1.
Rješavanje zadataka računalom i programski jezik Pascal

1. Postupak pripremanja programa

Naučili smo već da računalo, sastavljeno od procesora, spremnika, ulazno-izlaznih naprava i njihovih pristupnih sklopova uz pomoć operacijskog sustava — skupine programa koji omogućuju djelovanje sklopovlja — može obavljati vrlo raznovrsne zadatke. Za svaki od zadataka koji želimo rješavati treba pripremiti program u nekom višem programskom jeziku i zatim ga prevesti u strojni oblik kako bi se on mogao izvoditi u računalu.

Neke su primjene računala tako česte da se za njihovo ostvarenje pripremaju programi ili skupine programa koji se mogu kupiti gotovi. Programi se obično kupuju na nekom spremničkom mediju (disketi, optičkom disku) s podrobnim uputama o načinu njihova korištenja. Kupljene programe je najprikladnije pohraniti na disk računala u obliku jedne ili više datoteka. Ti gotovi programi se vrlo jednostavno pokreću tako da se odgovarajućom naredbom operacijskog sustava "pozovu" s diska i premjeste u radni spremnik računala, nakon čega ih procesor počinje izvoditi. Korisnik programa, u skladu s uputama za korištenje programa, može bez većih poteškoća i bez posebnih znanja o detaljima djelovanja računala obavljati svoj posao. Upute za korištenje nekih složenijih programa mogu biti vrlo opširne i zauzimaju i po nekoliko knjiga. U današnje vrijeme se upute također pohranjuju na računalu i mogu se jednostavno pregledavati.

Prisjetimo se da se *svi programi* i *svi podaci* u računalu trajno čuvaju u obliku datoteka u vanjskim spremnicima. Stoga je pri prvom upoznavanju računala i njegova operacijskog sustava važno naučiti osnovne operacije s datotekama. Mnoge računalne aktivnosti svode se na premještanje, mijenjanje i pohranjivanje datoteka. Tako se i svaki program pokreće iz neke datoteke.

Međutim, postavlja se pitanje kako je nastao neki program koji mi tako jednostavno koristimo. Netko je morao program dovesti u oblik prikladan za pokretanje na računalu.

Pripremanje novih programa

Mi ćemo se uporabom ovog udžbenika upoznati s načinima pripremanja novih programa. Bit će to, prirodno, programi za rješavanje jednostavnih zadataka. Međutim, svi veliki i složeni programi sastoje se od mnoštva "malih programa" koji međusobno surađuju u rješavanju nekog složenog zadatka. To je upravo i osnovno načelo pri rješavanju složenih zadataka: svaki veliki zadatak treba *raščlaniti* na manje *podzadatke* koji se mogu lakše savladati i zatim njihovim objedinjavanjem dosegnuti rješenje velikog zadatka.

Tijekom vremena pokazalo se da se neki podzadaci pojavljuju vrlo često u rješavanju mnogih zadataka, tako da se oni mogu samo *jedanput prirediti* i zatim koristiti u *mnogim primjenama*. Za takve podzadatke treba odabrati najbolji mogući i znani način ostvarenja. Primjeri jednostavnih programa koji će se izučavati u većini su slučajeva takvi da, osim što pokazuju mogućnosti uporabe programskog jezika, mogu poslužiti kao gradivni elementi složenijih programa.

Mi ćemo za pisanje programa upotrebljavati programski jezik *Pascal*. Pokazalo se da je taj programski jezik prikladan za prvo učenje programiranja, jer je tu svrhu prvenstveno imao na umu njegov autor[1]. Međutim, težište zanimanja neće nam biti izučavanje svih detalja jezika već ćemo naglašavati one njegove izražajne mogućnosti koje dolaze do izražaja pri rješavanju određene klase zadataka.

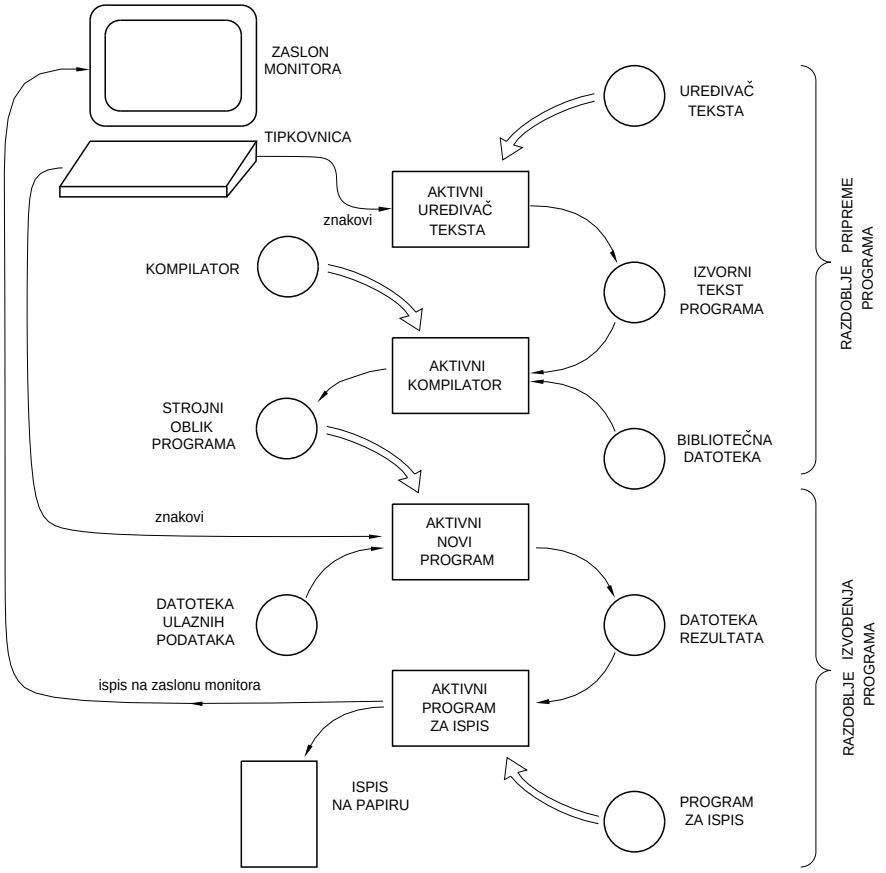
Za djelotvorni rad pri pripremi (ili razvoju) programa koristit ćemo neke gotove programe — možemo ih nazvati programskim pomagalima. Skup takvih programskih pomagala čini tzv. *radno* ili *razvojno okruženje*[2].

Postupak pripreme jednog programa i njegova korištenja prikazan je na slici 1.1. Krugovi na slici predstavljaju datoteke, a pravokutnici aktivne programe nastale iz datoteka. Aktiviranje datoteke u aktivni program simbolički je prikazano podebljanom strelicom koja pokazuje od kruga na pravokutnik.

Aktivni programi prihvaćaju ili pojedinačne znakove s tipkovnice ili sadržaje nekih datoteka kao svoje ulaze i proizvode izlazne datoteke. Ulazi i izlazi programa prikazani su kao jednostavne strelice koje pokazuju prema pravokutniku odnosno od pravokutnika.

[1] Autor jezika Pascal je Niklaus Wirth, profesor računarstva na Visokoj tehničkoj školi u Zürichu, poznatoj po kratici ETH (od njem. Eidgenosische Technische Hochschule), na kojoj su djelovali i naši nobelovci kemičari Lavoslav Ružička i Vladimir Prelog

[2] U današnje vrijeme je, po svemu sudeći, na osobnim računalima najrasprostranjenije radno okruženje američke tvrtke Borland. Tvrtka je svoje radno okruženje nazvala Integrated Development Environment (IDE) — cjelovito radno okruženje, a svoju inačicu jezika Turbo Pascal. U udžbeniku će se zbog rasprostranjenosti Turbo Pascala, onda kada je to potrebno činiti, navoditi posebnosti te inačice. Oni koji koriste neki drugu inačicu Pascala moraju stoga pažljivo proučiti priručnik te inačice.



Sl. 1.1. Postupak pripreme i korištenja programa

Tijek pripreme i izvođenja nekog programa može se podijeliti u dva razdoblja:

- razdoblje pripremanja programa[3], i
- razdoblje izvođenja programa[4]

Razdoblje pripreme programa

U razdoblju pripreme programa posao se obavlja u dva osnovna koraka:

- uređivanje teksta programa, i
- prevođenje (kompiliranje) izvornog teksta programa u strojni oblik.

[3] u engleskom se jeziku razdoblje pripreme naziva razdobljem kompiliranja — *compile-time*

[4] engleski: *run-time*

Postupak uređivanja teksta, tj. upisivanje programa u računalo potpomaže *uređivač teksta*[5].

U tu se svrhu aktivira program uređivača teksta. On prihvaća znakove s tipkovnice i pohranjuje ih postepeno u datoteku izvornog programa. Istovremeno se upisani tekst vidi i na zaslonu monitora. Uređivač teksta je posebno prilagođen za pisanje programa i potpomaže skladno oblikovanje teksta programa, te čak omogućuje isticanje ključnih riječi i potpomaže automatsko uvlačenje redaka.

Nakon što smo završili upisivanje programa u računalo i stvorili datoteku *izvornog teksta* programa možemo tu datoteku, uz odabiranje prikladnog imena, trajno pohraniti na disk ili disketu. Iako još ne znamo da li je upisani program dobar, to je razumno učiniti kako se, baš zbog neispravnosti našeg programa, u daljnjem njegovom ispitivanju ne bi dogodilo da nam trud upisivanja propadne. Kasnije ćemo lako prvotno pohranjeni tekst zamijeniti s krajnjim ispravnim tekstom programa. Uredivač teksta ima komande kojima se pohranjivanje jednostavno obavlja.

Nakon toga se iz svoje datoteke aktivira *kompilator*. On prihvaća datoteku izvornog teksta kao svoju ulaznu znakovnu datoteku i, uz dodatno korištenje bibliotečnih datoteka iz kojih dobavlja neke unaprijed pripremljene funkcije i procedure, izgrađuje *strojni program* ili *izvedbeni program*. Ako je prevođenje uspješno obavljeno, strojni program se pohranjuje u *datoteku programa pripremljenog za izvođenje*[6].

Razdoblje izvođenja programa

Razdoblje izvođenja programa započinje aktiviranjem strojnog programa. Novostvoreni aktivni strojni program program prihvaća *ulazne podatke* koje unosi čovjek s *pomoću tipkovnice* ili iz unaprijed pripremljene *ulazne datoteke*. Program izračunava rezultate i pohranjuje ih u *datoteku rezultata* ili *izlaznu datoteku*, onako kako je to određeno izlaznim instrukcijama programa. U odgovarajućem radnom okruženju može se sadržaj izlazne datoteke promatrati tijekom nastanka na zaslonu monitora. Može se smatrati da je zaslon terminala i datoteka u koju se pohranjuju razultati. Tu datoteku može promatrati čovjek i pamtiti njezin sadržaj. Datoteka će nakon toga biti “pohranjena u mozgu čovjeka”.

U neku se ruku može unošenje ulaznih podataka preko tipkovnice smatrati prihvaćanjem podataka iz neke datoteke. Sadržaj te ulazne datoteke određuje čovjek koji sjedi uz računalo. On tipka znak po znak podatke koje sam smišlja u trenutku tipkanja, doziva ih iz pamćenja ili ih čita s papira. Ulazna datoteka je prema tome ili “pohranjena u mozgu čovjeka” ili zapisana na papiru.

[5] engleski: *text editor*, ili kraće samo: *editor*

[6] U engleskom se jeziku izvorni tekst programa naziva *source program* ili *source code* — izvorni kôd, a strojni oblik programa naziva se *machine program* ili *machine code* odnosno *executable program* ili *executable code*. U Borlandovu IDE okruženju datoteke izvornih programa imaju dodatak .PAS, a kompilator nakon uspješnog prevođenja stvara datoteku izvedbenog programa s dodatkom .EXE, pa će program pisan u Pascalu i pohranjen u datoteku IME.PAS imati pridruženi izvedbeni program pohranjen u datoteku IME.EXE.

U Pascalu su stoga definirane dvije osnovne (standardne) znakovne datoteke — jedna ulazna i druga izlazna — koje se automatski pridružuju svakom programu. Ulazna datoteka nosi naziv *Input*[7] i pretpostavlja se da se njezin sadržaj unosi s tipkovnice. Osnovna izlazna datoteka nosi naziv *Output*[8] i pridružena je zaslonu monitora. Podaci koje vidimo na zaslonu možemo prepisati rukom i tako trajno pohraniti izlaznu datoteku na papiru[9].

Naučit ćemo kasnije kako se mogu stvarati, imenovati i koristiti druge nestandardne datoteke, kako bi se rezultati izračunavanja mogli trajnije sačuvati u računalu na nekom vanjskom spremniku.

Na kraju, nakon izvođenja programa može se izlazna datoteka koja je trajno pohranjena, propustiti kroz neki aktivirani program za pisanje na papir koji će tu datoteku prihvatiti kao svoju ulaznu datoteku i s pomoću pisača stvoriti pisani dokument — "datoteku na papiru".

Ispitivanje programa

U opisanom postupku pripremanja programa pretpostavili smo da uvijek pišemo ispravne programe i nikada ne činimo nikakve pogreške. U stvarnosti se događa upravo suprotno. Rijetko ćemo koji program pripremiti bez i jedne pogreške[10]. Stoga je jako važno da radno okruženje za pripremu programa sadrži i programska pomagala za ispitivanje programa.

Postoje tri glavne skupine mogućih pogrešaka. To su:

- pogreške koje mogu biti prepoznate *tijekom prevođenja*, tj. u razdoblju pripreme programa[11];
- pogreške koje se ustanovljuju u razdoblju *izvođenja programa*[12];
- pogreške koje se ne mogu ustanoviti niti tijekom prevođenja niti tijekom izvođenja programa.

Kažimo nekoliko riječi o svakoj od tih vrsti pogrešaka.

Glavnina pogrešaka *prve vrste* koje se ustanovljuju tijekom prevođenja su *sintaksne*[13] pogreške. Kada kompilator tijekom prevođenja naiđe na pogrešku on obustavlja daljnje prevođenje, ponovno se aktivira uređivač teksta, dio izvornog teksta programa u kojem je ustanovljena pogreška pokaže se na zaslonu i značka

[7] od engleskog *input* — ulaz

[8] od engleskog *output* — izlaz

[9] Mi sve svoje bilješke, pismene sastavke, matematičke zadatke i njihova rješenja zapisujemo rukom na stranice svojih bilježnica u obliku "znakovnih datoteka".

[10] Engleski je naziv za pogrešku *error*. U računalnom žargonu koristi se i naziv *bug* — insekt, stjenica, ali i bilo koji mikroorganizam koji izaziva bolest (bakterija ili virus). Za postupak pronalaženja i odstranjivanja pogrešaka iz programa koristi se stoga naziv *debugging*.

[11] engleski: *compile-time errors*

[12] engleski: *run-time errors*

[13] Sintaksa je dio gramatike koja izučava pravila gradnje rečenice i njezinih dijelova, te uporabu vrsta riječi i njihovih padežnih odnosno glagolskih oblika. U programskim jezicima propisana je stroga sintaksa — moglo bi se reći: pravila pisanja programa — čije zadovoljenje kompilator može provjeravati.

se postavlja na onaj redak izvornog teksta u kojem je ustanovljena pogreška. Na zaslonu se ispisuje broj tipa pogreške i, kada je to moguće, ispisuje se i dodatna informacija koja olakšava ispravljanje pogreške. Nakon ispravljanja dojavljene pogreške ponovno se mora pokrenuti kompilator i na taj način se postepenim ispravljanjem dolazi do ispravnog programa, kojeg kompilator konačno pretvara u strojni program.

Druga vrsta pogrešaka, koje nastaju tijekom izvođenja programa, nastaje zbog različitih razloga. Tako, primjerice, program može pokušati čitati odnosno pisati u datoteku koja ne postoji ili nije pripremljena za čitanje ili pisanje. Može se, nadalje, dogoditi da program mora izračunati drugi korijen iz broja kojeg dohvaća iz ulazne datoteke a u ulaznoj datoteci pronalazi negativni broj, ili da program mora obaviti dijeljenje s nekim brojem a iz ulazne datoteke dohvati broj nula. Kada se ustanovi takva pogreška, obustavlja se daljnje izvođenje programa, javlja se tip pogreške i, ako je to moguće, opis uzroka pogreške. Programer mora pažljivim promišljanjem ili ponavljanjem izvođenja s različitim vrijednostima podataka ustanoviti razlog nastajanja pogreške i nastojati je otkloniti.

Pogreške *treće vrste* teško se otkrivaju, jer se ni tijekom prevođenja, ni tijekom izvođenja ne dojavljuje nikakva pogreška, a program ne daje rezultate koje smo očekivali. One nastaju ili zbog toga što smo za rješavanje nekog zadatka zabunom primijenili krivi algoritam ili smo algoritam na krivi način preveli u program ili nismo na pravi način uzeli u obzir svojstva računala (primjerice, ograničenu preciznost brojeva s pomičnom točkom). Pronalaženje je takvih pogrešaka najteže i zahtijeva mnogo eksperimentiranja s različitim vrijednostima ulaznih podataka. Programska pomagala okruženja za izvođenje programa mogu olakšati pronalaženje takvih pogrešaka time što omogućuju postepeno izvođenje programa u koracima. Na taj način može se pratiti redoslijed izvođenja svake instrukcije. Pritom je razumno u program dodati instrukcije za prikazivanje međurezultata i pratiti njihove vrijednosti.

2. Pristup rješavanju zadataka računalom

U prethodnom odjeljku pozabavili smo se ukratko načinom pripremanja nekog novog programa. Međutim, postavlja se pitanje kako uopće dolazimo do programa koji treba pripremiti. Računala i računalni programi potrebni su nam za rješavanje raznolikih zadataka[14] koji se pojavljuju u životu i radu. Zadaci koji se rješavaju

[14] Danas se računala koriste za rješavanje zadatka praktički u svim granama ljudske djelatnosti. Ona olakšavaju rad mnogim ljudima, a i proširuju mogućnosti čovjeka, jer mogu vrlo brzo i vrlo precizno obavljati neke zadatke. Jedna od skupina zadatka jesu i računalne igre koje pružaju obilje zabave mladim ljudima, a najčešće pobuđuju i interes mladih za računala.

računalom mogu biti vrlo jednostavni i lako razumljivi (primjerice, rješavanje kvadratne jednadžbe) ali i vrlo složeni (primjerice, vođenje poslovanja nekog velikog poduzeća ili automatsko upravljanje zrakoplovom).

Pri učenju progamiranja koriste se uobičajeno vrlo jednostavni primjeri kako bi se pokazalo osnovna svojstva i izražajne mogućnosti nekog programskog jezika. Takvi primjeri mogu stvoriti lažni osjećaj da se i rješavanju nekog ozbiljnijeg zadatka može pristupiti tako da se neposredno sjeda za računalo i odmah počinje pisati program u nekom višem programskom jeziku. *Iskustveno je ustanovljeno da na takav površni način ne mogu nastati dobri i pouzdani programi*.

Pokazalo se da je razumno posao priređivanja programa za rješavanje zadanog zadatka podijeliti u nekoliko koraka. Postoji nekoliko mogućih načina podjele posla na uzastopne korake. Mi ćemo ukratko opisati jednu od takvih podjela i grubo opisati pojedine od njezinih koraka. Prema toj podjeli se pri razvoju programa obavljaju sljedeći koraci:

- analiza zahtjeva,
- specifikacija zadatka,
- zasnivanje programa,
- pisanje programa,
- provjeravanje ili ispitivanje programa,
- održavanje programa.

Razmotrit ćemo ukratko što bi u pojedinom koraku trebalo činiti.

Analiza zahtjeva

Prije negoli počnemo rješavati zadatak moramo znati što treba rješavati. Potreba i zamisao o zadatku može doći iz raznih sredina i ljudi koji zadatke žele rješavati ne moraju znati mnogo o računalima ili programiranju. Zadaci se početno obično opisuju riječima i kaže se, primjerice:

- treba napisati program učeničke evidencije koji mora omogućiti praćenje svih ocjena iz svih predmeta, izračunavanje srednjih ocjena po predmetima, kao i opći uspjeh svakog učenika, te redoslijed učenika po postignutom uspjehu, ili
- treba napisati program koji će omogućiti igranje igre "kružići i križići".

U tom koraku treba što je moguće opsežnije opisati postavljeni zahtjev i odgovoriti na što više pitanja i potpitanja (primjerice, da li se evidenciju o ocjenama vodi i mjesečno i tromjesečno ili samo polugodišnje i godišnje).

Pri analizi zahtjeva uzimaju se u obzir raznoliki uvjeti koji moraju biti zadovoljeni, kao što su trajanje izračunavanja, vjerojatnost nastajanja pogrešaka, vrste računala koje će se koristiti i sl.

Specifikacija zadatka

Nakon analiza zahtjeva slijedi specifikacija zadatka. Specifikacija se sastoji od nabrajanja pojedinosti na koje treba paziti, kao i opisa uvjeta koje budući program mora zadovoljiti. Evo nekoliko pitanja na koja valja odgovoriti prilikom izrade specifikacije:

- Što su ulazni podaci i kojeg su oblika?
- Koje vrijednosti mogu poprimati ulazni podaci?
- Što su izlazni podaci i kojeg su oni oblika?
- Tko će koristiti program i kako mora izgledati korisničko sučelje?
- Postoje li neki posebni slučajevi na koje treba obratiti pažnju?

U specifikacijskijskom koraku treba pripremiti i ispitne podatke koji će poslužiti za provjeravanje ispravnosti programa.

Treba naglasiti da specifikacija određuje samo ono što zadatak treba riješiti a ne i kako će on to rješavati. U tom koraku je još prerano razmišljati o algoritmu koji će se koristiti, jer odabir nekog određenog algoritma može ograničiti zamisao rješavanja zadatka.

Zasnivanje programa

U tom se koraku mora osmisliti moguće načine rješavanja zadatka koji je detaljno opisan u koraku specifikacije. U većini slučajeva zadatak je prevelik i pretežak za rješavanje u obliku jedne cjeline te se mora raščlaniti na manje podzadatke koji su jednostavniji za razmatranje i izgradnju. Takvi podzadaci postat će programski *moduli*[15] — samostalne programske jedinke koje se mogu nezavisno izgraditi i ispitati. Moduli mogu biti pojedinačne funkcije, procedure ili nakupine funkcija i procedura i dijelova programa. Poželjno je da moduli budu što je moguće više nezavisne jedinke koje će se samostalno pripremati i kasnije međusobno povezati. Međusobno povezivanje svodi se uglavnom na razmjenu podataka. Način razmjena podataka između modula možemo nazvati *sučeljem između modula*[16]. Vrlo je važno da ta sučelja budu dobro definirana, tj. da pravila koja opisuju oblik podataka i načine njihove razmjene budu jasna i jednoznačna.

Nakon raščlanjivanja zadatka na module treba obaviti i izbor algoritama za oživotvorenje pojedinih modula. Za svaki modul mora se znati raspoložive ulazne podatke i mora se odrediti kako će izgledati njegovi izlazni podaci. Na temelju željenih pretvorbi podataka treba odabrati jedan od prikladnih ekvivalentnih algoritma.

[15] od latinskog *modulus* — mjera

[16] enleski *interface*

Opis strukture modula i opise primjenjenih algoritama prikladno je načiniti u nekom jeziku zasnivanja programa[17], a ne neposredno u progamskom jeziku ostvarenja. Taj pristup ima nekoliko prednosti:

- programski modul i upotrijebljeni algoritmi mogu se opisati relativno slobodnim jezikom (koji može biti i hrvatski — što nije nevažno);
- ne mora se pretjerano paziti na stroga sintaksna pravila pisanja programa koja mogu skrenuti pažnju s osnovne zamisli na manje važne detalje;
- omogućuje se odgađanje odluke o izboru programskog jezika koji će poslužiti za pisanje programa.

Mi ćemo koristiti jezik zasnivanja programa koji je blizak programskom jeziku Pascal i koji je detaljno opisan u udžbeniku Informatike za 1. razred gimnazija[18]. On omogućuje raznolike mogućnosti zapisivanja programa koje se kreću od vrlo sažetog opisa programskih modula, pa do detaljnog zapisa algoritama kojega se može neposredno (takoreći instrukciju po instrukciju) prevoditi u programski jezik Pascal.

Na taj način može se zamisao programa u početnoj fazi opisati rečenicama hrvatskog jezika, a postepenim uvođenjem detalja može završiti s potpuno određenim specificiranjem svake pojedinačne varijable.

Pisanje programa

Nakon što je program dovoljno detaljno opisan jezikom zasnivanja programa pristupamo pisanju programa ili, bolje rečeno, prevođenju algoritama u odabrani programski jezik[19]. Već smo rekli da ćemo u ovom udžbeniku koristiti jezik Pascal[20]. Jezik zasnivanja programa koji koristimo omogućuje vrlo jednostavno prevođenje u Pascal, ali se uz malo više pažnje razrađeni algoritmi mogu prevesti i u programski jezik C.

Vrlo je rasprostranjeno shvaćanje da je ovaj korak pripremanja programa najvažniji, te da je dovoljno naučiti pravila jezika kako bi se postalo stručnjak za pripremu programa. Međutim, iskustveno je pokazano da to i nije najvažniji korak u životnom razdoblju nekog iole složenijeg programa.

To, međutim, ne znači da učenju nekog progamskog jezika ne treba posvetiti odgovarajuću pažnju. Bez poznavanja nekog programskog jezika ne može se na računalu oživotvoriti nijedan, bilo kako dobar algoritam.

---

[17] engleski: *program design language — PDL*

[18] L. Budin, *Informatika za 1. razred gimnazije*, Element, Zagreb, 1996.

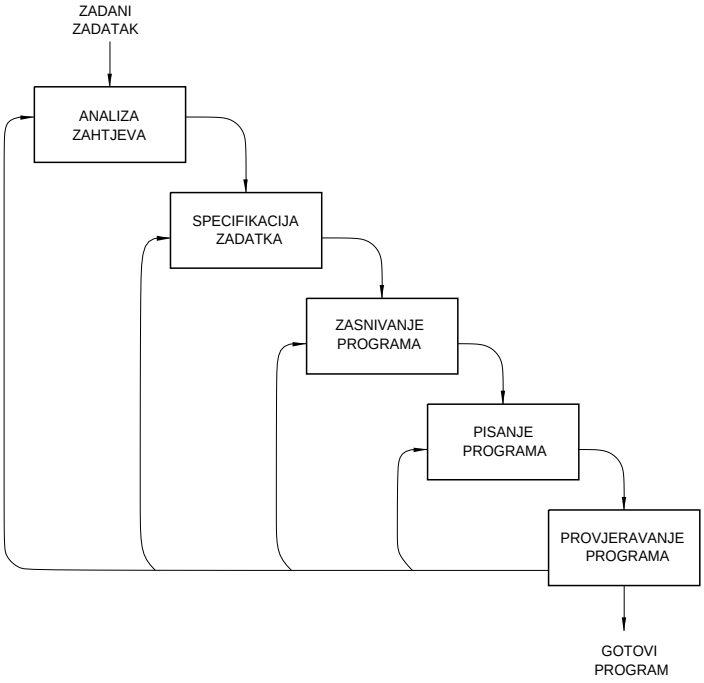
[19] Pisanje programa u nekom programskom jeziku naziva se još i kodiranjem (engleski: *coding*), a izvorni tekst programa naziva se programski kôd (engleski: *source code*). Program zapisan jezikom zasnivanja programa naziva se još i pseudokôd (engleski: *pseudocode*, od grčkog *pseudos* — laž nastao je prefiks *pseudo-* koji u složenicama označava lažnost, nešto tobožnje, prividno).

[20] Još jedanput naglasimo da ćemo koristiti inačicu Turbo Pascal tvrtke Borland. Prevođenje u neku drugu inačicu Pascala zahtijeva u nekim programskim primjerima samo neznatne promjene.

Današnji programski jezici i programska pomagala za njihovo pisanje i ispitivanje znatno olakšavaju taj korak pripreme nekog programa. U ovu fazu programiranja uključuje se i ispravljanje pogrešaka koje se ustanovljuju tijekom prevođenja. Kao što smo već rekli, to su pretežito sintaksne pogreške. Program se smatra napisanim kada uspješno bude preveden u strojni program.

Provjeravanje ili ispitivanje programa

Nakon što je program napisan i pripremljen za izvođenje moramo se uvjeriti da on rješava zadani zadatak ispravno. Postupak provjeravanja[21] obavljamo tako da pripremimo dovoljno veliki skup valjanih vrijednosti ulaznih podataka za koje znademo vrijednosti izlaznih podataka, te da provjeravamo da li program određuje takve vrijednosti.



Sl. 1.2. Koraci u pripremi programa

Neke od pogrešaka, koje smo nazvali pogreškama druge vrste, dojavit će nam računalo i zustaviti daljnje odvijanje programa.

Mnogo teže je otkriti i otkloniti pogreške treće vrste koje nastaju zbog toga što je sačinjen nekorektni algoritam ili je algoritam na pogrešan način preveden iz jezika zasnivanja programa u programski jezik. Temeljito provjeravanje ispravnosti

[21] engleski: *testing*

programa zahtijeva znanje, umješnost i iskustvo. Vrlo je važno znati odabrati prave vrijednosti ispitnih podataka. Tako je važno u ispitne podatke uključiti granične vrijednosti pojedinih varijabli (primjerice, ako se vrijednosti varijable N mogu kretati u granicama od 0 do 99, onda te granične vrijednosti treba uključiti u ispitne vrijednosti). Isto tako, ako je program pisan modularno, onda je prije ispitivanja cijelog programa razumno ispitati pojedinačne module.

Provjeravanjem programa može se ukazati potreba vraćanja ne neke od prethodnih koraka, a nekada je potrebno ponoviti cijeli postupak. Slika 1.2 prikazuje vezu između pojedinih koraka. Zavinute strelice usmjerene prema dolje ukazuju na slijed koraka pripreme programa, a povratne strelice ilustriraju vraćanje na prethodne korake pri ispravljanju programa.

Treba zapamtiti da se nakon uspješnog provjeravanja programa nikako ne može ustvrditi da je program potpuno ispravan. U većini slučajeva mi ne možemo programe ispitivati sa svim mogućim vrijednostima ulaznih varijabli i ne možemo biti sigurni neće li program za neke od vrijednosti koje pri ispitivanju nismo upotrijebili dati pogrešne rezultate. Zato treba imati na umu često ponavljanu uzrečicu da se *provjeravanjem programa može samo utvrditi prisutnost neke pogreške, ali nikako ne i odsutnost pogrešaka*.

Održavanje programa

Kod naših jednostavnih programa, koji će nam pomoći pri izučavanju algoritama i pri učenju programiranja, uspješno provjeravanje će označiti i kraj bavljenja tim programom. Međutim, ozbiljni programi, koji pomažu pri obavljanju nekih ljudima važnih poslova, zahtijevaju trajnu brigu — *održavanje programa*. Pod pojmom održavanja podrazumijeva se ispravljanje naknadno uočenih pogrešaka, prilagođavanje programa nekim manjim promjenama zadatka kojega rješavaju, prenošenje programa na neko drugo računalo i sl. Mi se tim dijelom "životnog razdoblja" nekog programa nećemo u ovom udžbeniku baviti, iako se to razdoblje smatra jednako važnim kao i razdoblje izrade nekog programa.

Svi do sada spomenuti koraci pri izradi i uporabi programa vrlo su važni i izučavaju se u posebnoj grani računarstva koje se naziva programsko inženjerstvo[22].

Izučavanje algoritama i učenje programiranja

Pripremanje programa koji služe za rješavanje nekih korisnih zadataka je posao koji zahtijeva *znanja iz područja odakle potiče zadatak, znanja o algoritmima* koji su prikladni za rješavanje zadanih zadataka, te *znanja, vještine i iskustva pri zasnivanju i pripremi programa*.

Znanja i vještine stjecat ćemo postepeno i to tako da krećemo od rješavanja jednostavnijih zadataka. Mi ćemo istodobno:

[22] engleski: *software engineering*

- izučavati neke korisne algoritme, i
- načine kako se oni pretvaraju u program koji će omogućiti da se oni automatski izvode na računalu.

Prema tome, naše izučavanje načina programiranja u programskom jeziku Pascal bit će povezano s primjerima koji rješavaju neke korisne podzadatke koji mogu kasnije poslužiti kao gradivni elementi za izradu složenijih programa.

Pisanje dobrih, jasnih i čitljivih programa nije jednostavno i zahtijeva mnogo iskustva. Iskustvo se stječe dugotrajnim radom. Međutim, iskustvo naših prethodnika, koje su oni stekli čineći i ispravljajući pogreške, može poslužiti i nama ako slijedimo i oponašamo neke uzorke nastale tijekom vremena. Programi se mogu napisati na različite načine i ne postoji neka isključiva mjera za izbor nekog uzorka programa. Korisno je, međutim, slijediti jedan od mogućih stilova[23] pisanja programa. Mi ćemo u jeziku zasnivanja progama i u pisanju programa u jeziku Pascal nastojati koristiti jedan što razgovijetniji stil pisanja. Tijekom postupnog stjecanja znanja, vještine i iskustva treba se prisjetiti "vodopadnog" modela postupka pripreme novog programa prikazanog slikom 1.2. Naime, mi ćemo pri početnom izučavanju pojedinih korisnih algoritma i načina kako se oni zapisuju programskim jezikom preskakati neke od spomenutih koraka i započinjati postupak pripreme negdje "u sredini vodopada" i nećemo uvijek polaziti od njegova "izvora". Obično se za izučavanje osnovnih algoritama koriste jednostavni zadaci i pojedini koraci "vodopadnog modela" pripreme međusobno se stapaju ili se potpuno preskaču.

Treba još jedanput naglasiti da se često pokušava i složenije zadatke rješavati tako da se neposredno sjeda za tipkovnicu i odmah "iz glave" počinje pisati program. Za neke jednostavne primjere vješti programer može tako i napisati dobar program, ali već i najmanje promjene zadatka ili njegovo proširenje dovode do niza poteškoća.

Ovo početno poglavlje treba češće ponovno pregledati i vratiti se na njega kada se savladaju neka nova znanja i vještine. Ono će uz svako novo stečeno iskustvo postajati sve razumljivije.

3. Osnovna svojstva programskog jezika Pascal

Već smo ustanovili da nam računalo može pomoći pri rješavanju različitih zadataka koji potiču iz vrlo različtih područja ljudske djelatnosti. Rekli smo da su prvi koraci u pripremi programa analiza zahtjeva i specifikacija zadatka i da se nakon što su oni obavljeni može pristupiti zasnivanju programa. Mi ćemo pri

[23] od grčkog *stylos* — držak, pisaljka; u prenesenom smislu: način pisanja, način ponašanja, navika

prvom upoznavanju načina pripreme programa, u većini slučajeva, pretpostaviti da su ti prvi koraci već obavljeni.

Dugogodišnje je iskustvo pokazalo da raščljanjivanje vrlo raznolikih zadataka vodi na neke tipične podzadatke koji se često ponavljaju. Pokazalo se vrlo vrijednim takve podzadatke podrobno analizirati i istražiti moguće algoritme za njihovo rješavanje, te odabrati najprikladniji način njihova ostvarenja računalom uz uvažavanje svih njegovih ograničenja. Od takvih riješenih podzadataka može se postupno izgrađivati program za rješavanje sve složenijih zadataka.

Mi ćemo na put u ovo vrlo zanimljivo područje ljudske djelatnosti, koje je u svakom pogledu potpuno otvoreno za stvaralačko djelovanje, započeti upravo izučavanjem već istraženih algoritama i načinima njihova prevođenja u programe koji se mogu izvoditi na računalu. Poznavanje i korištenje takvih algoritama i pripadnih programskih rješenja, koja smo dobili u nasljeđe od prethodnih generacija, olakšava nam pripremanje programa za rješavanje novih zadataka, jer ne moramo ponovno otkrivati već poznate i istražene postupke.

Imajmo stalno na umu da ćemo pri savladavanju gradiva ovog udžbenika istodobno:

- izučavati algoritme za rješavanje nekih tipičnih zadataka i podzadataka i
- izučavati kako se ti algoritmi zapisuju u programskom jeziku Pascal i zatim izvode računalom.

Za *izučavanje* i *zapisivanje algoritama* služit ćemo se već spomenutim *jezikom zasnivanja programa* ili *pseudojezikom*. Podsjetimo se da se u našem jeziku zasnivanja programa koriste hrvatske ključne riječi i da se grubi opis algoritma može načiniti slobodno oblikovanim hrvatskim rečenicama. To nam olakšava razumijevanje algoritama, jer zapisujemo misli onako kako smo ih od djetinjstva navikli oblikovati — na svom materinjem jeziku. Postupno se opis algoritama može pretvoriti u oblik koji sve više sliči hrvatskoj inačici Pascala. Međutim, algoritmi zapisani tim pseudojezikom mogu se, bez većih poteškoća, prevesti i u neki drugi programski jezik, promjerice, jezik C.

Prevođenje algoritama u Pascal zahtijeva poznavanje strogih pravila tog jezika. Već smo rekli da ćemo u ovom udžbeniku koristiti inačicu *Turbo Pascal* tvrtke Borland, ali se primjeri programa mogu uz neznatne izmjene prilagoditi drugim inačicama Pascala. Mi ćemo pravila pisanja uvoditi postupno, onako kako budemo napredovali u učenju mogućnosti jezika. Detaljni popis svih pravila programiranja može se naći u priručnicima koji opisuju Pascal. U tim se knjigama u posebnim prilozima nalaze opisi sintaksnih pravila.

Struktura programa napisana u programskom jeziku Pascal

Program napisan u Pascalu ima sljedeću osnovnu strukturu:

- zaglavlje programa;

- deklaracijski dio programa;
- instrukcijski dio programa.

U *zaglavlju programa* se uz ključnu riječ `program` pojavljuje ime programa.

Mi ćemo u programima zapisanim jezikom Pascal ključne riječi podcrtavati na jednaki način kao što činimo u jeziku zasnivanja programa. To činimo stoga da u knjizi, na ploči i u bilježnici budu na jednaki način istaknute. Preporučljivo je, zbog čitljivosti, sve programe zapisane jezikom pisati na taj način. Pri utipkavanju programa ključne riječi nećemo podcrtavati. Međutim, uređivač teksta u nekim radnim okruženjima ključne će riječi zapisati na zaslonu drugom bojom[24].

Drugi, *deklaracijski dio* progama mora u potpunosti opisati sve varijable i konstante koje se u programu koriste, te funkcije i procedure koje su napisane za taj program. Podrobniji izgled pojedinih komponenti deklaracijskog dijela postepeno ćemo saznavati.

Instrukcijski dio programa sastoji se od niza instrukcija[25] koje opisuju niz koraka algoritma. Instrukcijski dio programa počinje ključnom rječju `begin` i završava ključnom rječju `end` iza koje dolazi točka[26].

Primjer 1.1. Uobičajeni prvi primjer u mnogim programerskim početnicama je program koji ispisuje poruku "Pozdrav svijetu!". Mali program koji to obavlja izgleda ovako:

```
program Pozdrav;

begin
   write('Pozdrav svijetu!')
end.
```

Ovaj kratki program nema deklaracijskog dijela. U zaglavlju se pojavljuje nama odabrano ime programa `Pozdrav`. Instrukcijski dio programa sadrži samo jednu jedinu instrukciju koja će na zaslonu monitora ispisati tekst:

```
Pozdrav svijetu!
```

Ona se sastoji od riječi `write` iza koje je u zagradama između polunavodnika ("visokih zareza", apostrofa) napisan tekst koji želimo ispisati na zaslon[27]. Ta instrukcija aktivira ("poziva") proceduru za ispis sadržaja na zaslonu monitora.

□

Primjer 1.2. Drugi kratki program koji računa i ispisuje kvadrat upisanog cijelog broja mogao bi se napisati na sljedeći način:

[24] Programsko okruženje IDE za razvoj programa tvrtke Borland na višebojnim monitorima ključne riječi ispisuje drugom bojom

[25] engleski naziv za instrukciju je *statement*, što izvorno znači iskaz, očitovanje

[26] engleski: *begin* — početi, *end* — završiti

[27] engleski: *write* — pisati

```
program Kvadriranje;

var
   X,Y: Integer;

procedure Ulaz(var Z: Integer);
begin
   writeln ('Upisati jedan cijeli broj i pritisnuti tipku ENTER');
   readln(Z)
end;

procedure Izlaz (var Z: Integer);
begin
   writeln;
   writeln ('Rezultat izracunavanja je ',Z)
end;

function Kvadrat_od (var Z: Integer):  Integer;
begin
   Kvadrat_od := Z * Z
end;

begin
   Ulaz(X);
   Y := Kvadrat_od (X);
   Izlaz(Y)
end.
```

□

Prije nego objasnimo osnovne dijelove tog programa napišimo program koji će obaviti isti posao, a izgleda mnogo jednostavnije:

```
program Kvadriranje;

var
   X: Integer;

begin
   writeln ('Upisati jedan cijeli broj i pritisnuti tipku ENTER');
   readln(X);
   X := sqr (X);
   writeln;
   writeln ('Rezultat izracunavanja je ', X)
end.
```

Ovaj drugi jednostavniji program nakon zaglavlja u kojem je zapisano ime programa ima kratki deklaracijski dio u kojem je deklarirana iz ključne riječi `var` cjelobrojna varijabla `X`. Instrukcijski dio sastoji se od pet instrukcija. Zaglavlje, deklaracija varijable i svaka instrukcija zaključeni su razdjelnim znakom *točka-zarez*.

Objasnimo ovdje samo ukratko značenje pojedine od instrukcija (detaljnije ćemo ih izučavati kasnije). Instrukcija za ispis teksta `writeln` jednako kao i instrukcija `write` ispisuje tekst napisan unutar apostrofa, ali nakon ispisa izaziva još i pomak na početak novog retka. Instrukcija `readln` aktivira proceduru koja očitava niz otipkanih znamenki na tipkovnici i prudružuje vrijednost tako određenog broja varijabli `X`, te načini pomak u novi redak (što je određeno sufiksom `-ln`[28]). Nakon očitanja vrijednosti varijable u sljedećoj se instrukciji funkcijom `sqr(X)` izračunava njezin kvadrat i pohranjuje u istu varijablu[29]. Ta funkcija nalazi se u zbirci gotovih funkcija Pascala. Na kraju se instrukcijom `writeln` ispisuje jedan prazni redak, te rezultat izračunavanja s popratnim tekstom.

Nakon pokretanja programa, upisa traženog broja (primjerice: -12) i pritiska na tipku `ENTER`, može se na zaslonu monitora pročitati sljedeći tekst:

```
Upisati jedan cijeli broj i pritisnuti tipku ENTER
-12

Rezultat izracunavanja je 144
```

Potpuno jednako izgledao bi monitor i nakon izvođenja prvog programa. Njegov izgled je složeniji jer mu je deklaracijski dio veći. Međutim, njegov instrukcijski dio napisan je mnogo sažetije i preglednije. Promotrimo ga malo podrobnije i uočimo neke prednosti ovakve pripreme programa:

```
begin
   Ulaz(X);
   Y := Kvadrat_od(X);
   Izlaz(Y)
end.
```

U prvom redu, instrukcijski dio postao je vrlo pregledan jer su u njemu zapisane samo tri osnovne akcije koje se moraju provesti (kod složenijih programa to postaje još izrazitije). Nadalje, procedure `Ulaz(Z)` i `Izlaz(Z)` mogu poslužiti i za bilo koji drugi program i ne moramo ih posebno pisati. Zamjenom funkcije `Kvadrat_od (Z)` (ovdje je napisana kao umnožak broja samog sa sobom iako u Pascalu postoji već gotova funkcija `sqr`) s bilo kakvom drugom funkcijom rješavamo neki drugi zadatak i jednostavno dobivamo neki drugi program. Želimo li, primjerice, umjesto računanja druge potencije izračunavati treću potenciju upisanog broja, treba samo u deklaracijskom dijelu programa definirati novu funkciju

```
function Treca_potencija_od (var Z: Integer):  Integer;
begin
   Treca_potencija_od := Z * Z * Z
end;
```

[28] sufiks *ln* dolazi od engleske riječi *line* — redak

[29] Skraćenica *sqr* dolazi od engleskog *square* — kvadrat. U Pascalu postoji i druga funkcija koja računa drugi (kvadratni) korijen i ima skraćenicu *sqrt* od engleskog *square root* — kvadratni korijen. Potrebno je pripaziti da zbog razlike u samo jednom slovu ne nastane zabuna.

i u instrukcijskom dijelu programa promijeniti instrukciju pridruživanja i dobiva se:

```
begin
   Ulaz(X);
   Y := Treca_potencija_od(X);
   Izlaz(Y)
end.
```

Nakon što pokrenemo taj novi program dobit ćemo utipkavanjem broja -12 sljedeći izgled teksta na zaslonu monitora:

```
Upisati jedan cijeli broj i pritisnuti tipku ENTER
-12

Rezultat izracunavanja je -1728
```

□

U prethodnom primjeru pojavljuju se neke za sada neobjašnjene jezične konstrukcije. Tijekom izučavanja gradiva iz ovog udžbenika sve će one biti razjašnjene. Uočimo samo neke važne posebnosti.

Pri definiranju procedura mi im određujemo ime i u popisu parametara naglašavamo što su varijable tako da se ispred imena varijabli stavlja ključna riječ `var`. Ime varijable u definiciji procedure samo "čuva mjesto" za pravo ime varijable koje će se pojaviti pri pozivu procedure. Tako u sve tri definicije procedura koristimo isto ime `Z`, a kasnije u instrukcijskom dijelu programa koristimo prava imena varijabli `X` i `Y`. Kasnije ćemo se posebno pozabaviti uporabom varijabli unutar i izvan procedura.

Isto tako, u definiciji funkcija se uz ključnu riječ `function` može odabrati proizvoljno ime, a isto tako i neka proizvoljna imena njezinih ulaznih varijabli, koja se pri pozivima zamjenjuju pravim imenima. Za funkciju pri njezinu definiranju treba napisati koji tip rezultata će ona vratiti nakon provedenog izračunavanja. Ime funkcije mora se u definiciji pojaviti s lijeve strane instrukcije pridruživanja. Vrijednost koja će pri pozivu funkcije biti pridružena tom imenu vraća se u program koji je funkciju pokrenuo.

Primjer 1.3. Napišimo program koji će prihvatiti dva prirodna broja tako da je prvi od njih baza a drugi eksponent. Najjednostavniji način izračunavanja potencije bio bi uzastopno množenje s bazom i to onoliko puta koliko to određuje eksponent. Ako bazu zapišemo u varijablu `X`, a eksponent u varijablu `Y`, potenciranje ćemo obaviti s `Y` množenja u skladu sa sljedećim algoritmom:

```
Z := 1;
za I = 1 do Y činiti
  | Z := Z * X;
```

Međutim, potenciranje se može obaviti s manje množenja sljedećim algoritmom:

```
Z := 1;
dok je Y > 0 činiti
    ako je Y neparno
        onda Z := Z * X,
    X := X * X;
    Y := div(Y,2);
```

Uvjerite se na nekoliko primjera da se tim algoritmom stvarno izračunava tražena potencija. Funkcija cjelobrojnog dijeljenja `div(Y,2)` raspolavlja u petlji vrijednost od `Y` tako dugo dok je `Y > 0`. Početna vrijednost za potenciju `Z = 1` pomnožit će se s trenutnom vrijednosti `X` samo ako je trenutna vrijednost od `Y` neparna. Prije dijeljenja s `2` varijabli `X` pripisuje se nova vrijednost koja je kvadrat prethodne vrijednosti iste varijable. Algoritam daje dobar rezultat i za `Y = 0`. Tada je, naime, `Z =1`.

Pogledajmo vrijednosti međurezultata nakon svakog prolaza kroz petlju za početnu vrijednost $X = X_0$ i za početne vrijednosti $Y_0 = 15$ i $Y_0 = 16$.

Za $Y_0 = 15$ dobivamo:

	Z	X	Y
početne vrijednosti	1	X_0	15
nakon prvog prolaza	X_0	X_0^2	7
nakon drugog prolaza	X_0^3	X_0^4	3
nakon trećeg prolaza	X_0^7	X_0^8	1
nakon četvrtog prolaza	X_0^{15}	X_0^{16}	0
konačni rezultati	X_0^{15}	X_0^{16}	0

Za $Y_0 = 16$ dobivamo:

	Z	X	Y
početne vrijednosti	1	X_0	16
nakon prvog prolaza	1	X_0^2	8
nakon drugog prolaza	1	X_0^4	4
nakon trećeg prolaza	1	X_0^8	2
nakon četvrtog prolaza	1	X_0^{16}	1
nakon petog prolaza	X_0^{16}	X_0^{32}	0
konačni rezultati	X_0^{16}	X_0^{32}	0

Uočimo da se petlja ponavlja onoliko puta koliko nam bitova treba za zapisivanje eksponenta u binarnom obliku! U svakoj petlji obavlja se najviše dva množenja (jedno množenje ako je trenutna vrijednost od `Y` neparna odnosno dva množenja ako je vrijednost od `Y` neparna). Zadnje množenje za dobivanje nove vrijednosti `X` nije ni potrebno i moglo bi se izbjeći tako da se obavlja samo onda kada je `Y > 1`. Takav ispravljeni algoritam bi glasio:

```
Z := 1;
dok je Y > 0 činiti
    ako je Y neparno
        onda Z := Z * X,
    ako je Y > 1
    onda X := X * X;
    Y := div (Y,2);
```

Mi ćemo, međutim, ostaviti algoritam u prvotnom obliku i pogledati kako se on može prevesti u Pascal. U program nazvan `Potenciranje` dodane su instrukcije za obavljanje ulaznih i izlaznih operacija. U njemu se pojavljuju još neke nove riječi i konstrukcije. Tako se petlja

```
dok je uvjet činiti
```

u Pascalu zapisuje engleskim ključnim riječima

```
while Y > 0 do,
```

a ispitivanje uvjeta

```
ako je uvjet
  | onda instrukcija;
```

zapisuje se na sljedeći način

```
if uvjet
    then instrukcija;
```

Grupa instrukcija koje u jeziku zasnivanja programa označavamo vertikalnom crtom u Pascalu se ograđuje ključnim riječima `begin` i `end`.

U programu se pojavljuje još i funkcija `odd(Y)`. To je funkcija definirana unutar Pascala koja daje rezultat istinitost i ili neistinitost, tj. može poprimiti vrijednosti logičke varijable. Ona daje kao rezultat istinitost onda kada joj je argument neparan[30]. Nadalje, treba uočiti da se operator cjelobrojnog dijeljenja `div` piše izmedu operanada.

U programu su predviđene posebne varijable `X0` i `Y0` za sačuvanje početnih vrijednosti baze i eksponenta kako bi se na kraju izračunavanja one mogle koristiti pri ispisivanju rezultata. Program se može napisati na sljedeći način:

```
program Potenciranje;

var
    X,Y,X0,Y0,Z: Integer;

begin
   writeln('Upisati prirodne brojeve:');
   write(' baza = ');
   readln(X);
   write(' eksponent = ');
   readln(Y);
   X0 := X;
   Y0 := Y;
   Z := 1;
   while Y > 0 do
       begin
           if odd(Y)
              then Z := Z * X;
           X := X * X;
          Y := Y div 2
       end;
   writeln;
   writeln('Potencija(baza**eksponent)iznosi',X0,'**',Y0,'=',Z)
end.
```

U popratnom tekstu koji nam pomaže pri upisivanju ulaznih vrijednosti nije posebno istaknuto da pri unošenju brojeva treba pritisnuti tipku `ENTER`, kao što smo to učinili u primjeru 1.2. To se podrazumijeva samo po sebi.

[30] Ime joj dolazi od engleskog *odd* — neparan

Kada taj program pokrenemo, te nakon što utipkamo vrijednosti za bazu (primjerice: 2) i eksponent (primjerice : 10) onda kada to "zatraži" program na zaslonu ćemo konačno vidjeti napisan sljedeći tekst:

```
Upisati prirodne brojeve:
    baza = 2
    eksponent = 10

Potencija (baza**eksponent) iznosi 2**10 = 1024
```

Međutim, ako zapišemo za eksponent vrijednost 15 dobit ćemo sljedeći ispis:

```
Upisati prirodne brojeve:
    baza = 2
    eksponent = 15

Potencija (baza**eksponent) iznosi 2**15 = -32768
```

a za vrijednost eksponenta 16 dobiva se:

```
Upisati prirodne brojeve:
    baza = 2
    eksponent = 16

Potencija (baza**eksponent) iznosi = 0
```

Naime, u ovoj inačici Pascala cjelobrojna varijabla tipa `Integer` ima šesnaest bitova i u nju su pohranjeni brojevi u dvokomplementnom obliku. To razjašnjava čudne rezultate koje smo dobili, ali i ukazuje na to da mi sami moramo paziti na to što se zbiva s izračunavanjima koje smo zamislili. □

Ovih nekoliko primjera ukazuje na opći izgled programa napisanih u programskom jeziku Pascal. Iz njih je vidljivo da se algoritmi zapisani u jeziku zasnivanja programa mogu relativno jednostavno prevesti u programski jezik Pascal. Kako bi napisani programi bili čitljivi i razumljivi, korisno je programe pisati tako da im struktura bude jasno istaknuta. U prethodnim primjerima to je i načinjeno.

Napomenimo da za kompilator nije važno kako je program na papiru raspoređen. Važno je samo da su svi znakovi i simboli na svojim mjestima i da su instrukcije razdvojene znakovima točka-zarez. To nam ostavlja mogućnost da programe po izgledu proizvoljno oblikujemo. Međutim, dobro se je držati jednog načina pisanja programa, jer na taj način olakšavamo sebi njihovo razumijevanje.

Primjer 1.4. Program iz primjera 1.3. možemo napisati bez prepoznatljive strukture i bez isticanja ključnih riječi. Iako mi nećemo tako pisati naše programe pokušajmo niže napisani program propustiti kroz kompilator i nakon toga ga izvoditi. Uvjerit ćemo se da on potpuno jednako radi kao program iz primjera 1.3. Sa stanovišta kompilatora ta su dva programa potpuno jednaka.

```
program Potenciranje; var X,Y,X0,Y0,Z: Integer; begin
writeln('Upisati prirodne brojeve:'); write(' baza = '); readln(X);
write(' eksponent = '); readln(Y); X0 := X; Y0 := Y; Z := 1;
while Y > 0 do begin if Odd(Y) then Z := Z * X; X := X * X;
Y := Y div 2; end; writeln;
writeln('Potencija (baza**eksponent) iznosi ',X0,'**',Y0,' = ', Z);
end.
```

□