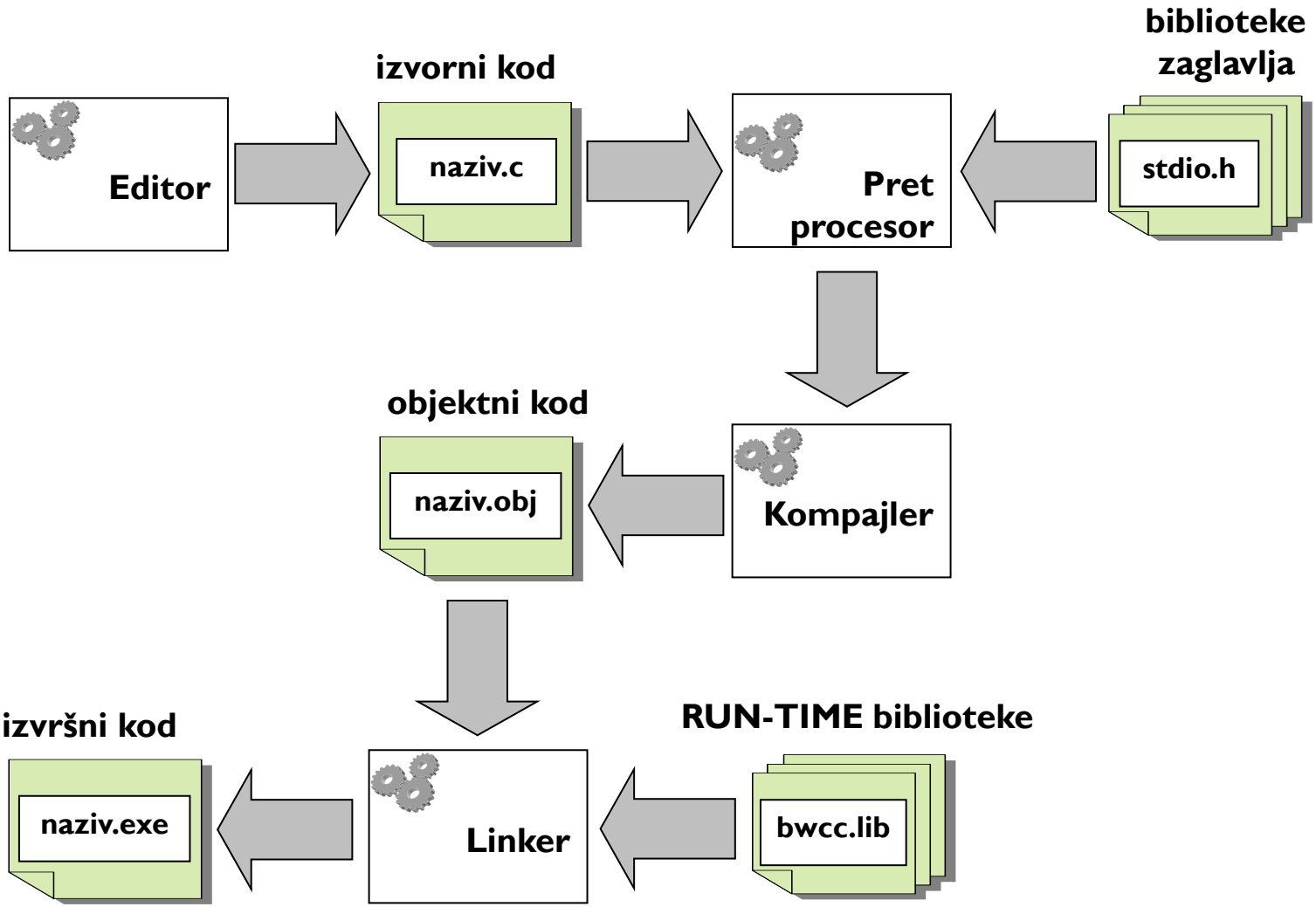
Proces prevođenja programa

Proces prevođenja programa



Izvor: D. Ivetić, *"Strukturirani pristup programiranju/Inženjering, algoritmi i programski jezici"*

Pretprocesor

Obrađuje (preprocesira) izvorni programski kod:

1. Eliminiše trigrafe (9 slučajeva – videti K&R)
   Primeri: ??[postaje { a ??- postaje ~ itd.
2. Svodi više blanko na jedan i eliminiše komentare
   // /* */
3. Zamenjuje specijalne karaktere sa ASCII kodovima
   \n u 10 13, \a u 07 ...
4. Identifikuje i izvršava pretprocesorske direktive

```
#include <stdio.h>
#include "naziv.h”
#define MAXPOSETILACA 350
#define CETVRTINA(VREDNOST) ((VREDNOST)/4)
#undef
#if ... #elif ... #else ... #endif
```

Pretprocesor – Primer

Direktiva #include

- Uključivanje zaglavlja datoteke
- Čita se i kopira sadržaj datoteke
- < ... > traži u standardnom include folderu
- " ... " traži u folderu u kojem se nalazi datoteka sa izvornim kodom
- Standardni ulaz/izlaz - *stdio.h*
- Funkcije opšte namene - *stdlib.h* (ovaj primer bi radio i bez stdlib)

```
#include <stdio.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Pretprocesor – Primer

```
#define CETVRTINA(VREDNOST)  ((VREDNOST)/4)
...
x = CETVRTINA(36);
...
x = ((36)/4);
x = 9;
```

```
#define MAXPOLAZNIKA 300
...
x = MAXPOLAZNIKA;
...
x = 300;
```

Funkcija main

Početna tačka svakog programa

Postoji samo jedna **main** funkcija!

Može biti navedena sa ili bez parametara – argumenti komandne linije

Upravlja tokom rada programa

```
#include <stdio.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Naredbe i programski blok

Programski blok u jeziku C čine naredbe zatvorene unutar para vitičastih zagrada **{ }**

Naredba je sekvenca sastavljena od jednog C izraza i separatora tačka-zarez **;**

Izraz predstavlja poziv funkcije, operaciju i sl.

```
{
   int a, int b = 113, float pi=3.14;
   printf("hello, world!\n");
}
```

Promenljive i tipovi podataka

Promenljive

Promenljiva je simboličko ime pridruženo lokaciji u memoriji računara.

Podaci se čuvaju u promenljivama, koncept sličan opštem broju u matematici. Za promenljive se zauzimaju lokacije u memoriji.

Dodela vrednosti promenljivoj (nije isto što i relacioni operator jednakosti u matematici): **a = a + b;** izračunava se vrednost izraza sa desne strane znaka = (suma trenutnih vrednosti *a* i *b*) i potom se rezultat pamti kao nova vrednost promenljive sa leve strane znaka =.

```
a = 5;        b = a + 13;        a = a + 2;
```

Promenljive

```
#include <stdio.h>

int main()
{
    int a = 5;
    int b;

    printf("Promenljiva a ima vrednost: %d\n", a);

    b = a + 13;
    a = a + 2;

    printf("\nPromenljiva b=a+13 ima vrednost: %d\n", b);

    printf("\nPromenljiva a=a+2 ima vrednost: %d\n", a);

    return 0;
}
```

Promenljive

Za svaku promenljivu definiše se:

- **Naziv** tj. **identifikator,** preko koga se identifikuje i koristi, mora imati deskriptivnu komponentu da bi se lakše identifikovala i koristila promenljiva
  - **C razlikuje mala i velika slova!**
- **Tip podatka** koji se čuva u promenljivoj (ceo broj, razlomljen broj, karakter, tekst, niz, niz od *n* brojeva, itd.) – **tip određuje šta predstavljaju bitovi tj. koliko mesta u memoriji zauzima promenljiva, iz kog skupa može uzimati vrednosti i koje operacije se mogu primenjivati**
- **Vrednost promenljive** (dobra praksa je uvek inicijalizovati vrednost promenljive)

```
int a = 5;        int b;   //Kolika je početna vrednost b?
```

Promenljive

Mesto deklarisanja promenljive nije ograničeno (može biti bilo gde u kodu)

Preporučuje se da se promenljive deklarišu na jednom mestu (najbolje na početku programskog bloka), jer se time olakšava održavanje programskog koda

Ako nije drugačije označeno, promenljiva važi (vidljiva je) od mesta deklarisanja do kraja programskog bloka

Programski blok (programsku celinu) određuju **{ }**

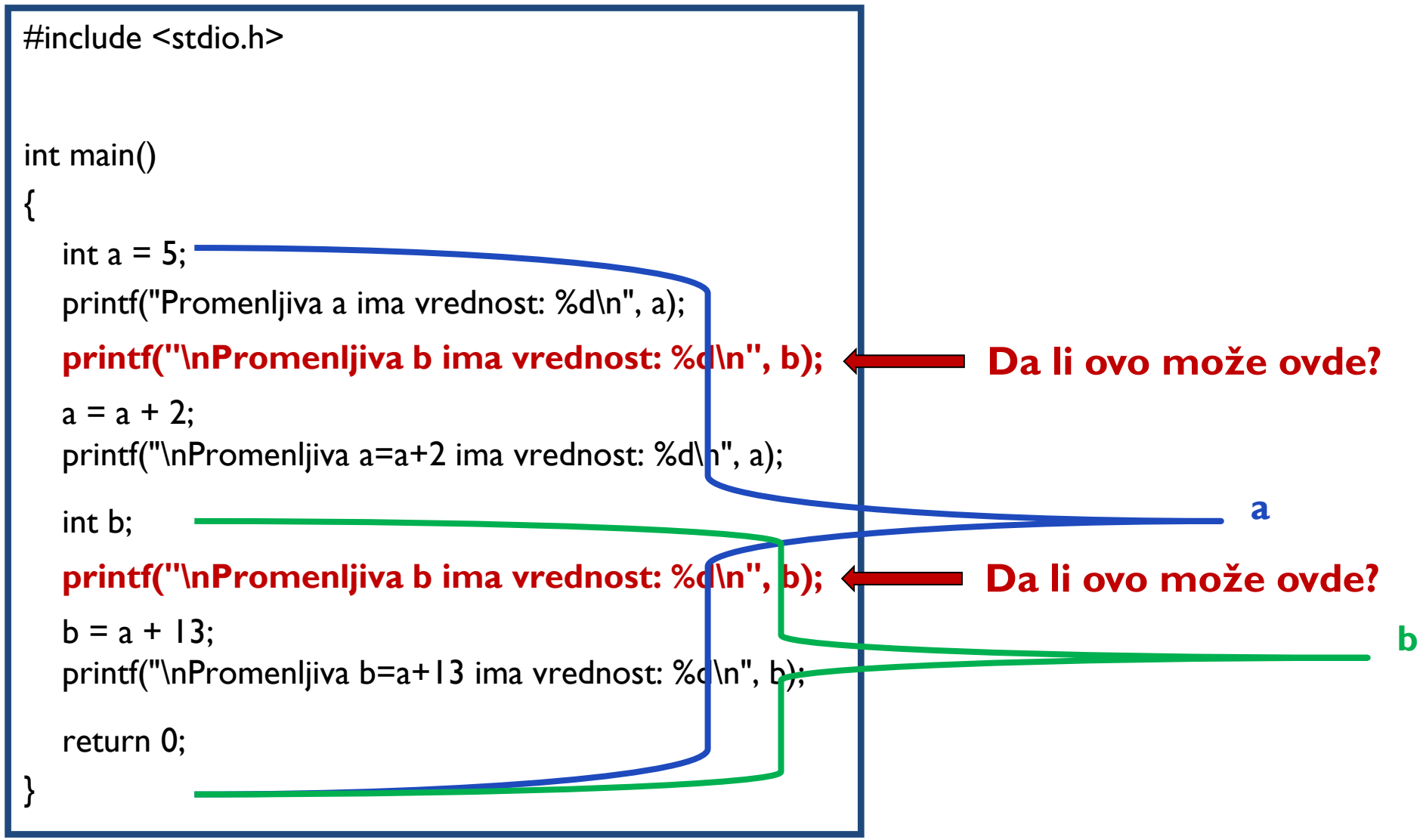
Doseg promenljive

```
#include <stdio.h>


int main()
{
    int a = 5;
    printf("Promenljiva a ima vrednost: %d\n", a);
    printf("\nPromenljiva b ima vrednost: %d\n", b);
    a = a + 2;
    printf("\nPromenljiva a=a+2 ima vrednost: %d\n", a);

    int b;
    printf("\nPromenljiva b ima vrednost: %d\n", b);
    b = a + 13;
    printf("\nPromenljiva b=a+13 ima vrednost: %d\n", b);

    return 0;
}
```



Tipovi podataka

Određuju:

- **količinu memorije koju će zauzeti promenljiva**
- **opseg mogućih vrednosti**
- **dozvoljene operacije**

Postoji više tipova podataka koji opisuju brojeve:

- **int, unsigned, short, long** (celi brojevi)
- **float, double, long double** (realni brojevi)
- **char** (karakter/pamti ceo broj koji predstavlja ASCII kod)
- **boolean (logički tip)? Ne postoji u jeziku C!**

Koliko koji tip zauzima memorije zavisi od konkretne implementacije prevodioca za programski jezik C

Tipovi podataka

Osnovni (skalarni) tipovi

- celi brojevi (označeni, neoznačeni, znakovni)
- realni brojevi (jednostruka/dvostruka preciznost)
- nabrojivi tip
- pokazivači

Tip void – prazan skup vrednosti

Složeni (izvedeni) tipovi

- nizovi
- strukture
- unije
- polja bitova

Tipovi podataka prema C99

Izvor: D. Ivetić, *"Strukturirani pristup programiranju/Inženjering, algoritmi i programski jezici"*

Celi brojevi

Celobrojne vrednosti, neoznačene i označene:

3, 7, -13, 12000

Najčešće korišćen celobrojni tip je **int**

unsigned/signed short/long int

signed int ili samo **int**

short int ili samo **short**

long int ili samo **long**

unsigned int ili samo **unsigned**

Znakovni tip

Karakteri:
'A', 'B', 'c', 'd', '1', '4', '#', '?', '\n'

Promenljive **char** tipa se koriste za čuvanje tekstualnih znakova:

- slova
- cifara
- specijalnih znakova
- neštampajućih (belih) znakova
- može se koristiti i za čuvanje malih celobrojnih vrednosti (**od 0 do 255 ili od -128 do 127**)

Znakovni tip

char je samo broj!

Svakom karakteru se pridružuje numerički kod

Postoje različiti skupovi kodova:

- ASCII (American Standard Code for Information Interchange) – 7 ili 8 bitova, najčešći
- EBCDIC – zastareo, retko se koristi
- Unicode – noviji, 16 bitova, dobija dominantu ulogu

U jeziku C se koristi ASCII kod

ASCII tabela

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Realni brojevi

Koriste se za predstavljanje realnih (razlomljenih brojeva):

12.22, 234.0004444, -11.4444

– **decimalna tačka, a ne decimalni zarez !**

Najčeće korišćen je **double**

float, double, long double

Kada se želi uštedeti na memorijskom prostoru, a raspon upotrebljenih vrednosti to dozvoljava, koristi se **float**

Kada se koriste izrazito velike vrednosti, ili je potrebno postići veliku preciznost, koristi se **double** (**long double**)

Primeri rada sa promenljivama

Koristiti %f kod printf umesto %d.

1. Program koji obračunava transakciju dinara u evro po kursu 117.5, pretpostaviti da se menja 155000 dinara
   / je operator deljenja

2. Program koji računa prosečnu potrošnju goriva na pređenih 100 kilometara, pređeno 350 km a potrošeno 17 l goriva
   *** je operator množenja

3. Program koji računa površinu i obim kruga poluprečnika

$$P = r^2\pi \qquad O = 2r\pi$$

Nabrojivi tip – enumeracija

Skup mogućih vrednosti je skup svih nabrojanih vrednosti

Svaka nabrojana vrednost kodira se16-bitnim brojem

```
enum {NE, DA};
enum temp {MIN = -15, RADNA = 20, MAX = 50};
enum vojnickiCin {RAZVODNIK, DESETAR,VODNIK};
enum meseci {JAN=1, FEB, MAR,APR, MAJ, JUN, JUL,AVG, SEP,
   OKT, NOV, DEC};
enum temp temperatura;  //moze -15, 20 ili 50

enum vojnickiCin vojnici;
vojnici = DESETAR;          // ali i sa vojnici = 1;
vojnici = vojnici + 1;      //koju vrednost sada ima vojnici?
```

Ove vrednosti mogu se koristiti kao ulaz/izlaz, ali samo u vidu int

Deklaracija promenljive

Promenljiva se deklariše, ali joj se ne dodeljuje inicijalna vrednost

tip identifikator;

```
int godina;
float pi;
short sat;
char znak;
```

lako vrednost nije eksplicitno navedena,

C kompajler dodeljuje neku vrednost

(ako imate sreće, to je **nula** ☺)

Definicija promenljive

Promenljiva se deklariše, ali joj se dodeljuje i inicijalna vrednost

tip identifikator = vrednost;

```
int godina = 2022;
float pi = 3.14;
short sat = 23;
char znak =  'a';
```

Dobra praksa je uvek inicijalizovati promenljivu na poznatu i željenu vrednost!

Inicijalizacija promenljive

Dodela vrednosti naknadno već deklarisanoj ili inicijalizovanoj promenljivi

identifikator = vrednost;

godina = 2022;

sat = sat – 1;

Da bi se promenljiva koristila mora prethodno biti deklarisana !

Konkretna vrednost zove se **neposredni operand** jer joj se vrednost neposredno zadaje i ne može se menjati

Memorijska klasa promenljive

Definiše u kojem delu memorije će se nalaziti promenljiva

Opciono se može navesti ispred oznake tipa u deklaraciji promenljive, ako se ne navede, podrazumeva se **auto**

<mem_klasa> tip identifikator;

Memorijske klase u C programskom jeziku su:

- **auto** – kreira se na početku bloka u kojem je deklarisana (u nekoj funkciji), traje do kraja bloka, zavisi da li je lokalna ili globalna promenljiva
- **register** – sugeriše kompajleru da smesti promenljivu u registar, ali kompajler ne mora to i da uradi
- **static** – promenljiva se kreira na početku programa i traje do kraja izvršavanja programa (vrednost promenljive se prenosi iz jednog izvršavanja funkcije u drugo izvršavanje funkcije)
- **extern** – označava da se deklaracija promenljive nalazi u nekoj eksternoj datoteci

Konstante

Konkretne vrednosti napisane direktno u okviru programskog koda koje se usled toga ne mogu menjati tokom izvršavanja

Netipske (simboličke)

- vrednost zamenjuje identifikator pre kompajliranja
- postaje neposredni operand

```
#define PUNOLETAN 18
#define PORUKALOZINKA "Unesite lozinku"
```

Tipske

- čuvaju se u memoriji, ali im se vrednost ne može menjati
- tip se ne mora eksplicitno navoditi (zaključuje se iz vrednosti)
- mora se dodeliti vrednost

```
const punoletan = 18;
const short radniDanSati = 8;
const double pi = 3.14159265358979323846264433;
```

Primeri rada sa konstantama

Šta su kandidati za konstante u prethodnim primerima?

kurs

kmprosek

pi

Deklaracije tipova

Korišćenjem ključne reči **typedef** postojećim tipovima podataka daju se nova imena (**ne nastaju novi objekti, niti novi tipovi!**)

typedef tip novoImeTipa;

novoImeTipa postaje sinonim za **tip**

Kasnije se u kodu koriste ravnopravno sa istim značenjem

Svrha i korisnost deklaracije tipova se uočava tek kod složenih tipova:

- omogućuje lakše razumevanje (čitanje) programskog koda
- omogućuje lakše dokumentovanje programa

Primeri deklaracije tipova

```
typedef int brzina;

...

brzina prva, druga, treca;
```

```
typedef  int daniuMesecu[31];

...

daniuMesecu januar, mart;
```

Pokazivači

Pokazivači

Pokazivač nosi indirektni podatak:
adresu memorijske lokacije gde se nalazi stvarni podatak

Tretira se kao skalar, jer je to zapravo broj
(adresa lokacije u memoriji)

Pokazivačke promenljive pokazuju na druge promenljive
ili na početak memorijskog bloka
(pokazivači na promenljive tipe *int*, *float*, i sl.)

Deklaracija pokazivača:

tip *pok;

pokazivač **pok** na promenljivu tipa **tip**

Pokazivači

Definisanje pokazivača:

tip *pok = &prom;

pokazivač **pok** na promenljivu tipa **tip** koji čuva adresu promenljive **prom**

Primenom unarnog operatora * posredno se pristupa nekom podatku pomoću adrese (**dereferenciranje**)

Inicijalizacija pokazivača:

pok = &prom;

pokazivaču **pok** se dodeljuje adresa promenljive **prom** (**pok** pokazuje na **prom**)

Unarni operator **&** daje adresu promenljive (**referenciranje**)

Pokazivači

Pristup vrednosti na koju pokazuje pokazivač:

prom2 = *pok;

promenljiva **prom2** dobija vrednost sa lokacije na koju pokazuje pokazivač **pok**

***pok =prom3;**

u lokaciju na koju pokazuje pokazivač **pok** upisuje se vrednost promenljive **prom3**

prom4 = pok;

promenljiva **prom4** dobija adresu lokacije na koju pokazuje pokazivač **pok**

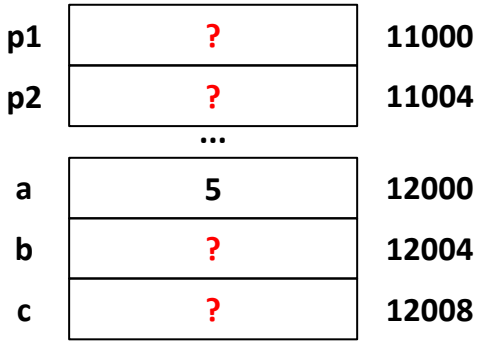
pok =prom5;

u pokazivač **pok** upisuje se vrednost promenljive **prom5**

Pokazivači – primeri

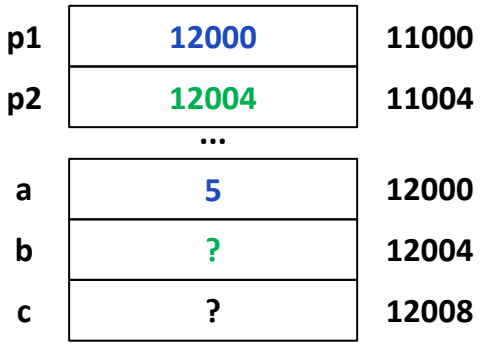
Deklarisanje:

```
int a = 5;
int b, c;
int *p1, *p2;
```



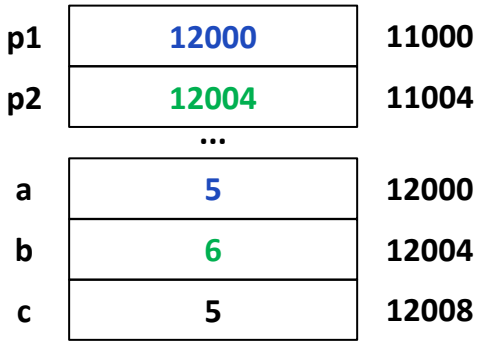
Dodela vrednosti:

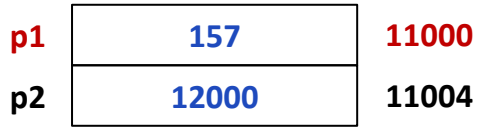
```
p1 = &a;
p2 = &b;
```



Pristup lokaciji:

```
c = *p1;
*p2 = 6;
p2 = p1;
p1 = 157;
```





Operacije sa pokazivačima

Dodela vrednosti dovodi do toga da oba pokazivača pokazuju na isto

```
int a = 5;
int *p1, *p2;
p1 = &a;
p2 = p1;        // p1 i p2 sada pokazuju na istu lokaciju
```

Mogu se sabirati/oduzimati sa brojem

```
p1 = p1 + 10;  // ako je p1 bilo 12000, posle sabiranja biće 12040
p2++;   // ako je p2 bilo 12002, posle inkrementiranja biće 12006
        // zbog int koji zauzima 4 bajta
```

Mogu se porediti

```
if (p1 == p2)   // proverava se da li su iste adrese
```

NULL konstanta

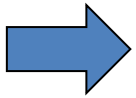
Definisana u zaglavlju **<stdio.h>**

Ako pozivač ima vrednost **NULL**,
onda ne pokazuje ni na šta

Primer:

```
int *p;
p = NULL;
```

Neinicijalizovana promenljiva nema vrednost NULL !
(mora se eksplicitno inicijalizovati)

```
int *p;
```

```
p != NULL
```

void tip

Konstante i promenljive ne mogu biti tipa **void**, već samo funkcije, lista parametara funkcije, izrazi i pokazivači

Generički tip pokazivača – **void**

```
void *pokVoid;
```

pokazivač **pokVoid** ne može do podatka, jer on ne pokazuje na konkretan tip

Može primiti vrednost drugih pokazivača:

```
int *pokInt;
pokVoid = pokInt;
```

Sada **pokInt** i **pokVoid** pokazuju na istu memorijsku lokaciju

Ipak, do podatka se može doći samo eksplicitnom konverzijom tipa:

```
int a = (int *) pokVoid + 12;
```

Rad sa pokazivačima

Dinamička alokacija memorije:

malloc funkcija, alocira memoriju tražene veličine i vraća pokazivač na alociranu memoriju (ako je uspelo) ili NULL (ako alociranje nije uspelo)

```
int *pokInt = (int *)malloc(10*sizeof(int));
```

- **sizeof** je operator za određivanje količine memorije koju zauzima neka promenljiva ili tip

calloc funkcija, alocira blok kontinualne memorije za **N** elemenata određene veličine, i za razliku od **malloc** postavlja vrednosti svih lokacija na 0, vraća pokazivač na alociranu memoriju (ako je uspelo) ili NULL (ako alociranje nije uspelo)

```
double *pokNizDoble = (double *)calloc(10,
sizeof(double));
```

Rad sa pokazivačima

realloc funkcija, pokušava da proširi alociranu memoriju tako da odgovara traženoj veličini, vraća pokazivač na alociranu memoriju (ako je uspelo) ili NULL (ako alociranje nije uspelo)

```
double *pokNizDouble = (double*)
realloc(pokNizDouble , 15* sizeof(double));
```

Kada se završi sa radom, alociranu memoriju je potrebno osloboditi, C program to ne radi automatski

free funkcija, koristi se za oslobađanje (dealociranje) dinamički zauzete memorije (**malloc, calloc, realloc**), ne vraća nikakvu vrednost

```
free (pokInt);
free (pokNizDouble);
```

Česte greške u radu sa pokazivačima

Iako je moguće, da li je smisleno deklarisati pokazivač na konstantu?

```
const int *pok;
```

Isto važi i za konstantan pokazivač na promenljivu

```
int * const pok = &a;
```

Nemoguće je promeniti adresu promenljive (ne zavisi od nas)

Sledeći izrazi su pogrešni:

```
p1 = &5;
p2 = &(i + 7);
p3 = &(x == y);
&x = &y;
&x = 150;
```