Univerzitet u Nišu

Elektronski fakultet

Skripta iz predmeta

Algoritmi i programiranje

Skripta je napisana po predavanjima doc. dr Vladimira Ćirića na Elektronskom fakultetu u Nišu školske 2012./2013. godine (nastavna grupa C). Tekst, slike i primeri su preuzeti sa predavanja.

Vasić Milan, Vulović Mirko

Sadržaj:

Osnovni pojmovi i način predstavljanja algoritama 1
Osobine algoritama 2
Načini predstavljanja algoritama 2
Osnovne algoritamske strukture 3
Načini predstavljanja algoritama - nastavak 6
Promenljive, tipovi i strukture podataka 7
Osnovni tipovi 8
Složeni (strukturni) tipovi 9
Razvoj programskog jezika C 13
Karakteristike jezika C 13
Faze u implementaciji programa na jeziku C 13
Azbuka jezika C 14
Tokeni jezika C 14
Deklaracija promenljivih 15
Tipovi podataka 15
Struktura C programa 16
Operatori 16
Naredbe za ulaz i izlaz 18
Kontrola toka programa 19
Polja 21
Dekompozicija i funkcije u C-u 21
Standardne biblioteke u programskom jeziku C 25
Izvedeni tipovi podataka 25
Stringovi 32
Osnovne operacije za rad sa stringovima 33
Dokumentovanje funkcija 34
Biblioteka string.h 34
Nizovi pokazivača 39
Ulaz/izlaz i fajlovi 40
Tipovi fajlova 42
Lokalne i globalne promenljive 45
Memorijske klase identifikatora 46
Dinamička alokacija memorije 47
Preprocesorske direktive 49

Osnovni pojmovi i način predstavljanja algoritama

Etape u rešavanju problema pomoću računara

U reševanju problema pomoću računara postoji 5 faza:

1. Precizan opis problema
2. Definisanje modela i izbor metode
3. Projektovanje algoritama
4. Pisanje i testiranje programa
5. Izrada prateće dokumentacije

Precizan opis problema

U okviru ove faze potrebno je definisati ulazne parametre algoritma, ono šta se očekuje kao izlaz i precizno definisati problem odnosno transformaciju ulaznih podataka u izlazne. Opis problema je u većini slučajeva tekstualni. Za standardizaciju opisa problema postoje specijalizovani jezici kao što je UML (Unified Modeling Language).

Definisanje modela i izbor metode

U okviru ove faze potrebno je definisati model za razvoj softvera. Postoji veliki broj različitih modela koji zavise od veličine softvera koji se razvija. Jedan od poznatijih modela za razvoj velikih softverskh projekata je iterativni model. Kod ovog modela se u kratkim vremenskim periodima rešenje daje korisniku na proveru. Što se algoritama tiče metode se dele u dve grupe:

1. Tačne metode
2. Približne metode

Tačne metode u konačnom broju koraka dovode do tačnog rešenja, dok približne metode daju rešenje koje je dovoljno tačno za datu primenu. Primer približnih metoda su iterativne metode koje su predmet istraživanja numeričke matematike. Kod iterativnih metoda postoji definisan postupak kako od među-rešenja odrediti tačnije rešenje

Primer: Iterativni metod za f-ju $X = \sqrt[n]{a}$

$X_0 = (a+n-1)/n$

$X_i = ((n-1)X_{i-1} + a/X^{(n-1)}_{i-1})/n$

Projektovanje algoritama

Algoritam je precizni postupak za rešavanje problema. Koren reči ALGORITAM je u imenu persijskog astronoma i matematičara Al-horezma. On je 825. godine napisao knjigu *"On calculation with Hindu numerals".* Ova knjiga je prevedena na latinski pod nazivom *"Algoritmi de numero Indorum"* koja je kasnije prevedena na engleski jezik pod nazivom *"Algoritmus on the numbers of the Indians".* Reč algoritam označava metod za izračunavanje.

Pisanje i testiranje programa

Algoritamski rešen problem moguće je izvršiti na računaru u vidu programa. Generalno, postoje tri kategorije programskih jezika a to su:

1. Mašinski jezik
2. Asemblerski jezik
3. Viši programski jezici

Mašinski jezik

Mašinski jezik je jedini jezik koji se direktno može izvršiti na računaru. Mašinski jezik je niz sukcesivnih naredbi predstavljenih sa 0 i 1. Problem kod mašinskog jezika je što programmer mora detaljno poznavati strukturu procesora. S druge strane teško je uočiti i ispraviti greške

Asemblerski jezik

Kod asemblerskih jezika naredbe su zamenjene simbolima (mov Ax,OxF1 …). Ovde i dalje postoji problem da je potrebno poznavati arhitekturu procesora. Da bi se asemblerski jezik izvršio potrebno ga je kompajlirati (prevesti) na mašinski jezik.

Viši programski jezici

Viši programski jezici su uvedeni u cilju prevazilaženja zavisnosti od konkretne arhitekture procesora. Sintaska viših programskih jezika je bliska govornim jezicima. Da bi se program napisan na višem programskom jeziku izvršio potrebno ga je prevesti na mašinski jezik. Prevođenje programa obajvlja se u više faza. Asembler je jedna od među-faza.

Izrada prateće dokumentacije

Prateća dokumentacija obuhvata korisnička i sistemska uputstva. Sistemska uputstva se koriste za održavanje i nadogradnju softvera.

Osobine algoritama

Osnovne osobine algoritama jesu:

1. Diskretnost
2. Determinisanost
3. Efikasnost(konačnost)
4. Rezultativnost
5. MasovnostOptimalnost

Diskretnost

Svaki algoritam se sastoji od jasno izdvojenih koraka. Takvih koraka ima konačno mnogo. Diskretnost se dobija dekompozicijom problema odnosno deljenjem problema na manje korake.

Determinisanost

Svaki korak algoritma treba da ima precizno određene ulaze, izlaze i zadatke koje obavlja.

Efikasnost(konačnost)

Svaki algoritam treba da da rezultat nakon konačanog broja koraka.

Rezultativnost

Algoritam treba da da rezultat za svaki skup ulaznih podataka.

Masovnost

Dobar algoritam rešava određeni problem za celu klasu ulaznih parametara, a ne samo za partikularni slučaj.

Optimalnost

Optimalan algoritam vodi rešenju nekog problema za što manji broj koraka.

Načini predstavljanja algoritama

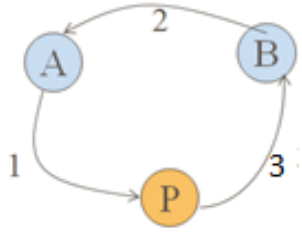
Postoji nekoliko načina za predstavljanje algoritama:

1. Tekstualno
2. Grafički
3. Pseudo kod
4. Strukturogrami

Tekstualni prikaz

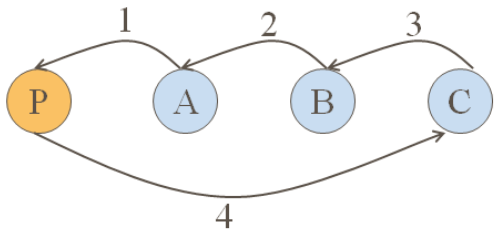
Kod tekstualnog načina predstavljanja algoritama koriste se prelazne rečenice govornog jezika.

Primer: Razmeniti sadržaje lokacija A i B.



Sadržaj lokacije A premestiti na lokaciju P, sadržaj lokacije B premestiti na lokaciju A i sadržaj lokacije P premestiti na lokaciju B.

Primer: Ciklično pomeriti sadržaje lokacija A,B i C za jedno mesto u levo.



Sadržaj lokacije A premestiti na lokaciju P, sadržaj lokacije B premestiti na lokaciju A, sadržaj lokacije C premestiti na lokaciju B i sadržaj lokacije P premestiti na lokaciju C.

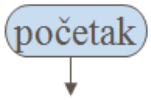
Primer: Napisati algoritam za nalaženje NZD prirodnih brojeva M i N po euklidovom algoritmu.

Veći od dva broja upisati u M, manji u N. Podeliti M sa N i ostatak upisati u R. Ako je R=0 onda je NZD=N. Ako je R≠0 onda N prebaciti u M, a R prebaciti u N i ponoviti postupak.

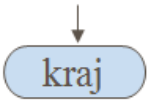
Grafičko predstavljanje algoritama

Osnovu grafičkog predstavljanja definiše matematička oblast teorija grafova. Kod grafičkog predstavljanja algoritmi se predstavljaju usmerenim grafom koga čine čvorovi i grane. Čvorovi predstavljaju obradu podataka a grane prelazak iz jednog čvora u drugi nakon izvršenja zadatka koji obavlja čvor.

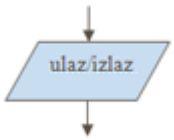
1. Startni, polazni čvor



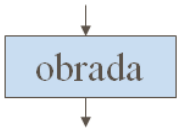
2. Krajnji, završni čvor



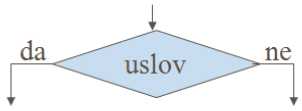
3. Ulaz/izlaz



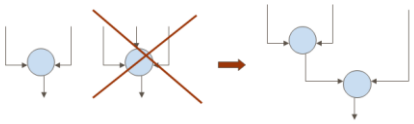
4. Blok obrade



5. Blok odluke



6. Blok za spajanje potega



Blok za ulaz/izlaz se koristi za komunikaciju sa spoljnim svetom. Preko ovih blokova zadaju se parametri i korisniku se prikazuje rezultat.

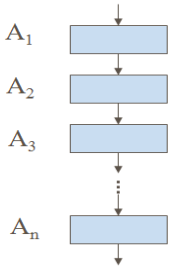
Blok obrade vrši obradu nad podacima.

Blok odluke se koristi za grananja u grafu nakon kojih izvršenje može krenuti jednim od više zadatih tokova u zavisnosti od uslova.

Osnovne algoritamske strukture

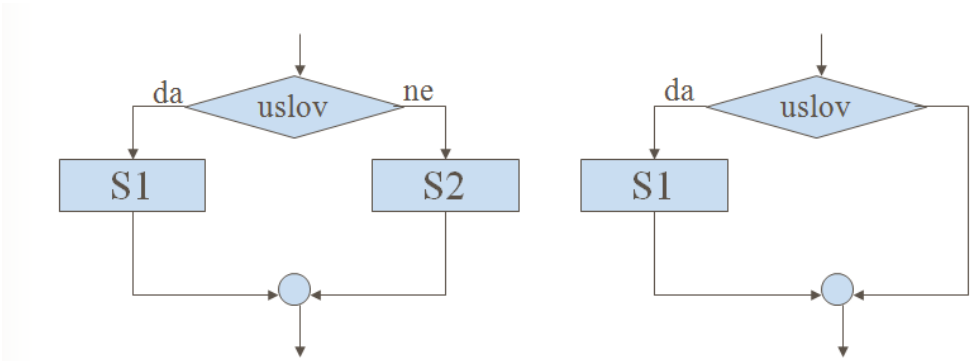
Sekvenca

Sekvenca je algoritamska struktura koja se dobija kaskadnim povezivanjem blokova obrade. Više sukcesivnih blokova je moguće zameniti jednim.



Alternacija

Alternacija je algoritamska struktura kod koje se u zavisnosti od zadatog uslova izvršava jedan od dva zadata pod-algoritma.



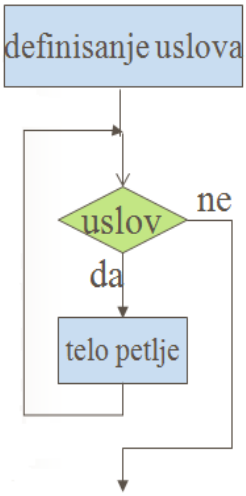
Petlje

Petlje omogućavaju višestruko izvršenje određenog dela algoritma. Postoje tri tipa petlji:

1. while
2. repeat-until (do-while)
3. for

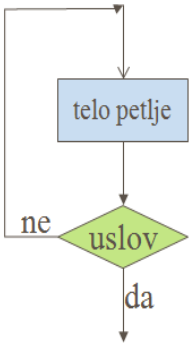
while petlja

Kod while petlje uslov je dat na početku, pre bloka naredbi. Blok naredbi se izvršava sve dok je uslov ispunjen.



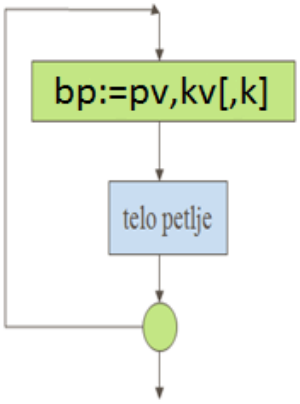
repeat-until (do-while) petlja

Blok naredbi se izvršava sve dok se uslov ne ispuni. Ono što je karakteristično za ovu petlju je to što se jedino kod ove petlje telo petlje mora izvršiti barem jednom.



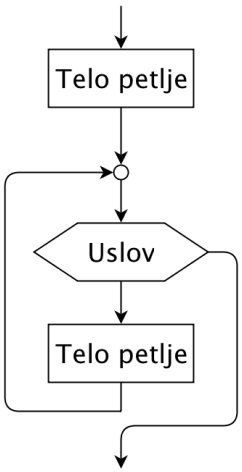
for petlja

Na početku izvršenja petlje tipa for brojač petlji (bp) dobija početnu vrednost (pv) i telo petlje se izvršava jednom. Nakon izvršenja brojač se uvećava za korak (k) i telo petlje se izvršava još jednom, i tako sve dok brojač ne dostigne krajnju vrednost (kv).

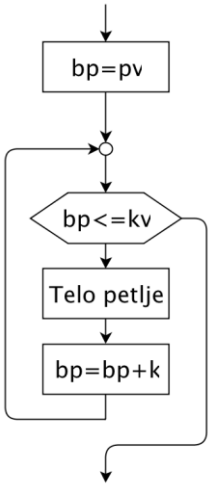


Minimalni broj izvršenja while petlje je 0, repeat-until petlje je 1, a za for petlje minimalan broj izvršenja jeste (kv–pv)/k (ako nije ceo broj onda se uzima prvi veći). Za implementaciju bilo kog algoritma dovoljan je samo jedan tip petlje a to je while. Preko while petlje se mogu prikazati petlje tipa for i repeat-until.

- repeat-until pomoću while



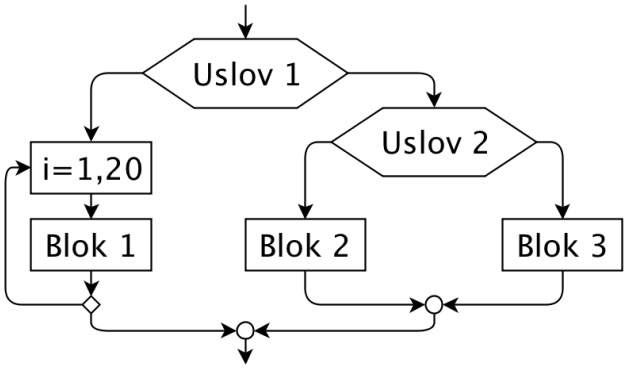
- for pomoću while



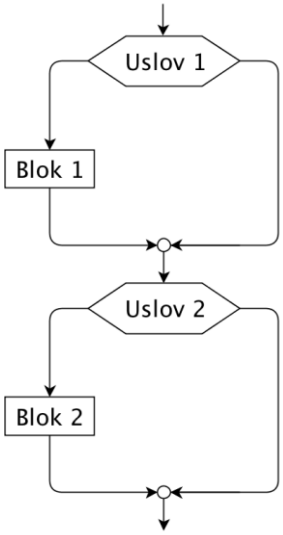
Strukturni algoritmi

Implementacija algoritama vrši se kombinovanjem osnovnih struktura. Strukture se mogu nadovezivati jedna na drugu ili biti ugnježdene jedne u druge. U klasu strukturnih algoritama spadaju algoritmi kod kojih svaki pod-algoritam ima jednu ulaznu i jednu izlaznu tačku.

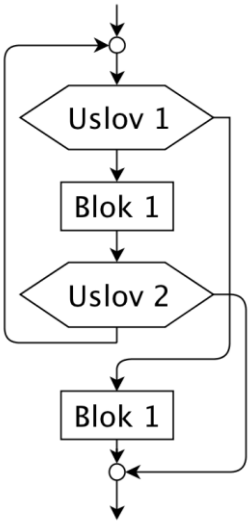
- Primer ugnježdenih struktura



- Primer nadovezivanja



- Primer nestrukturnog algoritma



Načini predstavljanja algoritama - nastavak

Pseudo kod

Kod pseudo koda za predstavljanje algoritama koristi se tekstualni način u osnovi koji je dopunjen formalizmima.

Sekvence

Sekvence se u pseudo kodu pišu tako što se navodi po jedan iskaz u jednom redu. Redovi se odvajaju tačkom i zarezom (;).

Primer:

```
x=3;
y=5;
z=x+y;
```

Alternacija

Alternacije se u pseudo kodu predstavljaju na sledeći način:

if:

```
If(uslov) then
    blok;
endif;
```

if-else:

```
If(uslov) then
        blok1;
else
        blok2;
endif;
```

Petlje

Petlje se prikazuju na sledeći način:

while:

```
while(uslov) do
        blok;
enddo;
```

repeat-until:

```
repeat
        blok;
until(uslov);
```

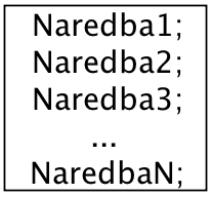
for:

```
for bp:=pv,kv[,k]do
    blok;
enddo;
```

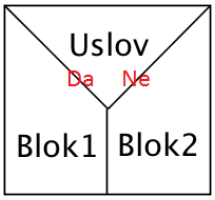
Strukturogrami

Strukturogrami su način za predstavljanje algoritama koji predstavlja kombinaciju tekstualnog i grafičkog načina predstavljanja algoritama. Osnove strukture se predstavljaju na sledeći način:

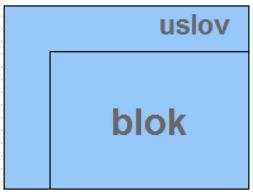
- Sekvenca



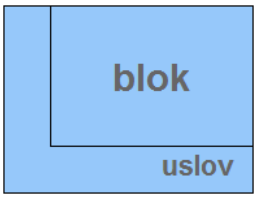
- Alternacija



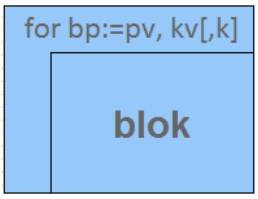
- while



- repeat-until

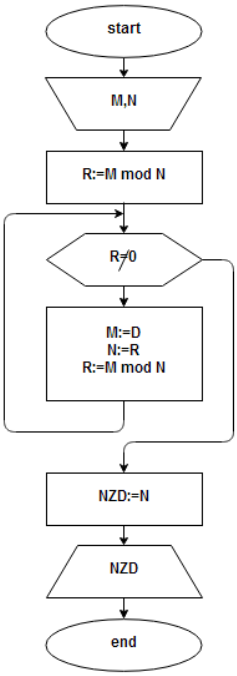


- for



Primer: Predstaviti Euklidov algoritam dijagramom toka, pseudo kodom i strukturalno.

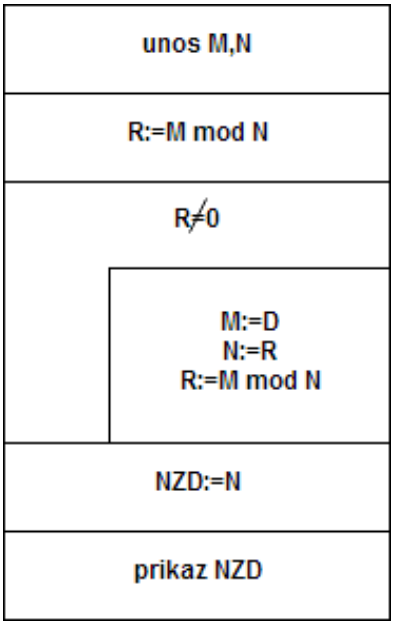
- Dijagram toka



- Pseudo kod

```
Start;
Unos M, N;
R:=M mod N;
while R≠0 do
 M:=N;
 N:=R;
 R:=M mod N;
enddo;
NZD:=N;
Prikaz NZD;
Kraj;
```

- Struktorogram



Promenljive, tipovi i strukture podataka

Promenljiva u programiranju predstavlja lokaciju u kojoj se privremeno čuvaju rezultati i međurezultati. Zbog jednostavnijeg pristupa i rada sa promenljivama, promenljive označavamo simbolički, slovima i brojevima. U većini programskih jezika postoji pravilo za imenovanje promeniljivih koje kaže da ime promenljive može biti bilo koji skup slova, cifara i znaka _ pod uslovom da ne počinje cifrom. Vrednost promenljivih može zadati korisnik unosom, može se dobiti kao rezultat izvršenja operacija ili se može zadati dodeljivanjem konstanti.

Primer:

```
op1:=10;
op2:=20;
R:=(op1 + op2)/2;
```

Svaka promenljiva pripada određenom tipu promenljivih. Tipom podataka određeno je:

- Skup vrednosti koje promenljiva može uzeti
- Skup operatora, odnosno operacija koje se mogu primeniti
- Memorijska reprezentacija

Tipovi podataka se dele na osnovne i složene(strukturne) tipove.

Osnovni tipovi

U osnovne tipove podataka spadaju numerički, logički i znakovni.

Numerički tipovi

U numeričke tipove spadaju:

1. Celobrojni tipovi
2. Realni tipovi
3. Kompleksni tipovi

Podaci **celobrojnog tipa** mogu uzimati celobrojne vrednosti iz dva skupa. Prvi skup je skup pozitivnih celih brojeva a drugi je skup celih brojeva. Opseg vrednosti zavisi od broja bitova (m) za reprezentaciju broja i krece se od $-(2^{m-1})$ do $2^{m-1} - 1$ za cele brojeve i od 0 do $2^m - 1$ za pozitvine cele brojeve. Skup operacija koje se mogu izvršiti jeste {+,-,*,/,%}. Rezultat izvršenja operacije nad celobrojnim tipom je celobrojni tip. Kod **realnog tipa** skup vrednosti je skup realnih brojeva. Kod memorijske reprezentacije pamti se mantisa i eksponent, a brojevi se predstavljaju kao $m*b^e$. Realni brojevi se u memoriji pamte u normalizovanom obliku za koji važi $1/b \leq m < 1$. Obično je m+e = 16, 32, 64… skup operatora je {+,-,*,/}. Skup vrednosti **kompleksnog tipa** su kompleksni brojevi koji se u memoriji predstavljaju pomoću dva realna broja. U nekim jezicima kompleksni tip nije podržan kao osnovni tip podataka (C). kod ovih jezika kompleksan tip je moguće kreirati kao složeni tip.

Logički tipovi

Skup vrednosti promenljivih ovog tipa jeste {TRUE, FALSE}. Postoje dva tipa memorijske reprezentacije. Kod prvog se za netačno uzima vrednost 0, a za tačno 1, dok se kod drugog netačno predstavlja kao 0, a tačno kao bilo koja vrednost rezličita od 0. Programski jezik C koristi drugu grupu. Operacije koje se koriste jesu logičke operacije {i, ili, ne}.

Znakovni tipovi

Znakovni tip podataka se koristi za predstavljanje tekstualnih podataka. Vrednosti koje mogu uzimati promenljive ovog tipa su velika i mala slova abecede, decimalne cifre, skup specijalnih znakova… Karakteri znakovnog tipa u memoriji se predstavljaju brojevima. Svakom karakteru odgovara jedan broj i ti brojevi su definisani ASCII tabelom. ASCII tabela, njen deo, predstavljen je na sledeći način:

rb.	karakter	kod
	…	
	0	48
	1	49
	2	
	…	
	A	65
	B	66
	…	
	a	97
	b	98

Operacije koje se mogu izvršavati su poređenje i konkatenacija (nadovezivanje).

Primer:

"A" < "B", "aab" < "abc", "A" # "B" = "AB", "aab" # "abc" = "aababc"

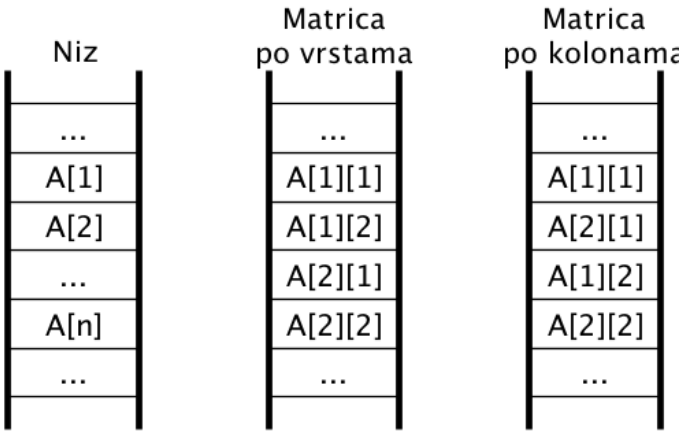
Što se algoritama tiče, tip promenljive biće određen prvom dodelom vrednosti toj promenljivoj. Što se programskih jezika tiče većina zahteva eksplicitnu deklaraciju tipa.

Složeni (strukturni) tipovi

Strukturni tipovi podataka se dobijaju kombinovanjem osnovnih tipova u složenije strukture. Te strukture mogu biti linearne i nelinearne. U linearne strukture spadaju polja, linearne lančane liste, magacini i redovi, a u nelinearne spadaju stabla i grafovi.

Polja

Polje je linearna homogena struktura koju čine više osnovnih podataka istog tipa. Polju se daje jednistveno ime u skladu sa prethodno definisanim pravilima, a pojedinim elementima polja se pristupa pomoću indeksa. Jednodimenziona polja su nizovi, a dvodimenziona su matrice. Elementi polja se smeštaju u memoriju u sukcesivnim memorijskim lokacijama. Za smeštanje dvodimenzionalnih i višedimenzionalnih polja u memoriju koristi se potupak koji se naziva linearizacija polja. Linearizacijom se recimo matrice mogu smestiti u memoriju po vrstama ili po kolonama.



Kod linearnih struktura svaki element osim prvog i poslednjeg ima svog prethodnika i svog sledbenika. Kod nelinearnih struktura te relacije su višestruke.

Sortiranje elemenata niza

Selection sort

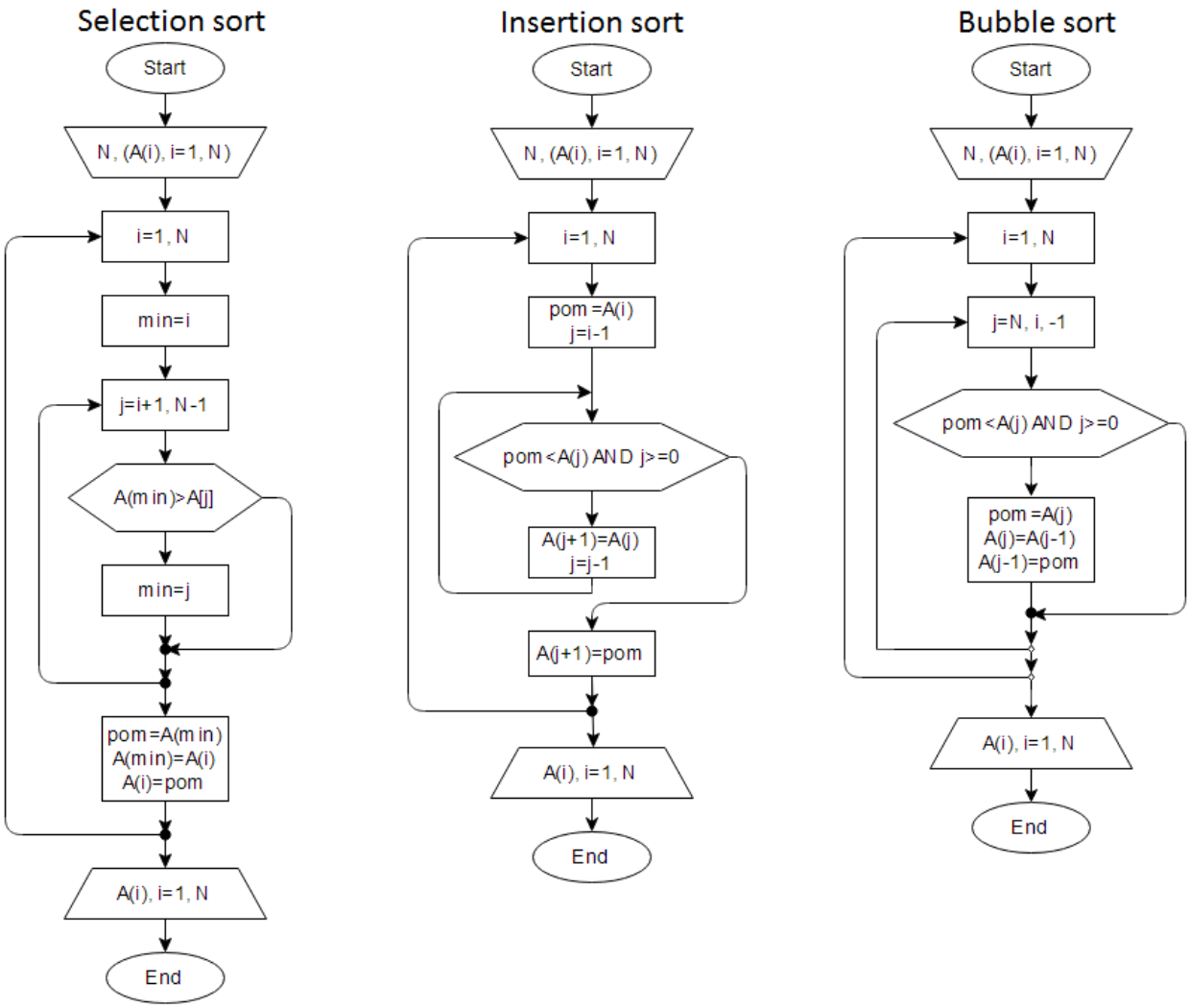
Kod sortiranja elemenata niza selekcijom ideja je da se u prvom koraku minimalni element stavi na početak niza. U drugom koraku minimalni element nesortiranog dela niza dolazi na drugu poziciju i sve tako redom do kraja niza.

Insertion sort

Kod sortiranja umetanjem za svaki element niza traži se pozicija u levo, počev od tog elementa, gde se taj element treba naći u sortiranom nizu.

Bubble sort

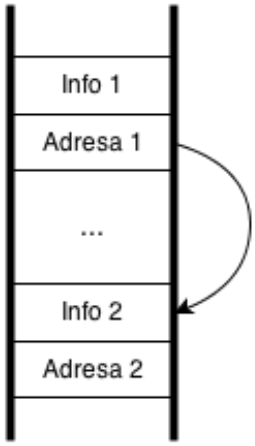
Ideja kod ovog sortiranja je ta da se uvek posmatraju dva susedna elementa. Ukoliko redosled elemenata nije odgovarajući, elementi razmenjuju mesta. U prvoj iteraciji provera ide sleva u desno do kraja niza nakon čega se dobija delimično uređen niz. Postupak se ponavlja sve dok se ne dobije uređen niz.



Linarne lančane liste

Linearne lančane liste spadaju u grupu linearnih struktura zato što svaki element liste (osim prvog i poslednjeg) ima jednog prethodnika i jednog sledbenika. Za razliku od nizova elementi linearnih lančanih lista ne moraju biti u sukcesivnim memorijskim lokacijama. Da bi se ovakva lista implementirala pored vrednosti svakog elementa potrebno je pamtiti i adresu sledećeg. Sama lančana lista ima sledeći oblik:

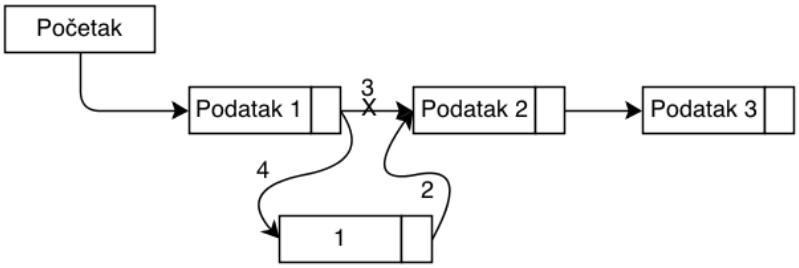
Memorijska interperatacija lančane liste je sledeća:



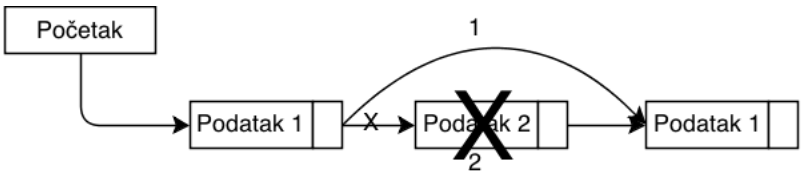
Karakteristika lančanih lista je to što da bi se pročitao podatak na i-toj poziciji potrebno je pristupiti prethodnim i-1 elementima.

Nad lančanim listama moguće je vršenje dve operacije i to:

1. Dodavanje elementa u listu:

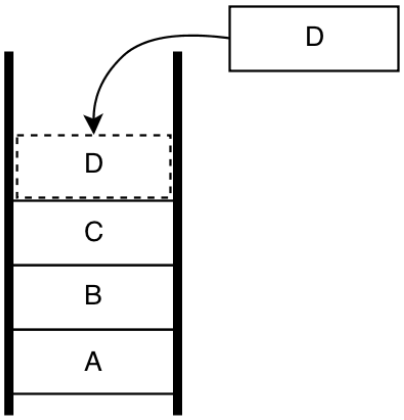


2. Brisanje elementa iz liste



Magacin

Magacin je linearna struktura koja se sastoji od elemenata istog tipa na sukcesivnim memorijskim lokacijama. Za razliku od nizova, kod kojih je moguće u bilo kom trenutku pristupiti bilo kom elementu niza, kod magacina je moguć pristup samo elementu na vrhu magacina. Podatak koji se upisuje se upisuje na vrh magacina, a podatak koji se čita, čita se sa vrha magacina. Pri čitanju podatak se briše iz magacina. Ovakve strukture se nazivaju LIFO strukturama (last in first out).



- Upis u magacin

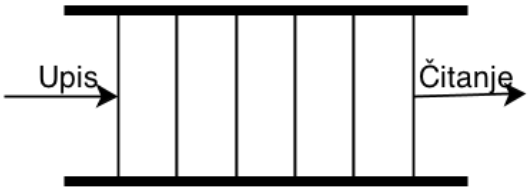
```
if(vrh<N) then
    M(vrh)=podatak;
    vrh=vrh+1;
endif;
```

- Čitanje iz magacina

```
if(vrh>0) then
    podatak=M(vrh);
    vrh=vrh-1;
endif;
```

Red

Redovi su linearne strukture kod kojih su svi elementi istog tipa i nalaze se u sukcesivnim memorijskim lokacijama. Za razliku od magacina, kod redova se upis vrši sa jedne strane, a čitanje sa druge. Ovakve strukture se nazivaju FIFO strukturama (first in first out).



Nelinearne strukture

Kod nelinearnih struktura nelineranost se ogleda u tome što jedan element može imati više prethodnika i više sledbenika.

Stabla

Stabla su nelinarne strukture kod kojih svaki element ima jednog prethodnika i više sledbenika, sem korena koji nema prethodnike. Kod stabla su elementi grupisani u nivoe i važi strogo pravilo da sledbenik mora biti u narednom nivou. Karakteristika stabla je da ne postoje zatvorene putanje.

Grafovi

Grafovi su nelinearne strukture kod kojih svaki element može imati više prethodnika i više sledbenika. Graf se u memoriji može predstaviti na dva načina.

Razvoj programskog jezika C

Programski jezik C je nastao u periodu između 1969. i 1973. godine. Za tvorca programskog jezika C smatra se Dennis Ritchie. Jazik C je razvijen za potrebe OS UNIX na platformi PDP7 u Bell-ovim laboratorijama. Tim programera, Dennis Ritchie, Brian Kernighan i Ken Thompson, razvio je programski jezik C da bi OS UNIX, koji je inicijalno pisan na asembleru za PDP7, preveli za platformu PDP11. Po Denisu Ritchie-u, godina kada je nastao programski jezik C jeste 1972. godina. Danas je jezik C široko prihvaćen, a na sintaksi jezika C zasnovani su mnogi jezici kao što su C++, C#, Java, PHP, JavaScript, J2ME…

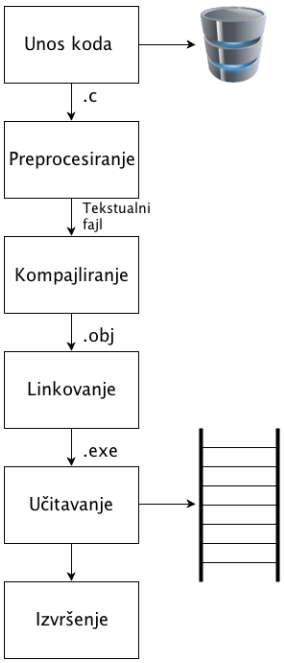
1978. godine Dennis Ritchie i Brian Kernighan napisali su knjigu „The C programming language" kada je C prvi put prikazan širokoj javnosti. 1983. godine počela je standardizacija jezika C, a 1989. godine je izdata verzija ANSI C tj. C89. Godinu dana nakon toga ISO (International Organization for Standardization) je izdala svoju verziju C90. Dopuna standarda izvršena je 1999. godine - C99. Bitnije izmene u odnosu na prethodne verzije odnose se na tipove podataka. Poslednja dopuna standarda izvršena je 2011. godine - C11. U ovoj verziji je dodata podrška za paralelno programiranje.

Programski jezik C nastao je na osnovu programskog jezika B, dok je programski jezik B nastao iz programskog jezika BCPL pojednostavljenjem jezičkih konstrukcija. Izmenama jezika B, C je postao jezik sa veoma jednostavnom sintaksom i kratkim sintaksnim konstrukcijama.

Karakteristike jezika C

1. Kratke sintaksne konstrukcije - Tim programera koji je razvio jezik C nije imao inicijalnu pretenziju da jezik C bude široko prihvaćen. Zahtev je bio što kraći zapis. Zbog ovoga jezik C u nekim slučajevima može biti izuzetno teško čitljiv.
2. Blizak asembleru - Po nekim konstrukcijama jezik C je blizak asemblerskom jeziku. U jeziku C je moguće definisati da li će se promenljiva pamtiti u memoriji ili u registru procesora.
3. Brzi programi - Programi pisani na programskom jeziku C su veoma brzi.
4. Mali broj ključnih reči - Jezik C ima relativno mali broj ključnih reči.
5. Složen - Operatorski jezik programskog jezika C je veoma bogat izrazima i može biti veoma složen.
6. Case-sensitive - Jezik je case-senzitivan, razlikuje velika i mala slova.
7. Slaba tipizacija - Jezik C ima slabu tipizaciju - U okviru istog izraza mogu figurisati promenljive različitog tipa. Kod jezika sa jakom tipizacijom eksplicitno se mora konvertovati tip.
8. Razvijem sistem funkcija - Poseduje razvijen sistem funkcija. Po sintaksi je i glavni program funkcija.

Faze u implementaciji programa na jeziku C



Nakon unosa koda u bilo kom editoru teksta dobija se kod u većini slučajeva sa ekstenzijom .c. Nakon toga slede faze prevođenja programa u izvršnu verziju. Prva faza prevođenja jeste preprocesiranje kojim se vrši razrešavanje preprocesorskih direktiva (#define, #include, #ifdef). Nakon toga sledi faza kompajliranja čime se od izvornog koda formira objektni kod. Objektni kod nije moguće izvršiti zato što u ovoj fazi nije povezan sa operativnim sistemom da bi ga operativni sistem mogao učitati. Ovu funkciju obavlja linker (povezivač). Nakon linkovanja dobija se izvršni fajl koji se može učitati i izvršiti.

Unos koda se može vršiti u bilo kom editoru teksta, a prevođenje u izvršnu verziju kompajlerom za konkretnu platformu. Za ceo proces implementacije programa postoje tzv. integrisana razvojna okruženja (IDE). Ova okruženja nude podršku za sve faze počev od unosa koda za koji vrše funkciju takozvanog bojenja teksta čime se jasno drugim bojama predstavljaju različiti sintaksni elementi.

Azbuka jezika C

Azbuku programskog jezika C čine mala i velika slova, cifre i specijalni znakovi.

Jezik za definisanje sintakse

Postoji puno načina na koji se može definisati sintaksa jezika. Jedan od njih je:

<pojam>:=<definicija>

Pojam koji se definiše može se koristiti za definiciju pojma. U okviru definicije mogu se naći drugi pojmovi ili specijalne oznake. Bilo šta napisano između [ ] uzima se kao opciona vrednost. Između { } navode se mogućnosti od kojih se može izabrati jedna. Mogućnosti se odvajaju uspravnom crtom |.

Tokeni jezika C

Token nekog jezika predstavlja elementarnu logičku celinu sastavljenu od elemenata azbuke nekog jezika. Tokeni mogu biti:

1. Identifikatori
2. Ključne reči
3. Separatori
4. Literali
5. Konstante
6. Operatori

Faza kompajliranja nekog programa deli se na dva dela:

1. Leksička analiza
2. Sintaksna analiza

Ulaz u leksičku analizu je kod programa, a izlaz niz tokena. Na osnovu ovog niza tokena sintaksni analizator kaže da li je konstrukcija sintaksno ispravna ili ne.

1. Identifikatori

Identifikatori se koriste za imenovanje promenljivih, funkcija, simboličkih konstanti, korisničkih tipova podataka i labela. Identifikator određuje programer tako da:

1. Identifikator nije u skupu ključnih reči;
2. Predstavlja bilo koji niz velikih i malih slova, cifara i znaka _ koji ne počinje cifrom.

Kod ranijih verzija kompajlera dužina identifikatora je bila ograničena na 31 znak ukoliko se koristi u fajlu u kojem je definisan i na 6 znaka ukoliko se koristi u eksternim fajlovima. Kod današnjih kompajlera uglavnom ne postoji ograničenje.

2. Ključne reči

auto	const	double	float	int	short	struct	unsigned
break	continue	else	for	long	signed	switch	void
case	default	enum	goto	register	sizeof	typedef	volatile
char	do	extern	if	return	static	union	while

3. Separatori

Separotori u programskom jeziku C su:

- { - početak bloka
- } - kraj bloka
- ; - kraj naredbe (ceo program se može napisati u jednom redu)
- (- početak liste parametara
-) - kraj liste parametara
- // - linijski komentar, deo koda koji kompajler preskače. Koriste se u cilju dokumentovanja programa tako što se dodaju tekstualni opisi delova koda zbog kasnijeg lakšeg snalaženja
- /* */ - blokovski komentar.

4. Literali

Literali su konstante nizova karaktera. Pišu se između navodnika.

"primer":= 'p' , 'r' , 'i' , 'm' , 'e' , 'r' , '\0'

5. Konstante

Konstante mogu biti celobrojne, realne i znakovne.

1. Celobrojne konstante - mogu biti dekadne, oktalne i heksadecimalne. Mogu biti bilo koji niz cifara iz opsega tipa. Ukoliko je 0 prva cifra radi se o oktalnoj konstanti. Heksadecimalne konstante imaju prefiks 0x. Celobrojne konstante mogu imati prefiks + ili - koji određuje znak konstante. Takođe, mogu imati sufiks l ili L (long), u ili U (unsigned).
2. Realne konstante - mogu biti zapisane u fiksnom ili u pokretnom zarezu:

   [+ | -][<celobrojni deo>].<razlomljeni_deo>
   [+ | -]<mantisa>{E | e}[+ | -].<eksponent>

   Mantisa i eksponent su celi brojevi (realne konstante u fiksnom zarezu).
3. Znakovne konstante su simboli iz ASCII tabele koji se pišu između apostrofa. Kod napisanog simbola se pamti u memoriji. Postoje i specijalni simboli koji se mogu koristiti:
   - '\n' - prelazak u novi red
   - '\t' - tabulacija
   - '\v' - vertikalna tabulacija
   - '\b' - backspace
   - '\f' - prelazak na novu stranu
   - '\'' - apostrof

6. Operatori

Operatori se u programskom jeziku C koriste kao oznaka za operaciju koju je potrebno izvršiti nad operandima. Po broju operanada operatori se dele na unarne, binarne i ternarne. Po funkcionalnosti se dele na:

- aritmetičke operatore
- relacione operatore
- logičke operatore
- operatore za rad sa bitovima
- operator dodele vrednosti
- sizeof, coma i cast operatore
- operator grananja
- operator referenciranja i dereferenciranja
- operator za pristup elementima polja i struktura

Deklaracija promenljivih

Deklaracija promenljivih je jezička konstrukcija kojom se u memoriji rezerviše prostor za pamćenje promenljivih. Deklaracija se definiše na sledeći način:

<deklaracija_prom>:=<tip><identifikator1>[=<konst_1>[,<identifikator2>[=<konst_2>...]]

Primer:

```
int x;  // deklaracija
int y=5; // inicijalizacija prilikom deklaracije
```

Tipovi podataka

U programskom jeziku C postoji podrška za razne tipove podataka. Svi tipovi podataka su podeljeni na ceobrojne tipove podataka, realne tipove podataka, logičke tipove podataka i na stringove.

Celobrojni tip

Celobrojni tipovi podataka su char, short, int, long ispred kojih može stajati signed ili unsigned. Celobrojni tip se definiše na sledeći način:

<celobrojni_tip>:=[{signed | unsigned}]{char | short | int | long}

Realni tip

Realni tip se definiše na sledeći način:

<realni_tip>:={float | double}

Logički tip

Za predstavljanje logičkih promeljivih u programskom jeziku C koristi se celobrojni tip. Za netačnu vrednost uzima se nula dok se za tačnu vrednost uzimaju sve ostale vrednosti različite od nule.

Stringovi

U programskom jeziku C ne postoji podrška za stringove kao osnovne tipove podataka. Stringovi su nizovi karaktera koji se koriste za pamćenje tekstualnih podataka (rečenica) i u programskom jeziku C se definišu kao složeni tip.

Struktura C programa

Ulazna tačka u svaki C program, odnosno naredba od koje kreće izvršenje jeste funkcija main. Osnovna struktura ima izgled:

```
[preprocesorske direktive]
[definisanje funkcija]
main ()
{
    <telo_programa>
}
```

Telo programa sadrži definicije promenljivih, konstanti, tipova podataka i izvršne naredbe. Deklaracija konstanti se vrši na sledeći način:

const <tip> <vrednost>

Za razliku od promenljivih, konstantama nije moguće promeniti vrednost. Generalno, kompajler neće izvršiti rezervaciju memorijskog prostora za konstantu već će u procesu kompajliranja sva pojavljivanja konstante zameniti konkretnom vrednošću konstante.

Definicija novog tipa vrši se na sledeći način:

type def <tip> <novi_tip>

Izvršne naredbe mogu biti elementarne i upravljačke naredbe. Elementarne naredbe se završavaju ; i generalno ih nije neophodno pisati svaku u novi red.

Telo programa u ANSI C-u počinje naredbama za deklaraciju promenljivih i sve naredbe za deklaraciju promenljivih moraju se naći pre prve izvršne naredbe. Kod novijih kompajlera i kod C++ kompajlera naredbe za deklaraciju promenljive mogu se naći bilo gde u kodu. MS Visual Studio zavisno od ekstenzije fajla poziva različite kompajlere. Ukoliko fajl ima ekstenziju .c poziva se ANSI C kompajler, a ukoliko fajl ima .cpp ekstenziju poziva se C++ kompajler.

Operatori

Aritmetički operatori

U aritmetičke operatore spadaju sledeći operatori:

1. unarni operatori + i -
2. unarni operatori inkrementiranja i dekrementiranja ++ i --
3. binarni operatori *, / i %
4. binarni operatori + i -

Redosled po kome su navedeni operatori označava prioritet operatora počev od najvišeg. Prioritetom operatora određen je redosled izvršenja operacija ukoliko zagradama nije drugačije rečeno.

Primer:

```
main ()
{
    int x;
    x=3*2+4*8;
}
```

Unarni operatori + i - stoje sa leve strane identifikatora promenljive i određuju znak.

Unarni operatori inkrementiranja i dekrementiranja ++ i -- se mogu naći i sa leve i sa desne strane identifikatora promenljive. Operator ++ povećava vrednost promenljive za 1, dok operator -- smanjuje vrednost promenljive za 1. Ukoliko se ovi operatori nađu sa leve strane identifikatora promenljive prvo će se izvršiti oni, a nakon toga i ostatak izraza. Ukoliko se nađu sa desne strane identifikatora promenljive prvo će se izvršiti celokupan izraz, pa tek onda i operatori ++ ili --.

Primer:

y=x++; ⇔ y=x; x=x+1;

Kod binarnih operatora *, / i % tip operatora je određen tipom operanada. Tako rezultat deljenja može biti celobrojni podatak ili realni podatak. Ukoliko operandi nisu istog tipa, kompajler će pre izvršenja operacija sve operande implicitno svesti na isti tip pri čemu važe sledeći prioriteti:

char < short < int < long < float < double

Relacioni operatori

Relacioni operatori su binarni operatori čiji su operandi promenljive ili izrazi, a rezultat je logička vrednost. Relacioni operatori su:

1. operatori <, >, <= i >=
2. operatori == i !=

Logički operatori

1. ! - unarni operator za negaciju, stoji sa leve strane identifikatora
2. && - binarni operator logičko I
3. || - binarni operator logičko ILI

Operatori za rad sa bitovima

1. ~ - logička negacija svih bitova bit po bit
2. & - binarni operator koji vrši AND operaciju između bitova na odgovarajućim pozicijama
3. | - binarni operator koji vrši OR operaciju između bitova na odgovarajućim pozicijama
4. ^ - binarni operator koji vrši XOR operaciju između bitova na odgovarajućim pozicijama
5. << - binarni operator koji vrši pomeranje sadržaju u levo
6. >> - binarni operator koji vrši pomeranje sadržaju u desno

Operator dodele

Operator dodele definiše se na sledeći način:

<dodela> := [<op>] = <op> := {+ | - | * | / | % | ^ | | | & | >> | <<}

Operator dodele je složen operator kod koga ispred znaka jednakosti (simbol dodele) se može naći neki drugi operator.

Primer:

x+=5; ⇔ x=x+5;
x*=2; ⇔ x=x*2;

Operator grananja

Operator grananja je ternarni operator koji u zavisnosti od rezultata izraza izvršava jedan ili drugi deo naredbe. Definiše se na sledeći način:

```
y = (<uslov>) ? <naredba_1> : <naredba_2>;
```

Primer:

```
y = (x==2) ? y=1 : y=8;  //Ukoliko je x=2 onda je y=1, a u svim drugim slučajevima je y=8.
```

„sizeof“ operator

sizeof je ključna reč i predstavlja unarni operator koji se može naći sa leve strane promenljive ili tipa. Vraća broj bajtova koji promenljiva, odnosno tip, zauzima u memoriji.

Primer:

```
int x, y;
y=sizeof(x); // y=4, y=sizeof(int)
```

„comma“ operator

Ovim operatorom se mogu razdvojiti nezavisni izrazi u okviru jedne naredbe.

Primer:

```
a=2, b=4, c=1;
```

„cast“ operator

Cast operator je unarni operator kojim se eksplicitno menja tip promenljive. Definiše se na sledeći način:

```
(<tip>)<promenljiva>
```

Primer:

```
int x=5, y=2;
float z;
z=x/(float)y;
```

Operatori referenciranja i dereferenciranja

1. & - operator referenciranja, unarni operator koji stoji sa leve strane identifikatora promenljive i vraća adresu memoriji na kojoj se promenljiva nalazi
2. * - operator dereferenciranja je unarni operator koji ukoliko stoji sa leve strane adrese vraća vrednost koja se nalazi u memoriji na zadatoj adresi

Prioritet operatora

1. ( ), [ ]
2. unarni operatori +, -, ++, --, ...
3. binarni operatori *, /, %
4. binarni operatori +, -
5. <<, >>
6. <, >, >=, <=
7. ==, !=
8. &
9. ^
10. |
11. &&
12. ||
13. ? :
14. =, [<op>]=
15. ,

Jedino su operatori pod 2. i pod 14. asocijativni s desna u levo, a svi ostali s leva u desno. Asocijativnost određuje koji će se operator prvi izvršiti ukoliko operatori imaju isti prioritet.

Naredbe za ulaz i izlaz

Programski jezik C nema naredbe za ulaz i izlaz. Da bi se koristio ulaz i izlaz potrebno je preprocesorskim direktivama uključiti dodatne biblioteke funkcija koje sadrže funkcije za ulaz i izlaz. Kompajler za C dolazi sa velikim brojem standardnih biblioteka. Svaka biblioteka je poseban fajl sa ekstenzijom .h (header file). Da bi se uključio neki header fajl koristi se preprocesorska direktiva #include. Header fajl koji sadrži ulaz i izlaz je stdio.h. Preprocesorske direktive se navode na samom početku programa. Naziv header fajla se može pisati između „ „ i između < >. Razlika je samo u skupu putanja na disku gde će kompajler tražiti biblioteku.

Standardni ulaz (unos vrednosti)

Sintaksa funkcije koja se koristi za standardni ulaz je:

scanf (<format>, <lista_adresa_promenljivih>);

Format je literal koji sadrži konverzione karaktere, odnosno pojedinačne ulazne konverzije. Svakoj promenljivoj iz liste adresa promenljivih odgovara po jedna pojedinačna ulazna konverzija koja određuje u kom formatu će korisnik zadati vrednost.

Sintaksa pojedinačne ulazne konverzije je sledeća:

%[w][h | l | L]<tip_konverzije>

Prilikom izvršenja funkcije scanf program će se blokirati i neće se nastaviti izvršenje sve dok korisnik ne unese vrednost.

Standardni izlaz (prikaz rezultata)

Sintaksa funkcije koja se koristi za standardni izlaz je:

printf (<format>[, <izraz_1> [, <izraz_2> [, ...]]);

Format je literal koji sadrži konverzione karaktere, odnosno pojedinačne izlazne konverzije. Ova funkcija radi tako što prikazuje niz karaktera zadat literalom, a na mesto pojedinačnih izlaznih konverzija prikazuje vrednosti izraza ili promenljivih. Sintaksa pojedinačne izlazne konverzije je sledeća:

%[-][+ |][#][w[.d]][h | l | L]<tip_konverzije>

- [-] - ukoliko je naveden, ispis se vrši tako da je vrednost poravnata uz levu ivicu dela predviđenog za prikaz
- [+] - prikaz znaka vrednosti
- [ ] - za pozitivne vrednosti na poziciji gde bi se trebalo naći znak + ispisuje blanko
- [#] - ispisuje oznaku brojnog sistema za celobrojni iskaz
- [w] - označava maksimalan broj karaktera dozvoljen za unos ili broj mesta predviđenih za prikaz
- [.d] - broj mesta
- h -konverzija u short int
- l - konverzija u long int
- L - konverzija u long double

Tipovi konverzije su sledeći:

- %d - format vrednosti je dekadni broj
- %u - neoznačeni ceo broj
- %o - oktalni ceo broj
- %x - heksadecimalni ceo broj
- %i - ceo broj određen na način na koji ga je korisnik uneo
- %f - realni broj u fiksnom zarezu
- %e - realni broj u pokretnom zarezu
- %g - realni broj u fiksnom ili pokretnom zarezu u zavisnosti od korisnika
- %c - karakter podaci
- %s - stringovi

Kontrola toka programa

Za kontrolu toka programa koriste se :

1. sekvenca - može se implementirati blokom naredbi
2. grananja - if, if else, switch i bezuslovne naredbe skoka break, continue, goto
3. petlje - for, while i do while

Blok naredbi

Sekvenca naredbi koja počinje znakom { i završava se znakom }.

Naredba „if“

Sintaksa if petlje je sledeća:

if({<uslov> | <izraz>})

{<naredba> | <blok_naredbi>}

Ukoliko struktura sadrži samo jednu naredbu nije neophodno pisati { i }. Ukoliko sadrži više naredbi, naredbe se pišu u vidu bloka.

Naredba „switch“

Sintaksa naredbe switch je sledeća:

```
switch (<izraz>)
{
    case <const1>:<blok1>;
    case <const2>:<blok2>;
    ...
    [default:<blokD>;]
}
```

Switch ne spada u osnovne naredbe jezika C. Može se implementirati višestrukim korišćenjem naredbe if. U zavisnosti od vrednosti izraza, ukoliko je vrednost izraza jednaka const1 izvršiće se blok 1 i svi ostali blokovi. Ukoliko je vrednost izraza jednaka const2, izvršenje počinje od bloka 2 i izvršavaju se svi ostali blokovi...

Naredbe bezuslovnog skoka

break

Prekida izvršenje trenutnog bloka i izvršenje se nastavlja od prve naredbe iza tog bloka. Ovo je nestrukturna naredba.

continue

Kao i break prekida izvršenje trenutnog bloka, ali se u slučaju petlje prekida samo unutrašnja iteracija i prelazi se na početak petlje.

goto

Vrši se bezuslovni skok na bilo koji deo programa.

Primer:

```
goto <labela>;
```

Labela je oznaka linije koda na koju treba skočiti. Piše se ispred linije, levo od naredbe i iza labele stoje : . goto je nestrukturna naredba.

switch-break

Ova struktura se dobija dodavanjem naredbe break u svakom bloku.

„for“ petlja

Petlje omogućavaju višestruko izvršavanje naredbe ili bloka naredbi. Sintaksa for petlje je sledeća:

```
for(<izraz1>; <izraz2>; <izraz3>)
    {<naredba> | <blok_naredbi>}
```

Broj izvršenja for petlje jednak je [(kv-pv)/k]+1.

Prvim izrazom u for petlji vrši se inicijalizacija petlje. Drugi izraz predstavlja uslov za završetak, dok treći izraz predstavlja promenu brojača za svaku iteraciju.

„while“ petlja

Sintaksa while petlje je sledeća:

```
while(<uslov>)
    {<naredba> | <blok_naredbi>}
```

„do - while“ petlja

Sintaksa do - while petlje je sledeća:

```
do {<naredba> | <blok_naredbi>}
```

```
while(<uslov>);
```

Polja

Polja su homogene strukture podataka kod kojih su svi elementi istog tipa. Zajedničko za sve elemente je ime (naziv) polja, a pojedinačnim elementima je moguće pristupiti kombinovanjem imena i indeksa polja. Indeks je pozicija elementa u polju.

Jednodimenziona polja

Deklaracija jednodimenzionog polja se vrši na sledeći način:

```
<tip> <identifikator>[<veličina>];
```

Tip polja je zajednički tip za sve elemente polja. identifikator je naziv polja i veličina predstavlja broj elemenata polja. Prvi element polja ima indeks 0, a poslednji N-1 gde je N dimenzija polja.

Primer:

```
main ()
{
    int A[100];
    float B[50];
    A[0]=1;
    A[1]=2;
    ...
    A[99]=100;
    ...
}
```

Ime polja nevedeno bez indeksa predstavlja adresu prvog elementa polja.

Višedimenziona polja

Deklaracija višedimenzionog polja se vrši na sledeći način:

```
<tip> <identifikator>[<dimenzija1>][<dimenzija2>][<dimenzija3>]...[<dimenzijaN>];
```

Prvi element polja ima sve indekse 0, dok poslednji ima sve indekse N-1.

Inicijalizacija polja

Inicijalizacija polja je proces dodele vrednosti elementa. Inicijalizacija se može izvršiti odmah nakon deklaracije.

Primer:

```
main ()
{
    int A[6]={3, 8, 1, 0, 15, 4};
    int B[ ]={5, 0, 3, 4, 1}; //Kompajler će sam preprojati elemente.
    char R[ ]={'p', 'r', 'i', 'm', 'e', 'r'};
    char P[ ]="Primer";
}
```

Kod višedimenzionih polja prilikom inicijalizacije može se izostaviti samo prva dimenzija.

Dekompozicija i funkcije u C-u

U cilju pojednostavljenja rešavanja problema na računaru često se vrši dekompozicija problema čime se problem razlaže na manje i lakše (jednostavnije) probleme. Ukoliko se određena programska celina često javlja u programu, programski jezici nude podršku da se taj deo koda izdvoji i poziva na izvršenje iz bilo kog dela programa gde je to potrebno. Dekompozicijom programa i uvođenjem funkcija struktura programa se može predstaviti na sledeći način.

Definicija i deklaracija funkcije

Funkcija se u programskom jeziku C može definisati navođenjem zaglavlja i tela funkcije bilo pre ili posle funkcije main.

ANSI C

```
<tip_funkcije><identifikator>(<lista_fiktivnih_parametara>)
<definicija_parametara>
{
    <deklaracija_lokalnih_promenljivih>
    <telo_funkcije>
}
```

C99 / C++

```
<tip_funkcije><identifikator>(<definicija_parametara>)
{
    <telo_funkcije>
}
```

Tip funkcije

Tip funkcije je tip podatka koji funkcija vraća glavnom programu. Kod tipa funkcije može se napraviti analogija između funkcija u programiranju i matematici. Funkcija matematički data na način y=f(x) znači da se x uzima kao parametar, vrši se neko izračunavanje nad njim i rezultat se dodeljuje y. U našem slučaju tip funkcije jednak je tipu promenljive y.

<tip_funkcije>:={int | float | double | char | void}

Ukoliko funkcija ima tip void, funkcija ne vraća ništa.

Identifikator

Identifikator je naziv funkcije.

Lista fiktivnih parametara

<lista_fiktivnih_parametara>:=<ident.1>[<ident.2>]...[<ident.N>];

Parametri funkcije su podaci koje funkcija dobija od glavnog programa i nad kojima se vrši obrada.

Definicija parametara

Definicija parametara je navođenje tipa svakog pojedinačnog parametra.

<definicija_parametra>:=<tip><ident.1>[,<tip><ident.2>]...[,<tip><ident.N>];

Poziv funkcije

Ukoliko postoji definicija funkcije u programu a funkcija ni jednom nije pozvana, funkcija se nikad neće izvršiti. Pozivom funkcije izvršenje se prenosti na taj deo koda i funkciji se predaju stvarni parametri. Poziv se vrši na sledeći način:

[<promenljiva>=]<identifikator>(<lista_stvarnih_parametara>);

Promenljiva se koristi za smeštanje rezultata koje funkcija vraća. Identifikator je ime funkcije, a lista stvarnih parametara predstavlja skup realnih podataka nad kojima funkcija vrši obradu.

Primer:

```
#include <stdio.h>
int max(a,b) //a i b su fiktivni parametri.
    int a; int b; //Definicija parametara.
{
    int pom; //Lokalna promenljiva.
    if(a>b)  //Telo funkcije.
            pom=a;
    else
            pom=b;
```

```
    return pom; //Povratna vrednost.
}
main ()
{
    int x=8, y=10, z;
    z=max(x,y); //Poziv funkcije. x i y su stvarni parametri.
    printf("%d", z);
}
```

Primer:

```
#include <stdio.h>
int max(int a, int b); //Deklaracija funkcije.
main ()
{
    int x=8, y=10, z;
    z=max(x,y);
    printf("%d", z);
    z=max(25, 4);
}
int max(int a, int b) //Definicija funkcije.
{
    int pom;
    if(a>b)
            pom=a;
    else
            pom=b;
    return pom;
}
```

Prilikom kompajliranja kompajler kreće od prve linije koda i ide redom, u našem primeru do poziva funkcije. U tom trenutku, ukoliko funkcija do tada nije bila deklarisana, kompajler će prijaviti grešku da je funkcija nedefinisana zato što se definicija nalazi iza poziva. U ovakvim slučajevima pre glavnog programa potrebno je navesti deklaraciju funkcije. Deklaracije se još naziva i prototip funkcije. Za deklaracijom nema potrebe ukoliko je definicija pre poziva. Ukoliko dve funkcije imaju iste identifikatore, a različite parametre, C će ih tretirati kao nezavisne funkcije.

Naredba „return“

Naredba return se može naći bilo gde u funkciji. Izvršenjem ove naredbe prekida se funkcija i upravljanje vraća glavnom programu. Ukoliko naredba return ima parametre, ta vrednost se vraća glavnom programu. Naredba return može vretiti najviše jedan skalarni podatak.

```
return[{<promenljiva> | <izraz> | <konst>}];
```

Prenost parametara po vrednosti

Primer:

```
#include <stdio.h>
int f(int a)
{
    int pom;
    pom=a+1;
    a=a+2;
    return pom;
}
main ()
{
    int x, y;
```

```
    x=10;
    y=f(x);
    printf("%d\n %d", x, y);
}
```

U trenutku poziva funkcije u memoriji se kreira nova promenljiva a (parametar funkcije) i odmah nakon kreiranja vrednost stvarnog argumenta x se kopira u novu promenljivu a. Ovo se zove prenos parametara po vrednosti. Vrednost te promenljive se može menjati proizvoljno u funkciji. Nakon izvršenja funkcije oslobađa se memorijski prostor koji su zauzeli fiktivni parametri i lokalne promeljive.

Promena fiktivnih parametara u funkciji je dozvoljena ali nije vidljiva u glavnom programu. U pragramskom jeziku C, funkcija može vratiti samo jednu vrednost u slučaju prenosa parametra po vrednosti.

Prenost parametara po referenci

Da bi promena bila vidljiva u glavnom programu, fiktivnom argumentu i svim pojavljivanjima fiktivnog argumenta treba dodati operator dereferenciranja, a stvarnom argumentu operator referenciranja.

Primer:

```
#include <stdio.h>
int f(int *a)
{
    int pom;
    pom=*a+1;
    *a=*a+2;
    return pom;
}
main ()
{
    int x, y;
    x=10;
    y=f(&x);
    printf("%d\n %d", x, y);
}
```

Pomoću prenosa parametara po referenci funkcija pozivanjem programu može vratiti više vrednosti.

Parametri funkcije „main“

Osnovni gradivni element programa u programskom jeziku C je funkcija. Za razliku od drugih programskih jezika C ne poseduje mehanizam procedura. Koncept procedure je proširenje funkcija tako što funkcija može imati više ulaznih i više izlaznih parametara Procedure u programskom jeziku C ne postoje eksplicitno jer se ovaj problem rešava prenosom parametara po referenci. Program napisan na programskom jeziku C može imati proizvoljno mnogo funkcija. i glavni program u programskom jeziku C je predstavljen funkcijom. Naziv ove funkcije je main a sintaksa je sledeća:

```
[{void | int}] main ([int argc, char *argv[]])
{
    <telo_funkcije>
    [return[{0 | 1}]];
}
```

Ulazni parametri i povratna vrednost bilo koje funkcije služe za komunikaciju te funkcije sa glavnim programom, odnosno funkcijom koja je tu funkciju pozvala. U slučaju glavne funkcije (programa) ova komunikacija se obavlja između korisničkog programa i operativnog sistema. Operativni sistem preko parametra argc daje ukupni broj parametara koji se prenose, a preko niza stringova argv i vrednosti tih parametara. Povratna vrednost može biti celobrojna i u slučaju da program vrati nulu operativni sistem smatra da se program izvršio bez greške. U slučaju da je vrednost rezultata različita od nule, operativni sistem prikazuje korisniku poruku o grešci. Ukoliko se povratna vrednost ne koristi za tip funkcije se može navesti void. Navođenje tipa funkcije je obavezno počev od MS Visual Studio-a 2010.

Operativni sistem vrši prenos parametara na dva načina:

1. Konzolna aplikacija

   C:\mojprogram.exe parametar1 15 parametar2 <enter>

U ovom slučaju argc će biti 4, a program će prikazati sledeće:

mojprogram.exe
parametar1
15
parametar2

2. Windows OS

Drag & drop prenos parametara na ikonicu izvršnog programa.

Rekurzivne funkcije

Rekurzivne funkcije su klasa funkcija koje u telu funkcije pozivaju samu sebe. Pojam rekurzije je definisanje nekog pojma korišćenjem tog istog pojma. Svaka rekurzivna funkcija mora da ima dva dela, deo za završetak rekurzije i deo u kome se vrši obrada podataka. Broj poziva u okviru funkcije koji je potreban za izračunavanje vrednosti funkcije naziva se dubina rekurzije.

U slučaju da funkcija u okviru tela funkcije poziva samu sebe, pozivajuća instanca funkcije će se pauzirati a svi podaci neophodni za povratak biće smešteni na stek. Sledeći ugnježđeni poziv će takođe pauzirati trenutnu instancu i svi parametri će automatski biti upisani na stek. Rekurzija ide u dubinu dok se ne dođe do uslova za završetak rekurzije.

Standardne biblioteke u programskom jeziku C

Biblioteka funkcija u programskom jeziku C je bilo koji fajl koji ima ekstenziju .h i u kome se nalaze definicije funkcija. Umesto pisanja funkcija u istom fajlu sa glavnim programom, programer svoje funkcija može izdvojiti u poseban header fajl (<naziv_header_fajla>.h) i uključiti ga u glavni program preprocesorskom direktivom #include "<naziv_header_fajla>.h". Uz svaki kompajler dolazi veliki broj standardnih biblioteka. Najpoznatije su: stdio.h, math.h i string.h. Neke od funkcija ovih biblioteka su:

- stdio.h -> printf, scanf, fopen, fclose, fscanf, _inp, _outp, ...
- math.h -> abs, fabs, sin, cos, tg, log, pow, sqrt, ...
- string.h -> strlen, strcpy, strcmp, ...

Izvedeni tipovi podataka

Pored osnovnih tipova podataka, u programskom jeziku C postoje i izvedeni tipovi podataka. Izvedeni tipovi podataka izvode se iz osnovnih tipova i mogu biti:

1. Nizovi
2. Stringovi
3. Pokazivači
4. Strukture
5. Unije

Nizovi

Nizovi ili polja predstavljaju homogenu strukturu podataka kod koje su svi podaci istog tipa. Indeks prvog elementa je 0, a poslednjeg N-1, gde je N dimenzija niza.

Primer:

```
void main ()
{
    long A[100];
    A[0]=15;
    A[1]=16;
    A[2]=A[1]+A[0];
    ...
}
```

Stringovi

Stringovi su posebna klasa nizova kod kojih su svi elementi karakter podaci i koji se u programskom jeziku C završavaju specijalnim znakom '\0'.

Primer:

```
void main ()
{
    char R[]="Ovo je string.";
    if(R[1]=='v')
            printf("Drugo slovo je v.");
    else
            printf("Drugo slovo nije v.");
}
```

Pokazivači (pointeri)

Pokazivači spadaju u izvedene tipove podataka. U promenljivoj pokazivačkog tipa pamti se adresa nekog podatka u memorji, a pokazivač se izvodi iz osnovnog tipa podataka na koji pokazivač pokazuje. Deklaracija pokazivačkog tipa je sledeća:

```
<tip>* <identifikator>;
<tip> *<identifikator>;
```

Tip je tip osnovnog podatka na koji pokazivač pokazuje. * je oznaka da se radi o pokazivaču, a identifikator je naziv promenljive. Ukoliko * stoji uz tip odnosi se na sve identifikatore u naredbi. Ukoliko stoji uz promenljivu odnosi se samo na promenljivu.

Primer:

```
void main ()
{
    int* x,y; //I x i y su pokazivači.
    int *x,y; //Samo je x pokazivač.
}
```

Pokazivačka algebra

Nad promenljivama pokazivačkog tipa moguće je izvršavati aritmetičke operacije +, -, *, /, ++, --. Memorija je adresibilna na nivou bajta, međutim, dodavanjem jedinica na vrednost pokazivača (inkrementiranjem) vrednost pokazivača će se povećati za broj bajtova koji zauzima tip iz koga je pokazivač izveden. Tako će u slučaju integera inkrement povećati vrednost pokazivača za 4.

Primer:

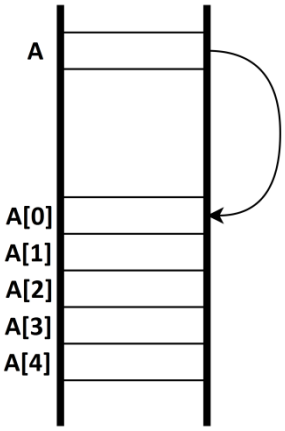
```
main ()
{
    int x,y; int *p;
    x=3; y=5;
    p=&x; *p=*p+1;
    p++; *p=*p+1;
}
```

Operatori inkrementiranja i dekrementiranja imaju veći prioritet u odnosu na operator dereferenciranja, što znači da (*p)++ nije isto što i *p++.

Nizovi i pokazivači

Prilikom deklaracije niza navodi se tip, ime niza i broj elemenata niza. U memoriji se tom prilikom odvaja sizeof(<tip>)*<broj_elemenata> bajtova plus jedan pokazivač koji pokazuje na početak niza, a čije je ime jednako imenu niza.

int A[5]:



*A ⇔ A[0]
*(A+1) ⇔ A[1]
*(A+2) ⇔ A[2]
...
*(A+i) ⇔ A[i]

Pokazivači i funkcije

Prilikom poziva funkcije za svaki fiktivni argument funkcije (parametri u definiciji funkcije) izdvaja se mesto u memoriji gde se kopira vrednost stvarnog argumenta sa kojim je funkcija pozvana. Ovo se naziva prenos parametara po vrednosti i praktično znači da se vrednost argumenta u funkciji može menjati, ali ta vrednost neće biti vidljiva u glavnom programu jer se radi sa kopijom. Ukoliko se prenosi pokazivač (adresa, referenca), promena vrednosti funkcije biće vidljiva i u glavnom programu.

Primer:

1. Prenos parametara po vrednosti

```
#include <stdio.h>
void zamena(int a, int b)
{
int pom;
pom=a;
a=b;
b=pom;
}
main ()
{
int x,y;
x=5; y=10;
zamena(x, y);
printf("%d %d\n\n", x, y);
}
```

2. Prenos parametara po referenci

```
#include <stdio.h>
void zamena(int *a, int *b)
{
int pom;
pom=*a;
*a=*b;
*b=pom;
}
main ()
{
int x,y;
x=5; y=10;
zamena(&x, &y);
printf("%d %d\n\n", x, y);
}
```

Prenos nizova preko argumenta funkcije vrši se uvek po referenci tako što se prenosi pokazivač na prvi element niza.

NULL pointer

NULL je specijalna konstanta u programskom jeziku C čija je numerička vrednost nula (0). Ova konstanta se obično dodeljuje pokazivačima čime se naglašava da pokazivač ne pokazuje ni na šta.

Primer:

```
main ()
{
    int *p;
    p=NULL;
}
```

Ukoliko se u programu vrši dereferenciranje pokazivača sa vrednošću NULL, kompajler neće prijaviti grešku već će doći do greške pri izvršenju programa zbog toga što memorijska lokacija sa adresom 0 nije u vlasništvu programa već operativnog sistema.

Strukture

Strukture su izvedeni tipovi podataka koji se sastoje od jednog ili više osnovnih podataka. Strukture su korisnički definisani tipovi podataka a koriste se za grupisanje podataka koji su u nekoj sematičkoj vezi. Sintaksa deklaracije strukture je sledeća:

```
struct <naziv> <identifikator> [, <identifikator2>…];
```

Sintaksa definicije strukture je seldeća:

```
struct <naziv_strukture>
{
    <tip_1> <identifikator_1>;
    <tip_2> <identifikator_2>;
    …
    <tip_N> <identifikator_N>;
}
```

Definicija strukture se može naći van glavnog programa ili u deklarativnom delu programa i funkcija, a od pozicije definicije zavisi vidljivost strukture u programu. Ukoliko je definicija strukture u glavnom programu, struktura se ne može koristiti u funkcijama, a ukoliko je definicija strukture van glavnog programa, može se koristiti bilo gde.

Primer:

```
#include <stdio.h>
struct tacka
{
    int x;
    int y;
} ;
main ()
{
    int i, j;
    struct tacka  t1, t2;
    float x;
    struct tacka  T;
…
}
```

Da bi se prilikom deklaracije promenljivih izbeglo pisanje ključne reči struct moguće je koristiti naredbu za definisanje tipa typedef. Nakon definicije strukture, noviji kompajleri, za razliku od ANSII C kompajlera, podrazumevaju definiciju tipa tako da ključnu reč struct nije neophodno navoditi.

Primer:

```
main ()
{
    typedef struct tacka point;
    point P;
}
```

Inicijalizacija strukture i pristup članovima

Promenljiva strukturnog tipa se može inicijalizovati prilikom deklaracije na sledeći način:

Primer:

```
struct kompleksni
{
    float Re;
    float Im;
}
main ()
{
    kompleksni Z={1.0,2.0};
    ...
}
```

Za pristup elementima strukture koristi se binarni operator . . Sa leve strane ovog operatora navodi se promenljiva strukturnog tipa, a sa desne strane naziv elementa kome se pristupa:

```
<naziv_promenljive>.<naziv_elementa>
```

Primer:

```
main ()
{
    kompleksni z, x, y;
    x.Re=10.0;
    x.Im=20.0;
    z.Re=1.0;
    y.Im=15.1;
    z.Re=x.Re+y.Re;
    z.Im=x.Im+y.Im;
    printf("%f + %fi", z.Re, z.Im);
}
```

Strukture i funkcije

Strukture mogu biti parametri funkcije čija se vrednost prenosi bilo po vrednosti ili po referenci. U slučaju prenosa po vrednosti svi elementi strukture će se kopirati u odgovarajuće elemente fiktivnog argumenta strukturnog tipa.

Primer:

Napisati funkciju za sabiranje kompleksnih brojeva.

```
#include <stdio.h>
struct cplx
{
    int Re;
    int Im;
};
typedef struct cplx cplx;
cplx zbir(cplx x, cplx y)
{
    cplx z;
    z.Re=x.Re+y.Re;
    z.Im=x.Im+y.Im;
    return z;
}
main ()
{
    cplx a,b,c;
    scanf("%d + %di", &a.Re, &a.Im);
    scanf("%d + %di", &b.Re, &b.Im);
    c=zbir(a,b);
    printf("%d + %di", c.Re, c.Im);
}
```

Polja struktura

Promenljive strukturnog tipa mogu se grupisati u polja.

Primer:

```
#include <stdio.h>
struct tacka
{
    int x;
    int y;
    int z;
};
typedef struct tacka tacka;

main ()
{
    tacka A[100];
    A[0].x=2;        A[0].y=3;        A[0].z=14;
    A[1].x=10;       A[1].y=15;       A[1].z=54;
    ...
}
```

Ugnježdene strukture

U okviru strukture moguće je pored elemenata nekih od osnovnih tipova (int, float, char, …) definisati i druge strukturne elemente. Takde strukture se nazivaju ugnježdene strukture.

Primer:

```
#include <stdio.h>
struct tacka
{
    int x; int y;
};
typedef struct tacka tacka;

struct boja
{
    int R; int G; int B;
};
typedef struct boja boja;

struct pravougaonik
{
    tacka gore_levo, dole_desno; boja B;
};
typedef struct pravougaonik pravougaonik;

main ()
{
    pravougaonik P;
    scanf("%d", &P.B.R);
    scanf("%d", &P.B.G);
    scanf("%d", &P.B.B);
    ...
}
```

Pokazivači na strukture

Ukoliko se umesto promenljive koristi pokazivač, binarni operator . menja se binarnim operatorom -> .

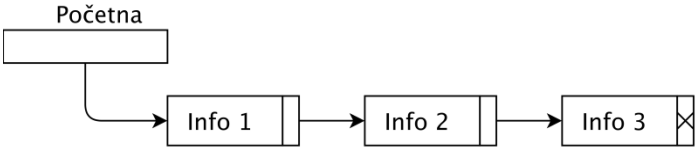
Primer:

```
main ()
{
    tacka T, *p;
    p=&T;
    p->x=10; // Ekvivalentno je izrazu: (*p).x=10
}
```

Samoreferencirajuće strukture

Strukture mogu sadržati elemente osnovnih ili strukturnih tipova, ali ne mogu sadržati rekurzivno elemente te istr strukture. Struktura može sadržati referencu na tu istu strukturu. Ovim je omogućeno kreiranje složenih struktura podataka tipa lančane liste, stabla i slično.

Grafički prikaz lančanih lista:




```
#include <stdio.h>
struct cvor
{
    int info;
    cvor *link;
};
typedef struct cvor cvor;

main ()
{
    cvor X[3];
    cvor *pocetak;
    pocetak=&X[0];
    X[0].link=&X[1];
    X[1].link=&X[2];
    X[2].link=NULL;
}
```

Unije

Unije u programskom jeziku C predstavljaju specijalni tip struktura kod kojih se, za razliku od struktura, umesto odvajanja memorijskog prostora za svaki element strukture kreira samo jedna univerzalna memorijska lokacija. Razlika kod definicije je jedino u ključnoj reči - union, a sve ostalo što važi za strukture važi i za unije.

Primer:

```
#include <stdio.h>
union primer
{
    int x; float y;
};
```

```
main ()
{
    primer p;
    p.x=8;
    p.y=13.0;
    printf("%d", p.x);
}
```

Na izlazu će biti prikazano: 1095761920

Stringovi

Stringovi su izvedeni tipovi podataka koji se koriste za pamćenje tekstualnih podataka. Programski jezici se generalno dosta razlikuju po ugrađenoj podršci koju imaju za stringove. S jedne strane postoje jezici kod kojih se stringovi deklarišu kao osnovni tip, a za operacije nad stringovima postoje definisani operateri. Sa druge strane, postoje jezici kod kojih podrška za stringove na osnovnom nivou ne postoji. C spada u drugu grupu jezika. Jedini dodatak koji programski jezik C nudi za rad sa stringovima u odnosu na nizove jeste inicijalizacija prilikom deklaracije u okviru koje se stringu može dodeliti literal. Operatori za operacije nad stringovima u programskom jeziku C ne postoje, uključujući i dodelu. Međutim, C kompajleri dolaze sa bibliotekom string.h koja sadrži veliki broj funkcija za rad sa stringovima.

Deklaracija i inicijalizacija

Primer:

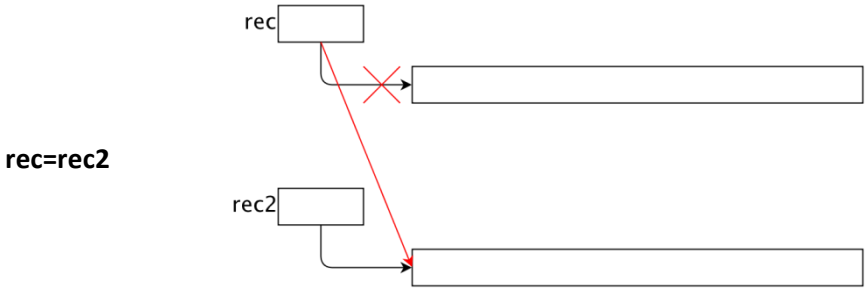
```
main ()
{
    char str[50];
    char rec[]={'o', 'v', 'o', ' ', 'j', 'e', ' ', 's', 't', 'r', 'i', 'n', 'g'};
    char rec2[]="Ovo je string";
    rec="Ovo je string";      //Nesipravno!
    rec[0]='O';
    rec[1]='v';
    rec[2]='o';
    …
}
```

Poređenje stringova

Primer:

```
main ()
{
    if(rec==rec2)
            printf("Jednaki su.");
    else
            printf("Nisu jednaki.");
}
```

Ovakvo poređenje je sintaksno ispravno i kompajler neće prikazati grešku. Poređenje u uslovu if petlje poredi da li pokazivač na početak niza rec pokazuje na istu adresu kao pokazivač rec2. Ovaj uslov je uvek netačan. Sadržaj stringova se na ovaj način ne može porediti.



Operator dodele programskog jezika C ne može kopirati vrednost stringa u string. Dodela je sintaksno ispravna, ali je posledica ta da se veza ka sadržaju prvog stringa raskida.

Unos i prikaz stringova

Za unos i prikaz stringova mogu se koristiti funkcije printf i scanf sa konverzionim karakerom %s.

Primer:

```
main ()
{
    char R[50];
    ptintf("Unesite string: ");
    scanf("%s", R);
    printf("Uneto je: \n%s", R);
}
```

Funkcija scanf prilikom unosa stringa za konverzioni karakter %s vrši učitavanje stringa do prvog blanko znaka, odnosno jedan konverzioni karakter %s vršu učitavanje jedne reči iz unete rečenice. Ovo se naziva formatirani unos.Zbog toga što se format ulaznih podataka tačno zadaje.

Primer:

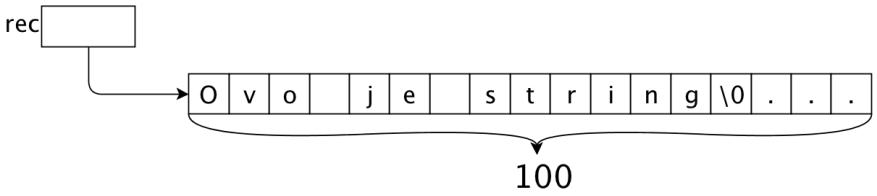
```
main ()
{
    char ime[25], prezime[25];
    int brInd;
    ptintf("Unesi broj indeksa, ime i prezime: ");
    scanf("%d%s%s", &brInd, ime, prezime);
    printf("Broj indeksa: %d\n", brInd);
    printf("Ime: %s\n", ime);
    printf("Prezime: %s", prezime);
}
```

U slučaju da je u programu potrebno uneti string za koji je broj reči unapred nepoznat funkcija scanf je praktično neupotrebljiva. U tom slučaju se koristi funkcija gets iz biblioteke string.h.

Primer:

```
#include <stdio.h>
#include <string.h>

main ()
{
    char Recenica[100];
    gets(Recenica);
    printf("Uneto je: %s", Recenica);
}
```

Bez obzira preko koje funkcije se stringovi unose, funkcije za unos stringova na kraju stringa dodaju simbol '\0', tj. NULL. Ovaj specijalni karakter se dodaje nakon poslednjeg unešenog karaktera. Memorija nakon unosa stringa izgleda ovako:



Razlozi za dodavanje NULL karaktera na kraj stringa su ograničenje dužine kod pretrage i ograničavanje broja karaktera koje funkcija printf prikazuje.

Osnovne operacije za rad sa stringovima

Dodela vrednosti

Dodela vrednost je složena operacija kod koje se vrednost svakog elementa niza redom kopira na odgovarajuće mesto u odredišnom stringu. Programski jezik C u skupu operatera nema podršku za dodelu, a što se tiče algoritama i drugih programskih jezika koriste se operatori <- i =.

Konkatenacija

Konkatenacija je operacija nadovezivanja stringa na string. Programski jezik C nema operatorsku podršku za konkatenaciju. Za konkatenaciju se, kod jezika koji imaju operatorsku podršku, koriste operatori ||, & i +.

Poređenje

Poređenje stringova je složena operacija koja poredi redom karakter po karakter sve dok su karakteri jednaki, a ukoliko naiđe na različite karaktere poredi njihhove pozicije u abecedi što daje ukupan leksički odnos dva stringa. U programskom jeziku C ne postoji operatorska podrška za poređenja stringova, a u jezicima koji imaju operatorsku podršku za poređenje obično se koristi =.

Pretraga

Pretraga je složena operacija koja određuje poziciju na kojoj se sadžaj jednog stringa javlja u drugom stringu.

Dokumentovanje funkcija

Uz svaki C kompajler dolazi veliki broj funkcija grupisanih u header file-ove. Uz MS Visual Studio dolazi dokumentacija pod nazivom MSDN (Microsoft Developer Network) koja sadrž opise svih funkcija. MSDN se može otvoriti iz Help-a, ili pritiskom na taster F1. Ukoliko je kursor pozicioniran na naziv funkcije i pritisne se taster F1 otvara se opsi funkcije. Opis svake funkcije sadrži sledeće:

1. Deklaracija funkcije (tip, ime, lista parametara)
2. Dejstvo funkcije
3. Opis povratne vrednosti
4. Opis parametara
5. Ime biblioteke u kojoj se nalazi
6. Primer upotrebe

Biblioteka string.h

Uključivanjem biblioteke string.h u program uvodi se mnoštvo funkcija za rad sa stringovima. Među funkcijama za rad sa stringovima su funkcije za određivanje dužine stringa, kopiranje, poređenje, konkatenaciju i pretragu.

Funkcija strlen

1. int strlen(char *string)
2. Funkcija vraća celobrojni podatak koji predstavlja broj karaktera od početka stringa do NULL karaktera.
3. Vraća ceo broj.
4. Parametar funkcije je adresa stringa čija se dužina određuje.
5. Nalazi se u biblioteci string.h.

Primer:

```
#include <stdio.h>
#include <string.h>

main ()
{
    char R[100];
    int L;
    gets(R);
    L=strlen(R);
    printf("Duzina je: %d", L);
}
```

Funkcija strcpy

1. char* strcpy(char *S1, char *S2)
2. Funkcija strcpy vrši kopiranje sadržaja stringa S2 u string S1.
3. Funkcija vraća pokazivač na string S1.
4. S1 je string u koji se kopira sadržaj, a S2 je string iz koga se kopira sadržaj.
5. Nalazi se u biblioteci string.h.

Primer:

```
#include <stdio.h>
#include <string.h>

main ()
{
    char S1[]="Hello World!";
    char S2[50];
    strcpy(S2, S1)
    printf("%s", S2);          //Biće prikazano "Hello World!"
}
```

Funkcija strncpy

1. char* strcpy(char *S1, char *S2, int n)
2. Funkcija strncpy vrši kopiranje prvih n karaktera sadržaja stringa S2 u string S1. Prilikom ovog kopiranja ne dodaje se NULL karakter na kraju.
3. Povratna vrednost je pokazivač na početak stringa S1.
4. S1 je string u koji se kopira sadržaj, S2 je string iz koga se kopira sadržaj, a n je broj karaktera koji se kopira.
5. Nalazi se u biblioteci string.h.

Primer:

```
#include <stdio.h>
#include <string.h>

main ()
{
    char S1[]="Hello World!";
    char S2[]="World";
    strncpy(S1, S2, 4)
    printf("%s", S2);//Biće prikazano "Worlo World!"
}
```

Funkcija strcmp

1. int strcmp(char *S1, char *S2)
2. Funkcija strcmp vrši poređenje karakter po karakter i vraća odgovarajuću vrednost koja odgovara leksičkom uređenju stringova
3. Povratna vrednost je ceo broj. Funkcija vraća nulu ako su stringovi jednaki, negativnu vrednost ako je string S1 leksički ispred stringa S2, i pozitivnu vrednost ako je string S2 leksički ispred stringa S1.
4. Stringovi S1 i S2 su stringovi koji se porede.
5. Nalazi se u biblioteci string.h.

Primer:

```
#include <stdio.h>
#include <string.h>

main ()
{
    char odg[2];
    printf("Unesite da ili ne: ");
    gets(odg);
    if(!strcmp(odg, "da"))
            printf("Uneli ste da");
    else
            printf("Uneli ste ne");
}
```

Funkcija strncmp

1. int strcmp(char *S1, char *S2, int n)
2. Funkcija strcmp vrši poređenje karakter po karakter prvih n karaktera.
3. Povratna vrednost je ceo broj. Funkcija vraća nulu ako su prvih n karaktera stringova jednaki, negativnu vrednost ako je prvih n karaktera stringa S1 leksički ispred prvih n karatera stringa S2, i pozitivnu vrednost ako je prvih n karaktera stringa S2 leksički ispred prvih n karaktera stringa S1.
4. Stringovi S1 i S2 su stringovi koji se porede, n je broj karaktera koji se porede.
5. Nalazi se u biblioteci string.h.

Funkcija strcat

1. char* strcat(char *S1, char *S2)
2. Funkcija strcat vrši konkatenaciju (nadovezivanje) stringova. String S2 se nadovezuje na string S1. Ukoliko u S1 nema dovoljno mesta za oba stringa, funkcija automatski vrši realokaciju prostora, tj. stringu S1 dodeljuje novi, veći prostor.
3. Povratna vrednost funkcije je pokazivač na početak stringa S1.
4. String S1 je string na koga se nadovezuje string S2.
5. Nalazi se u biblioteci string.h.

Primer:

```
#include <stdio.h>
#include <string.h>

main ()
{
    char S1[]="Hello ";
    char S2[]="World!";
    strcat(S1, S2);
    printf("%s", S1);          //Biće prikazano "Hello World!"
}
```

Funkcija strncat

1. char* strcat(char *S1, char *S2, int n)
2. Funkcija strncat vrši nadovezivanje prvih n karaktera stringa S2 na string S1. Ukoliko u S1 nema dovoljno mesta za oba stringa, funkcija automatski vrši realokaciju prostora, tj. stringu S1 dodeljuje novi, veći prostor.
3. Povratna vrednost funkcije je pokazivač na početak stringa S1.
4. String S1 je string na koga se nadovezuje string S2, n je broj karaktera koji se nadovezuje na S1.
5. Nalazi se u biblioteci string.h.

Funkcija strchr

1. char* strchr(char *S, char C)
2. Funkcija strchr određuje poziciju prvog pojavljivanja karaktera C u stringu S.
3. Povratna vrednost funkcije je adresa na prvo pojavljivanje karaktera C.
4. S je string u kome se traži karakter, C je karakter koji se traži.
5. Nalazi se u biblioteci string.h.

Primer:

```
#include <stdio.h>
#include <string.h>

main ()
{
    char S[]="Hello World!";
    char *rez;
    rez=strchr(S, 'o');
    printf("%s", rez);          //Biće prikazano "o World!"
}
```

Funkcija strstr

1. char* strstr(char *S1, char *S2)
2. Funkcija strstr vraća adresu prvog pojavljivanja stringa S2 u stringu S1.
3. Povratna vrednost funkcije je adresa od koje se string S2 javlja u stringu S1.
4. String S2 je string koji se traži, a string S1 je string u kome se traži. Ukoliko string S2 ne postoji u stringu S1 funkcija vraća vrednost NULL.
5. Nalazi se u biblioteci string.h.

Primer:

```
#include <stdio.h>
#include <string.h>

main ()
{
    char S1[]="Hello World!";
    char S2[]="lo";
    char *rez=strstr(S1, S2);
    printf("%s", rez);          //Biće prikazano "lo World!"
}
```

Funkcija strpbrk

1. char* strpbrk(char *S1, char *S2)
2. Funkcija strpbrk vraća adresu prvog pojavljivanja bilo kog karaktera iz stringa S2 u stringu S1.
3. Povratna vrednost funkcije je adresa prvog pojavljivanja bilo kog karaktera iz stringa S2 u stringu S1. Ukoliko u stringu S1 ne postoji karakter koji se takođe nalazi i u stringu S2, funkcija vraća NULL.
4. String S1 je string u kome se traži, string S2 je skup karaktera za pretragu.
5. Nalazi se u biblioteci string.h.

Primer:

```
main ()
{
    char S1[]="Hello World!"; char S2[]="aeiou";
    char *rez=strpbrk(S1, S2);
  printf("%s", rez);  //Biće prikazano "ello World!"
}
```

Funkcija strspn

1. size_t strspn(char *S1, char *S2)
2. Funkcija strspn vraća dužinu levog dela stringa S1 koji sadrži isključivo karaktere koji se nalaze u S2.
3. Povratna vrednost ove funkcije je tipa size_t. size_t je tip podataka koji se koristi za pamćenje celih brojeva. Opseg vrednosti su pozitivni celi brojevi toliki da je moguće predstaviti veličinu bilo kog objekta u memoriji. Povratna vrednost je pozitivan ceo broj i to 0 ukoliko ni jedan karakter iz stringa S2 nije sadržan u stringu S1, i broj različit od 0 ukoliko levi deo stringa S1 bez prekida sadrži karaktere iz stringa S2.
4. String S1 je string u kome se traži, string S2 je skup karaktera za pretragu.
5. Nalazi se u biblioteci string.h.

Primer:

```
#include <stdio.h>
#include <string.h>

main ()
{
    char S1[]="Hello World!";
    char S2[]="leoH";
    size_t *rez=strspn(S1, S2);
    printf("%d", rez);          //Biće prikazano 6
}
```

Funkcija strtok

1. char* strtok(char *S1, char *S2)
2. Funkcija strtok razvija string S1 na reči i može da vrati reč po reč.
3. Povratna vrednost je pokazivač na izdvojenu reč. Ukoliko je string S1 različit od NULL karaktera, funkcija vraća pokazivač na prvu reč u stringu S1. Ukoliko je string S1 jednak NULL karakteru, funkcija vraća pokazivač na narednu reč u stringu S1 u odnosu na reč sa kojim je funkcija pozvana prethodni put.
4. String S1 je string u kome se traže reči, a string S2 je skup karaktera kojim su reči razdvojene.
5. Nalazi se u biblioteci string.h.

Primer:

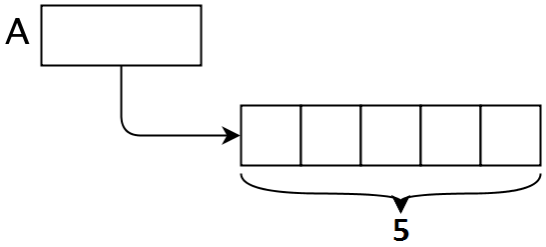
```
#include <stdio.h>
#include <string.h>

main ()
{
    char S1[]="Ovo je neki string.";
    char S2[]=" ,._";
    char *rez=strtok(S1, S2);
    while(rez!=NULL)
    {
            printf("%s\n", rez);
            rez=strtok(NULL, S2);
    }
}
```

Nizovi pokazivača

Inicijalizacija prilikom deklaracije niza zadaje vrednosti elementima u trenutnu formiranja niza. Sama deklaracija niza se vrši na sledeći način:

```
int A[5];
```

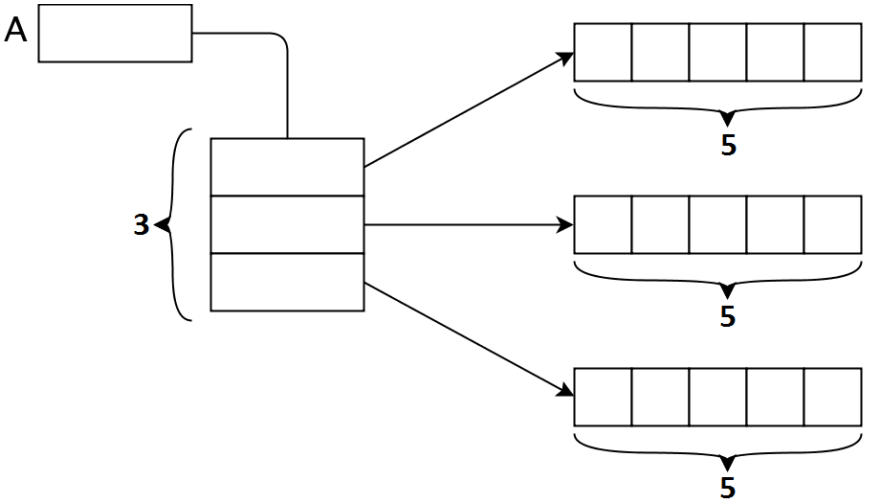
Tom prilikom se u memoriji odvaja niz od 10 sukcesivnih lokacija za cele brojeve i jedan pokazivač koji pokazuje na početak niza. Memorijska reprezentacija izgleda ovako:



Deklaracija dvodimenzionalnih polja se vrši na sledeći način:

```
int A[3][5]
```

Samo ime matrice je pokazivač na niz pokazivača koji pokazuju na nizove (vrste). Po tipu A je int **A. Memorijska reprezentacija izgleda ovako:



Navođenjem samo prvog indeksa vrši se referenciranje na odgovarajuću vrstu zato što se navođenjem prvog indeksa dobija podatak iz niza pokazivača, što je ustvari adresa prvog elementa niza (vrste). Niz stringova moguće je zapamtiti kao matricu karaktera.

Primer:

Napisati strukturni program koji od 10 unetih prezimena određuje i prikazuje najduže.

```
#include <stdio.h>
#include <string.h>
main ()      {
    int i, maxD; char maxP[25], P[10][25];
    for(i=0; i<10; i++)
            gets(P[i]);
    maxD=strlen(P[0]);
    strcpy(maxP, P[0]);
    for(i=1; i<10; i++)
            if(maxD<strlen(P[i]))      {
                    maxD=strlen(P[i]);
                    strcpy(maxP, P[i]);
            }
    printf("\nNajduze prezime je: %s", maxP);
}
```

Ulaz/izlaz i fajlovi

Programski jezici nude podršku za rad sa fajlovima (datotekama) u okviru programa. Fajlovi se koriste za trajno skladištenje podataka na hard disku računara. Podrška za rad sa fajlovima može biti:

1. Podrška na nivou tokova - stream
2. Podrška na sistemskom nivou

Podrška na nivou tokova vrši apstrakciju pristupa fajlovima tako da je sam pristup i mehanizam upisa i čitanja skriven od korisnika. Kod ovog načina korisnik kreira tok podataka koji se može vezati za fajl na disku računara nakon čega može da piše i čita iz kreiranog toka. Postoje dva tipa tokova, standardni i korisnički. Standardne nije potrebno definisati i u njih spadaju:

- stdin - standardni tok za ulaz podataka koji je, ukoliko nije drugačije rečeno, vezan za tastaturu
- stdout - standardni tok za izlaz podataka (prikaz na monitoru)
- stderr - izlazni tok koji se koristi za prikaz grešaka. Izdvaja se od stdout-a da u slučaju izlaza na štampač greške ne bi bile štampane već prikazane na monitoru.
- stdprn - standardni tok koji je vezan za štampač
- stdaux - standardni tok vezan za serijski port računara

Funkcije printf i scanf imaju proširenje za rad sa tokovima - fprintf i fscanf. Obe funkcije se nalaze u biblioteci stdio.h.

Primer:

```
#include <stdio.h>
main ()
{
    int x;
    fscanf(stdin, "%d", &x);  //Ekvivalentno naredbi "scanf("%d", &x);
  fprintf(stdout, "%d", x);
}
```

Standardni ulaz stdin je buffer. To znači da će se podaci koje korisnik unese preko tastature pamtiti u neki buffer (pomoćna memorija) i da će se prilikom pritiska tastera Enter preneti u program. Ostali tokovi nisu buffer.

Podrška na sistemskom nivou za razliku od tokova omogućava korisniku da vodi računa o mehanizmima za pristup fajlovima. Ova podrška omogućena je pozivom sistemskih funkcija (operativnim sistemom) koje se nazivaju API (Aplication Interface). API se u korisnički program uključuje posebnim header fajlovima i pored rada sa fajlovima nudi kompletnu podršku koju pruža operativni sistem (mreža i slično).

Preusmeravanje tokova podataka

Standardni tokovi stdin i stdout se mogu preusmeriti na fajl. Ovo preusmeravanje vrši operativni sistem na zahtev korisnika iz konzole Windows-a. Preusmeravanje je moguće izvršiti binarnim operaterima < ili >. Ovi operateri se navode prilikom poziva preusmeravanja i sa leve strane imaju izvršni fajl, a sa desne fajl na koji se vrši preusmeravanje. Operator < vrši preusmeravanje toka iz fajla na standardni ulaz, a operator > vrši preusmeravanje izlaznog toka na fajl.

Primer:

```
C:\aip.exe>izlaz.txt
aip.exe<ulaz.txt


aip.exe<ulaz.txt>izlaz.txt
```

Korisnički tokovi podataka

U programskom jeziku C moguća je definicija korisničkih tokova podataka. Svaki tok podataka ima svoj tip. Tip podataka FILE, koji je definisan u biblioteci stdio.h, predstavlja strukturu koja pamti veći broj podataka o toku, tj. fajlu koji opisuje. Ovaj tip podataka pamti poziciju fajla na disku, veličinu fajla,

poziciju do koje se stalo sa čitanjem fajla… S obzirom da veliki broj funkcija za rad sa tokovima vrši neku modifikaciju (npr. povećavanje brojača pozicije do koje se stiglo sa čitanjem), ova struktura se uglavnom koristi preko pokazivača na strukturu.

Funkcije za rad sa fajlovima

Osnovne funkcije za rad sa fajlovima su:

1. fopen - funkcija za otvaranje fajla
2. fscanf - funkcija za čitanje iz fajla
3. fprintf - funkcija za upis u fajl
4. fclose - funkcija za zatvaranje fajla

Otvaranjem fajla se kreira tok podataka između fajla i korisničkog programa i tom toku se dodeljuje jedna struktura tipa FILE. U programu može biti kreiran proizvoljan broj tokova. Za pristup svakom pojedinačnom toku koristi se njegova struktura tipa FILE. Nakon kreiranja toka moguć je upis i čitanje iz fajla, a na kraju rada je potrebno zatvoriti fajl. Prilikom čitanja i upisa hard disk računara podatke privremeno čuva u buffer-u diska čiji se sadržaj periodično prazni upisom na disk.Ovo se radi da bi se postigla veća brzina pristupa disku. Ukoliko se korisnički program završi, a fajl se ne zatvori, postoji mogućnost trajnog gubitka podataka u slučaju nestanka struje. Zatvaranjem fajla se eksplicitno buffer prazni i strukture oslobađaju. Zatvaranje fajla nije obavezno, ali je poželjno.

Funkcija za otvaranje fajla (kreiranje toka)

1. FILE* fopen(char *ime, char *mod)
2. Funkcija fopen vrši sve pripremne radnje i kreira tok podataka.
3. Nakon kreiranja toka funkcija kreira jednu strukturu tipa FILE, ažurira i vraća u vidu pokazivača na tu strukturu. Ukoliko fajl nije moguće otvoriti, funkcija vraća NULL. Fajl nije moguće otvoriti u dva slučaja, u slučaju čitanja kada fajl ne postoji na disku i u slučaju upisa kada je fajl već otvoren ili kada nema dovoljno mesta na disku.
4. Prvi parametar funkcije je string koji sadrži samo ime fajla ili putanju (apsolutnu ili relativnu). Drugi parametar je string koji ukazuje na to kako treba otvoriti fajl.
   - "r" - fajl se otvara za čitanje - ukoliko ne postoji funkcija vraća NULL
   - "w" - fajl se otvara za upis - ukoliko ne postoji, kreira se, ukoliko postoji sadržaj se briše
   - "w" - fajl se otvara za upis na kraj fajla, postojeći sadržaj ostaje nepromenjen
   - "r+" - fajl se otvara za čitanje sa upisom
   - "w+" - fajl se otvara za upis sa čitanjem
   - "a+" - fajl se otvara za upis sa čitanjem, postojeći sadržaj ostaje nepromenjen
5. Nalazi se u biblioteci stdio.h.

Funkcija za zatvaranje fajla

1. int fclose(FILE *f)
2. Funkcija fclose vrši zatvaranje fajla f.
3. Funkcija vraća nulu ukoliko je zatvaranje uspešno.
4. Parametar funkcije je pokazivač na strukturu tipa FILE.
5. Nalazi se u biblioteci stdio.h.

Primer:

```
#include <stdio.h>

main ()
{
    int x=10; FILE *p;
    p=fopen("ime.txt", "w");
    fprintf(p, "%d\n", x);
    fclose(p);
}
```

Tipovi fajlova

Svi fajlovi mogu se podeliti u dve grupe:

1. tekstualni fajlovi
2. binarni fajlovi

Tekstualni fajlovi

Tekstualni fajlovi su na hard disku zapamćeni tako da se u susednim lokacijama nalaze ASCII, UTF8 ili slični kodovi za predstavljanje karaktera. Ove fajlove moguće je otvoriti i čitljivi su.

Funkcija fprintf

1. int fprintf(FILE *tok, char *format[,<argumenti>])
2. Funkcija fprintf se koristi za upis podataka u tekstualni fajl.
3. Funkcija vraća nulu ukoliko je upisivanje uspešno.
4. Ova funkcija upisuje podatke prenete preko argumenata po zadatom formatu (isto kao printf) u prethodno otvoreni fajl na koji pokazuje pokazivač toka.
5. Nalazi se u biblioteci stdio.h.

Primer:

```
#include <stdio.h>

main ()
{
    int x=3; float y=+3.2-E4; FILE *f;
    f=fopen("primer.txt", "w");
    fprintf(f, "x=%d, y=%f\n", x, y);
    fclose(f);
}
```

Funkcija fscanf

1. int fscanf(FILE *tok, char *format,<argumenti>)
2. Funkcija fscanf se koristi za čitanje podataka iz tekstualnog fajla.
3. Funkcija vraća broj učitanih podataka.
4. Ova funkcija vrši učitavanje podatka iz prethodno otvorenog toka po zadatom formatu (isto kao scanf) na adrese navedenih argumenata. Ona modifikuje strukturu tok tako da pokazivač na bajt koji je potrebno pročitati pomera iza poslednjeg pročitanog bajta. Sledeći poziv funkcije čita sledeći podatak.
5. Nalazi se u biblioteci stdio.h.

Primer:

```
#include <stdio.h>

main ()
{
    int ocena, bri; char ime[25], prezime[25];
    FILE *std=fopen("studenti.txt", "r");
    fscanf(std, "%d", ocena);
    fscanf(std, "%s%s", ime, prezime);
    fscanf(std, "%d", bri);
    fclose(std);
}
```

Funkcija feof

1. int feof(FILE *tok)
2. Prilikom čitanja iz fajla može se doći do kraja fajla. Funkcija feof se koristi za ispitivanje statusa čitanja fajla.
3. Funkcija vraća nulu ukoliko se nije došlo do kraja fajla, a vrednost različitu od nule ukoliko se došlo do kraja fajla.
4. Parametar funkcije je tok čiji se status proverava.
5. Nalazi se u biblioteci stdio.h.

Primer:

```
#include <stdio.h>

main ()
{
    int i=0, A[100]; FILE *f=fopen("ocene.txt", "r");
    while(!feof)
    {
          fscanf(f, "%d", A+i);
              i++;
    }
}
```

Funkcije sprintf i sscanf

Ove dve funkcije su po formatu identične kao funkcije fprintf i fscanf, a jedina razlika je to što se kod ovih funkcija upis ne vrši iz toka već iz stringa. Deklaracije funkcija su sledeće:

1. int sprintf(char* odredište, char* format[,<argumenti>])
2. int sscanf(char* odredište, char* format[,<argumenti>])

Funkcijom sscanf moguće je izdvojiti reči iz rečenice po zadatom formatu.

Primer:

```
#include <stdio.h>

main ()
{
    char imeprezime[]="Petar Peric", char ime[25], prezime[25];
    sscanf(imeprezime, "%s%s", ime, prezime);
}
```

Dodatne funkcije za upis i čitanje

1. int fgetc(FILE *tok) - funkcija za čitanje karaktera iz fajla
2. int getc(c) - funkcija za čitanje kraktera iz standardnog toka
3. inf fputc(FILE *tok) - funkcija za upis karaktera u fajl
4. int putc(c) - funkcija za upis kraktera u standardni tok
5. int fgets(FILE *tok) - funkcija za čitanje stringa iz fajla
6. int fputs(FILE *tok) - funkcija za upis stringa u fajl

Binarni fajlovi

Velika većina fajlova pak pripada grupi binarnih fajlova. Kod binarnih fajlova se podaci pamte onakvi kakvi jesu, bez kodiranja. Otvaranje binarnog fajla se vrši na isti način kao i otvaranje tekstualnih fajlova. Jedina razlika je u tome što se modu funkcije fopen dodaje slovo b.

Primer:

```
main ()    {
  FILE *f=fopen("ime.gif", "rb");
}
```

Čitanje iz binarnih fajlova

Čitanje iz binarnog fajla se obavlja sledećom funkcijom:

```
size_t fread(void* buffer, size_t velicina_podatka, size_t broj_podataka, FILE *tok)
```

Funkcija fread čita [broj_podataka] podataka veličine [veličina_podatka] iz fajla [tok] sa memorijskom lokacijom [buffer].

Upis u binarne fajlove

Upis u binarni fajl se obavlja sledećom funkcijom:

```
size_t fwrite(void* buffer, size_t veličina_podatka, size_t broj_podataka, FILE *tok)
```

Funkcija fwrite upisuje [broj_podataka] podataka veličine [veličina_podatka] u fajl [tok] sa memorijskom lokacijom [buffer].

Funkcija za pozicioniranje u binarnom fajlu

Nakon čitanja podataka funkcija fread pozicioniraće fajl na [[veličina_podatka]*[broj_podataka]+1] bajt. Kod binarnih fajlova, pored sekvencionih čitanja podataka, pozicioniranje na podatak koji se treba pročitati može se izvršiti u bilo kom trenutku sledećom funkcijom:

```
int fseek(FILE *tok, long offset, int pozicija)
```

Parametar pozicija može iamti tri vrednosti koje su definisane konstantama:

SEEK_CUR - trenutna vrednost

SEEK_SET - početak fajla

SEEK_END - kraj fajla

Funkcija fseek pokazivač na sledeći bajt koji treba pročitati postavlja na [[pozicija]+[offset]] bajt gde je [pozicija] oznaka dela fajla definisanog konstantama.

Primer:

```
#include <stdio.h>

main()
{
    FILE *f=fopen("podaci.bin", "rb");
    fseek(f, 0, SEEK_END);          //"Premotavanje na kraj fajla
    fseek(f, -5, SEEK_CUR);         //"Premotavanje" 5 bajta unazad
    fseek(f, 0, SEEK_SET);          //"Premotavanje" na početak fajla
}
```

“Premotavanje” na početak fajla moguće je izvršiti i sledećom funkcijom:

```
int rewind(FILE *tok)
```

Bez obzira da li se radi o binarnom ili tekstualnom fajlu, na početak fajla može se pozicionirati zatvaranjem i ponovnim otvaranjem fajla:

```
int freopen( char *ime, char *mod, FILE *tok)
```

Primer:

```
#include <stdio.h>
main()      {
    FILE *f=fopen("ulaz.bin", "rb"); int A[]={1,2,3,4,5,6,7,8,9,0}, B[15];
    fwrite(A, sizeof(int), 10, f);
    fclose(f);
    FILE *g=fopen("ulaz.bin", "rb");
    fread(B, sizeof(int), 10, g);
    fseek(g, 2*sizeof(int), SEEK_SET);
    fread(B+10, sizeof(int), 5, g);
    fclose(g);
}
```

Lokalne i globalne promenljive

Pored tipa i skupa vrednosti promenljive u programskim jezicima definisane su i memorijskom klasom i vremenskim intervalom u kom je promenljiva prisutna u memoriji. Po vremenu za koje je promenljiva prisutna u memoriji razlikuje globalne i lokalne promenljive

Lokalna promenljiva je lokalna za određeni blok programa i postoji samo za vreme izvršavanja tog bloka. Kod ANSII-C-a promenljive mogu biti lokalne na nivou funkcije.

Primer:

```
#include <stdio.h>
int f(int a)
{
    int pom;
    pom=a+1;
    return pom;
}

main()
{
    int x,y;
    scanf("%d", &x);
    y=f(x);
}
```

Promenljiva pom je lokalna za funkciju f. To znači da će se memorija za ovu promeljivu rezervisati tek nakon poziva funkcije f u glavnom programu. Nakon završetka funkcije memorija će se oslobiditi. Promenljiva pom je vidljiva samo iz funkcije i ne može se pozvati u glavnom programu. Promenljive x i y su lokalne promeljive za glavni program i nisu vidljive iz funkcija.

Primer:

```
#include <stdio.h>

int f1(int a)
{
    int x;
    x=a+10;
    return x;
}

main()
{
    int x=10, y;
    y=f1(x);
}
```

Bez obzira što i funkcija ima lokalnu promenljivu x, kao i glavni program, ovo su dve različite promeljive na dve različite lokacije koje postoje u različito vreme. Kod C99/C++ kompajlera promenljiva može biti lokalna na nivou bilo kog bloka u kodu.

Primer:

```
main()      {
    int x=10, y;
    for(int i=0; i<11; i++)
            int pom=i+x;
    i=5; pom=10;    //Nije moguće pošto i i pom postoje samo dok se izvršava petlja.
}
```

Promenljive x i y su lokalne za glavni program i vidljive su u svom podblokovima koje glavni program sadrži. Promenljive i i pom su lokalne za for petlju, postoje samo za vreme izvršenja petlje i nisu vidljive nakon izvršenja petlje. Da petlja dalje sadrži ugnježdene strukture, promenljive i i pom bile bi vidljive u njima.

Globalne promenljive se deklarišu van funkcija i glavnog programa i vidljive su iz bilo kog dela programa.

Primer:

```
#include <stdio.h>
int a,b,c;

void zbir()
{
    int pom;
    pom=a+b;
    c=pom;
}

main()
{
    scanf("%d%d", &a, &b);
    zbir();
    printf("%d", c);
}
```

Memorijske klase identifikatora

Promenljive u programskom jeziku C mogu pripadati jednoj od 4 memorijske klase:

1. automatska
2. eksterna
3. statička
4. registarska

Automatska klasa promenljivih

Ukoliko drugačije nije rečeno, u programskom jeziku C promenljiva pripada automatskoj memorijskoj klasi promenljivih. Kod ove klase kompajler sam vodi računa o lokaciji promeljive u memoriji. Ključna reč za eksplicitnu deklaraciju promenljive koja pripada ovoj klasi je auto i dodaje se ispred tipa u deklaraciji promenljive. Za sve promenljive kod kojih ne stoji ništa ispred tipa ovo je podrazumevana vrednost.

Primer:

```
main()
{
    auto int x;
}
```

Eksterna klasa promenljivih

Koncept eksternih promenljivih postoji u mnogim programskim jezicima. Kod eksternih promenljivih promenljiva se deklariše u okviru jednog fajla, a koristi se u okviru drugog. Ova dva fajla potrebno je kompajlirati istovremeno.

Primer:

fajl1.c

```
int a;
main()
{
    ...
}
```

fajl2.c

```
extern int a;
int f_ja(...)
{
    ...
}
```

Statička klasa promenljivih

Statičke promenljive su lokalne promenljive za neki blok programa. Karakteristika statičkih promenljivih je da se memorijska lokacija ne oslobađa nakon završetka bloka. Vrednost takođe ostaje nepromenjena.

Primer:

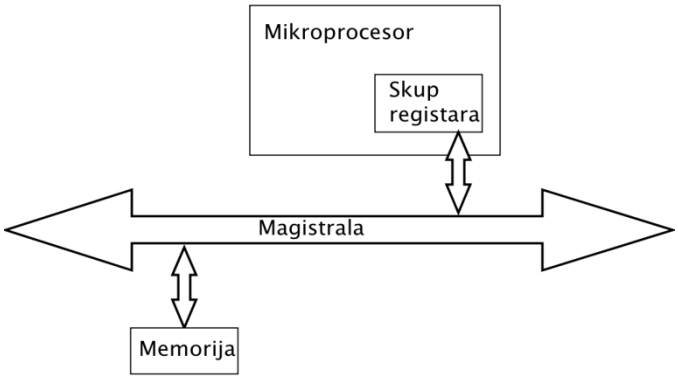
```
#include <stdio.h>
void prikazi()
{
    int a=0; static int b=0;
    printf("%d %d\n", a, b);
    b++; a++;
}
main()
{
    for(int i=0; i<10; i++)
            prikazi();
}
```

Nakon izvršenja ovog koda na ekranu će biti prikazano:

```
0 0
0 1
0 2
0 3
...
0 9
```

Registarska klasa promenljivih

U cilju bržeg izvršavanja programa kompajleru se može izadati naredba prilikom deklaracije da promenljivu pamti u registrima procesora umesto u memoriji računara. Registri procesora su memorijske jedinice malog kapaciteta (2b, 4b, 8b) koje se implementiraju u okviru samog procesora i koriste se za čuvanje operanada i rezultata. Kod nekih procesora (zavisno od arhitekture) da bi se operacija izvršila potrebno je operande prebaciti u registre procesora, izvršiti operaciju, a zatim operande vratiti u memoriju. Ključna reč je register. Ukoliko nema dovoljno slobodnih registara, kompajler će promeljivu smestiti u memoriju.



Dinamička alokacija memorije

Deklaracijom promenljive vrši se rezervacija prostora za smeštanje vrendnosti promenljive. Tip promenljive kompajleru daje opis načina smeštanja promenljive i potrebnu veličinu prostora, a ime promenljive daje oznaku kako će se toj promenljivoj pristupati. Deklaracije promenljivih se nalaze u deklarativnom delu programa i njihov broj nije moguće menjati u toku izvršenja programa. Ovakve promenljive se nazivaju statičke promenljive. Za veliku klasu problema potrebno je posedovati mehanizam koji će omogućiti rezervaciju novog memorijskog prostora proizvoljne veličine na inicijativu programera, bilo gde u programu. Ovakva alokacija u memoriji se naziva dimanička alokacija memorije. Primer programa gde je neophodna dinamička alokacija bio bi telefonski imenik. Ključna razlika između statičke i dinamičke alokacije je ta da se kod statičke alokacije alokacija vrši u trenutku kompajliranja programa, a kod dinamičke se alokacije vrši u bilo kom trenutku izvršenja programa. Dinamička alokacija memorije se vrši pomoći sledećih funkcija:

1. malloc i calloc
2. realloc
3. free

Sve tri funkcije su sadržane u bibliotekama stdlib.h i malloc.h. Pomoću ovih funkcija program traži od operativnog sistema rezervaciju određene veličine memorije.

Funkcija malloc

1. void* malloc(size_t velicina)
2. Funkcija malloc vrši dinamičku alokaciju memorije i vraća pokazivač na tu memoriju.
3. Parametar funkcije je veličina zahtevanog memorijskog prostora.
4. Povratna vrednost funkcije je pokazivač na prvu adresu u okviru dodeljenog memorijskog prostora. Tip koju funkcija vraća je void pointer (pokazivač na bilo šta) koji je potrebno cast-ovati u željeni tip.
5. Nalazi se u bibliotekama stdlib.h i malloc.h.

Primer:

```
#include <malloc.h>
main()
{
    int *p, *q;
    p=(int*)malloc(4);
    q=(int*)malloc(sizeof(int));
    *p=10;
    *q=*p+50;
}
```

Funkcija calloc

1. void* malloc(size_t num, size_t velicina)
2. Funkcija calloc vrši dinamičku alokaciju memorije i vraća pokazivač na tu memoriju.
3. Prvi parametar predstavlja broj memorijskih lokacija koje će biti rezervisane, dok drugi parametar predstavlja veličinu svake individualne memorijske lokacije. Ukupna memorija koja će biti rezervisana korišćenjem ove funkcije je [num]*[velicina].
4. Funkcija vraća adresu prve lokacije u dodeljenoj memoriji. Ukoliko nema dovoljno mesta u memoriji, funkcija vraća NULL.
5. Nalazi se u bibliotekama stdlib.h i malloc.h.

Primer:

```
#include <malloc.h>

main()
{
    int *A;
    A=(int*)calloc(10, sizeof(int));    //Niz
    A[0]=1;
    A[1]=2;
    A[2]=3;
    ...
}
```

Kod statičke alokacije niza broj elementa niza mora biti konstantan. Kod dinamičke alokacije pomoću funkcije calloc moguće je rezervisati mesto za onoliko podataka koliko je u tom trenutku potrebno.

Funkcija realloc

1. void* realloc(void *memorijski_blok, size_t nova_velicina)
2. Funkcija realloc vrši promenu veličine dodeljenog memorijskog prostora.
3. Povratna vrednost funkcije je početna adresa novog memorijskog bloka.
4. Pri parametar je adresa memorijskog bloka čija se veličina menja, a drugi parametar je veličina novog bloka u bajtovima.
5. Nalazi se u bibliotekama stdlib.h i malloc.h.

Primer:

```
#include <malloc.h>

main()
{
    int *A;
    A=(int*)calloc(10, sizeof(int));
    A[0]=1;
    A[1]=2;
    A[2]=3;
    ...
    A=(int*)realloc(A,20*sizeof(int));
    ...
}
```

Funkcija free

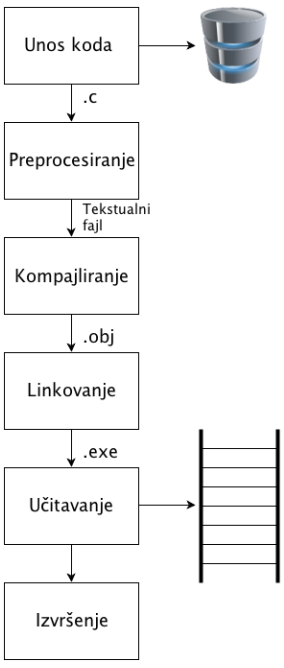
1. void free(void *memorijski_blok)
2. Funkcija free vrši oslobađanje dodeljenog memorijskog prostora.
3. Funkcija nema povratnu vrednost.
4. Parametar funkcije je adresa memorijskog prostora.
5. Nalazi se u bibliotekama stdlib.h i malloc.h.

Primer:

```
#include <malloc.h>
main()
{
    int *A, n;
    scanf("%d", &n);
    A=(int*)calloc(n, sizeof(int));
    A[0]=1;
    free(A);
}
```

Preprocesorske direktive

Preprocesorske direktive su mehanizam kojim je moguće dati kompajleru dodatne specifične instrukcije kako da prevodi program. Pozicija preprocesora u procesu prevođenja programa je sledeća:



U preprocesorske direktive spadaju:

1. simboličke konstante
2. makroi
3. uključivanje biblioteka
4. uslovno prevođenje programa

Simboličke konstante

Simboličke konstante su konstante za koje se ne vrši rezervacija memorijskog prostora. Jedini razlog za korišćenje ovih kosntanti jeste uvođenje simboličkih imena za vrednosti koje se inače teško pamte. Ključna reč za definisanje simboličkih konstanti je #define i definicija se piše na samom početku programa. Sintaksa je sledeća:

```
#define <simoličko_ime> ><vrednost>
```

Primer:

```
#define TRUE 1
#define FALSE 2
#define AND &&
#define OR ||
#define EQ ==
#include <stdio.h>

main()
{
    int x, y;
    ...
    if(x EQ y)
    ...
}
```

Nakon faze preprocesora sva pojavljivanja kosntante biće zamenjena konkretnim vrednostima. Simboličkim konstantama se u toku izvršenja programa ne mogu zadavati vrednosti.

Makroi

Makroi su simboličke konstante sa argumentima.

Primer:

```
#define kvadrat(x) (x*x)
main()
{
    int a, b;
    a=10;
    b=kvadrat(a);
}
```

Poziv kvadrat(a) nije poziv funkcije, iako liči. Za vreme preprocesiranja ovaj makro će biti zamenjen vrednošću iz definicije makroa sa konkretnim argumentom. Kod će postati:

```
main()
{
    int a, b;
    a=10;
    b=(a*a);
}
```

Primer: Makro za određivanje većeg od dva broja

```
#define MAX(a, b) (a>b)?a:b
main()
{
    int x=5, y=10, z;
    z=MAX(x,y);
}
```

Uklanjanje definicija

U bilo kom trenutku izvršenja programa definicija simboličke konstante se može ukloniti sledećom preprocesorskom direktivom:

```
#undef <simboličko_ime>
```

Od ove direktive pa nadalje konstanta više ne postoji i ne može se koristiti.

Primer:

```
#define Pi 3.1415
main()
{
    float x, y;
    x=2*Pi;
    #undef Pi
    y=Pi;     //Neispravno jer konstanta Pi više ne postoji
}
```

Uključivanje biblioteka

Biblioteke se u programskom jeziku C uključuju preprocesorskom direktivom na jedan od sledeća dva načina:

```
#include "ime_header_fajla"
#include <ime_header_fajla>
```

Razlika između ova dva načina je u skupu direktorijuma koje će kompajler pretraživati u potrazi za header fajlom.

Uslovno prevođenje programa

Koncept uslovnog prevođenja programa omogućava da se različiti delovi programa uključe ili isključe iz kompajliranja u zavisnosti od definisanih parametara. Primer upotrebe uslovnog prevođenja programa su programi pisani za različite OS gde je odredjeni problem na jednom OS potrebno rešiti drugačije nego na drugom OS. Uslovno prevođenje programa je f-ja pretprocesora koja je omogućena sledećim pretprocesorskim direktivama:

1. #ifdef
2. #ifndef
3. #if
4. #else
5. #elif
6. #endif

#ifdef

Pretprocesorska direktriva `#ifdef` može se naći bilo gde u programu. Ukoliko je simbolička konstanta koja stoji uz `#ifdef` definisana, deo programa između `#ifdef` i `#endif` će biti kompajlirano. Ukoliko nije definisana kompajler će preskočiti taj deo koda.

```
#ifdef <simbolička konstanta>
```

Ova direktiva se može implementirati i na sledeći način uz pomoć `#if` direktive:

```
#if defined <simbolička konstanta>
```

#ifndef

Pretprocesorska direktiva `#ifdef` uključuje deo koda ukoliko simbolička konstanta nije definisana. Ova direktiva se može implementirati i na sledeći način uz pomoć `#if` direktive:

```
#if !defined <simbolička konstanta>
```

#if

Deo koda obuhvaćen ovom pretprocesorkom direktivom biće uključen u proces prevođenja programa ukoliko je vrednost izraza tačna.

```
#if <izraz>
```

#else

Stoji uz `#if` i predstavlja NE granu te direktive.

#elif

Pretprocesorska direktiva `#elif` predstavlja kompinaciju `#if` i `#else` i koristi se za postavljanje dodatnog uslova u NE grani pretprocesorske direktive `#if`.

#endif

Pretprocesorska direktiva `#endif` zatvara blok za uslovno prevođenje programa.

Primer:

```
#define system unix
#if system == unix
        #define HDR "unix.h"
#endif
#if system == windows
        #define HDR "windows.h"
#endif
#if system == android
        #define HDR "android.h"
#endif
#include HDR
main()
{
    ...
    #if system == android
    ...
    #endif
    ...
}
```

Primer: Napisati program za određivanje faktrorijala. Preprocesorskim direktivama za uslovno prevođenje programa omogućiti prikaz svih među rezultata ukoliko je definisana simbolička konstnta.

```
#define provera
#include <stdio.h>
main()
{
    int n,F=1,i;
    scanf("%d", &n);
    for(i=2;i<=n;i++)
    {
            F*=i;
            #ifdef provera
                    printf("%d\n",F);
            #endif
    }
    printf("%d",F);
}
```