарства и инспирисали су многе дијаграмске технике које се и данас примењују. Дијаграми тока програма спадају у **блок-дијаграме** јер се алгоритам описује дијаграмом који се састоји од појединачних блокова (представљених правоугаоницима, трапезима, елипсама итд.) повезаних стрелицама. Иако дијаграми тока нису јединствено стандардизовани, постојала су нека општа правила како се пишу. Тако се, на пример, трапез оријентисан наниже користио за опис улазних података, трапез оријентисан навише за опис резултата израчунавања, правоугаоници су се користили за опис израчунавања (обично су обухватали математичке формуле које описују како се нове вредности израчунавају на основу постојећих), ромбови су се користили да опишу гранања у програмима (питања на основу чијег се одговора одређује путања којом извршавање алгоритма треба да се настави) итд. Линије означене стрелицама означавају наредни блок који треба да буде извршен. Пример алгоритма за проналажење максимума понуђених бројева могао би да се опише блок-дијаграмом приказаним на слици.

Мана дијаграма тока програма је у томе што омогућавају пренос тока у произвољну тачку програма (то инспирише употребу наредбе скока тј. GOTO), чиме настају „шпагети” програми, које је тешко анализирати и који често крију неке грешке.



Дијаграми MIT Scratch/Blockly

Још једна веома популарна дијаграмска техника програмирања – пре свега у образовању младих програмера – појавила се 2006. у систему за учење програмирања Scratch развијеном на универзитету MIT у САД. Такође популаран пројекат An hour of code (http://code.org/), који се бави популаризацијом програмирања, користи веома сличан дијаграмски језик. Дијаграмске описе омогућава библиотека Blockly компаније Google. Алгоритми се описују слагањем блокова (попут лего коцкица) и обично су такви да описују кретање неке фигуре (цртаног јунака) по екрану. Већи блокови, који обухватају мање, обично се користе за опис понављања корака у мањим блоковима. Наш пример алгоритма за проналажење максимума могао би се описати дијаграмом приказаним на слици.



Програми – потпуно прецизни описи алгоритама

Сви начини за описивање алгоритма који су да сада поменути (осим MIT Scratch/Blockly дијаграма) неформални су, тако да рачунар не може да их спроведе. Да би се неки алгоритам спровео, он мора да буде описан на **програмском језику**. За такве описе користи се термин **рачунарски програм** или **програм**. Процес записивања алгоритма у програмском језику назива се **кодирање**, **имплементација** или **програмирање** (мада се процес програмирања често схвата шире, тако да обухвата и осмишљавање и имплементацију алгоритама). Програмски језици су вештачки и прецизно описани јер свака језичка конструкција има прецизну структуру и једнозначно значење које јој се придружује.

Током кратке али интензивне историје дигиталних електронских рачунара развијен је велики број разнородних програмских језика. Приликом осмишљавања програмских језика прави се баланс између два захтева. Први захтев је тај да језик буде довољно прецизан како би рачунар могао да спроведе алгоритам на основу описа (изврши тражено израчунавање). Други захтев је да је опис што апстрактнији (на пример, да скуп елементарних операција које се могу јавити у опису алгоритма буде што богатији) како би програмер могао да опише своје идеје што једноставније. Током своје историје програмски језици су се померали од првог ка другом наведеном захтеву. Важан корак ка томе била је идеја да програмер не мора да опише алгоритам на нивоу елементарних инструкција које рачунар може да разуме, већ да свој опис зада апстрактније на неком **вишем програмском језику**, а да се затим његов опис, коришћењем специјализованих програма (тзв. **програмских преводилаца**), преводи на кôд на **машинском језику**, који рачунар може директно да извршава. Више речи о овоме биће у посебном поглављу, посвећеном управо програмским језицима. Да бисмо вам приказали како изгледају описи алгоритма у правим програмским језицима, прикажимо како се алгоритам проналажења максимума пет бројева може изразити у језику C и у језику Pascal (оба ова језика спадају у више програмске језике).

```c
/* Kôd u jeziku C */
int i, maksimum, broj;
scanf("%d", &broj);
maksimum = broj;
for (i = 1; i < 5; i++) {
   int broj;
   scanf("%d", &broj);
   if (broj > maksimum)
```

```pascal
{ Kôd u jeziku Pascal }
var i, maksimum, broj: Integer;
begin
  ReadLn(broj);
  maksimum := broj;
  for i := 1 to 4 do
  begin
     ReadLn(broj);
```

```
      maksimum = broj;
}
printf("%d\n", maksimum);
```

```
      if broj > maksimum then
          maksimum := broj
    end;
    WriteLn(maksimum)
end;
```

Можете приметити да структура оба програма прилично одговара датом псеудокоду, али су унети додатни детаљи који су неопходни за аутоматско спровођење на рачунару, а специфични су за сваки од ових програмских језика.