П1 – Слајдови са предавања

Програмирање је **вештина** у којој....
... програмер у програмском језику **прави** (ствара, пише, креира, твори)
програм извршив на рачунару, за задовољење потреба корисника.

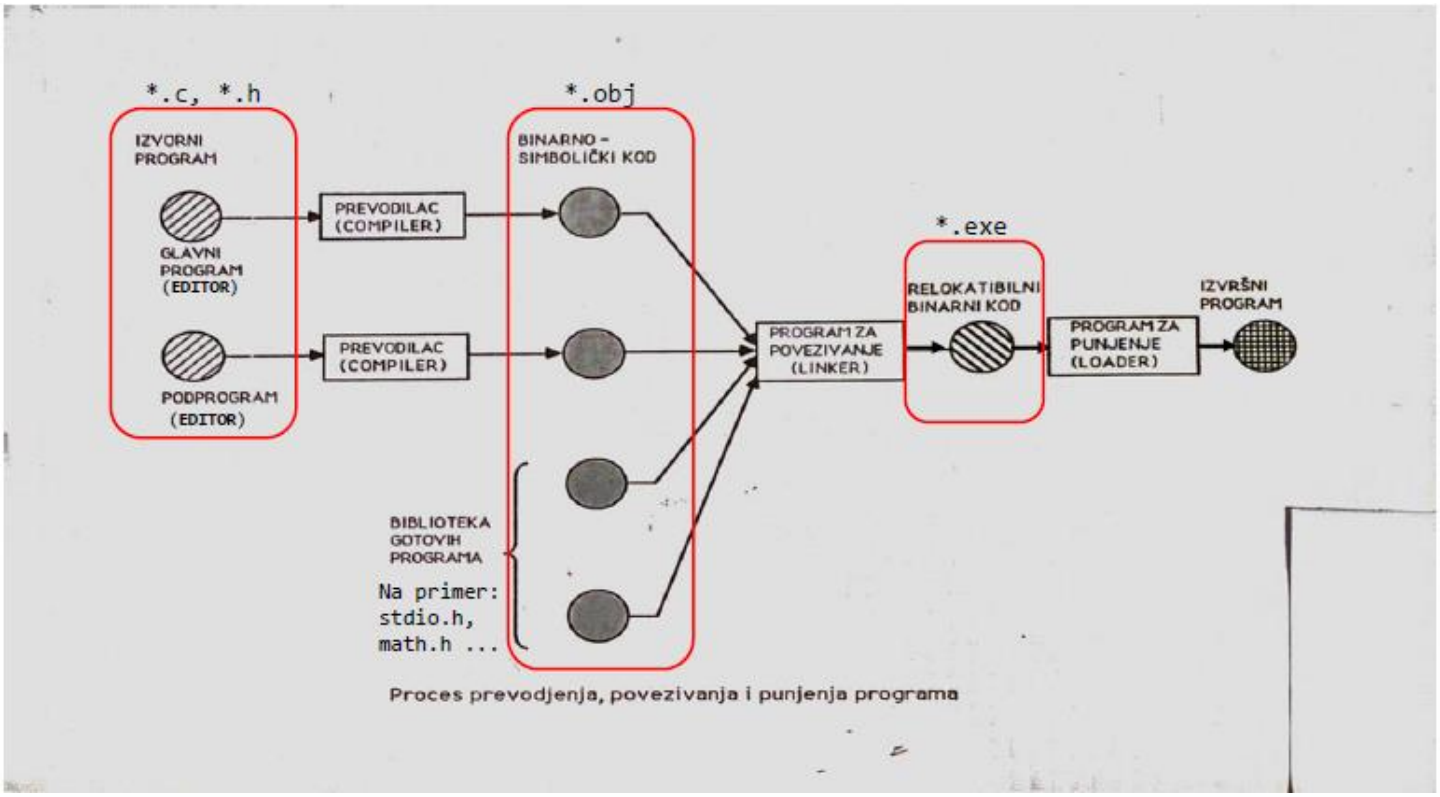

Figure 1 ***Proces prevodjenja iz .C u .Exe***

Презентација 2

Карактеристике програмског језика Ц:

- Виши програмски језик (по структурираним типовима података и управљачким структурама), али са особинама машински зависних програмских језика (по манипулацији битовима, коришћењу процесорских регистара, приступу подацима помоћу адресе и хардверски оријентисаним операторима)
- Императивни језик
- Модуларни језик
- Компактан језик (мали скуп резервисаних речи)
- *Case-sensitive* и *free-form* језик

Елементи програмског језика Ц 1/5

Скуп знакова (*character set*) – мала и велика слова енглеске абецеде (a – z, A – Z), десет декадних цифара (0 - 9), двадесет девет знакова интерпункције:
! " # % & ' * ( ) + - / : ; ^ < > ~ , ? = [ ] _ { . \ | }

- Дирафи:
- Триграфи:

Дирaф	Еквивалент
<:	[
:>	]
<%	{
%>	}
%:	#
%:%:	##

Триграф	Еквивалент
??(	[
??)	]
??<	{
??>	}
??=	#
??/	\
??!	|
??'	^
??-	~

Лексички симбол = лексема = токен (*token*) – недељиви низ знакова, најмања програмска семантичка јединица.
Лексеме се деле на: идентификаторе, константе, резервисане речи, операторе и сепараторе
Бели знакови (*white spaces*) – размак, хоризонтална табулација, вертикална табулација, нови ред, нова страна

Коментари (*comments*) –
Вишередни или блок коментар: /* */ Једноредни коментар: //

Директиве препроцесора (*preprocessor directives*) – упутства помоћу којих може да се утиче на ток превођења програма. Оне нису део програма, па се зато не убрајају у исказе. Њих узима у обзир претпроцесор, који врши припрему изворног кôда пре самог превођења. Једна директива – један ред: свака директива пише се у засебном реду и мора да почиње знаком #. Не смеју да се мешају са исказима програма

Искази (*statements*) – низови лексичких симбола. Деле се на:
- Декларативне исказе (*declarativ st.*) којима се декларишу делови програма (подаци, потпрограми итд)
- Извршне (егзекутивне) исказе (*executive st.*) = наредбе којима се одређују акције програма

Идентификатори (*identifiers*) – служе за означавање (именовање) свих врста елемената програма.

❖Идентификатори могу да се састоје од слова, цифара и доње црте (_). Први знак не сме да буде цифра

❖Прави се разлика између великих и малих слова:
procenat ≠ Procenat ≠ PROCENAT

❖Идентификатор може бити произвољне дужине, али се најмање првих 31 знакова користи за идентификацију

Функција (*function*) — основни градивни елемент програмског језика Ц; састоји се из секвенце исказа; унутар функције се наредбе (тј. извршни искази) могу груписати у блокове (*blocks*)

❖Две врсте функција:
- Готове функције из стадардне библиотеке - прототипови ф-ја из стандардне библиотеке налазе се у датотекама заглавља (*header files, *.h*)
- Програмерски креиране функције, налазе се у датотекама изворног кода (*source code files, *.c*); могу да се налазе у програмерски креираним библиотекама *(*.h + *.obj*)

❖Дефиниције функција не могу се угнездити једна у другу, тј. не постоје подфункције

❖Функције могу међусобно да се позивају; функција може и саму себе да позива (рекурзија)

У ПЈ Ц постоје две врсте изворних датотека:
1)*.c – изворне датотеке, садрже програмски кôд
2)*.h – датотеке заглавља, садрже прототипове функција, дефиниције константи и дефиниције макроа

Област важења идентификатора :
Постоје четири врсте области важења:
1.**Област датотеке** (*file scope*) – ако се идентификатор декларише изван свих блокова и листа параметара, његова област важења је датотека. Тада се може користити свуда унутар датотеке, почев од места декларације па до њеног краја

2.**Област блока** (*block scope*) – сви идентификатори декларисани у блоку, изузев лабела, важећи су у том блоку. Декларације се не морају налазити пре наредби у блоку. Нпр. тело дефиниције функције чини један блок

3.**Област прототипа функције** (*function prototype scope*) – имена параметара у прототипу ф-је припадају области важења тог прототипа. Пошто имена параметара немају значење ван прототипа, корисна су само за подсећање, па се зато могу изоставити

4.**Област функције** (*function scope*) – област важења лабеле увек је блок ф-је у којој се појављује, чак и када је блок угнежђен

Тип података је алгебарска структура и одређује се као:
- скуп вредности /домен/,
- скуп операција над датим скупом вредности /интерфејс/ и
- скуп константи.

ТИП ПОДАТАКА = СВ + СО + СК
тип података је одређен меморијском репрезентацијом

У ПЈ Ц, појам **објекат** означава место у меморији чији садржај може да представља вредност.

❖Објекти који имају имена називају се **променљиве** (*variables*)

❖Објекти који имају имена и додељену вредност која се не може мењати називају се **константе** (*constants*)

❖ПЈ Ц познаје само нумеричке типове података

Уколико се у програм укључи заглавље *stdbool.h* могу се користити идентификатори:
- bool (макро, синоним за *_Bool*),
- true (симболичка константа, вредност 1) и
- false (симболичка константа, вредност 0)

❖Набројиви тип или енумерација (*enumeration, enumerated type*) је целобројни тип који програмер сам дефинише у програму:

```
enum boja { CRVENA, BELA, PLAVA };
```

❖Енумерационе константе *CRVENA, BELA, PLAVA* имају вредности 0, 1 и 2, респективно.
Могућа је и додела сопствених вредности:

```
enum boja { CRVENA, BELA, PLAVA, SIVA = 8, BRAON};
```

❖Литерал је лексема која означава непроменљиву вредност. Та вредност може бити број, знак или ниска (низ знакова, знаковни низ).

❖Управљачки знаци почињу обрнутом косом цртом и представљају један знак
❖Важнији упр.знаци:

- \' полунаводник
- \" наводник
- \? знак питања
- \\ обрнута коса црта
- \a упозорење (*alert*)
- \b брисање уназад (*backspace*)
- \f нова страна (*form feed*)
- \n нови ред (*new line*)
- \r почетак реда (*carriage return*)
- \t хоризонтални табулатор
- \v вертикални табулатор

Вишередни текст:

```
char *info = "Ovo je tekst\
napisan \n u tri\
reda, a bice prikazan u dva. Zasto\?\n";
```

Презентација 5

❖**Низ** (array) је колекција података; садржи податке истога типа који се чувају у суседним меморијским локацијама. Ти подаци се називају елементи низа.

❖Капацитет (димензија) низа = максималан број елемената које низ може да садржи

❖Број елемената низа = тренутни број (искоришћених) елемената низа

❖Величина низа = број бајтова (By) које заузима низ

❖Елементима низа се приступа преко индекса.

❖Ниска = низ знакова = знаковни низ = стринг (string)
❖Ниска је:
непрекидна секвенца знакова која се завршава NULL знаком
ИЛИ
низ чији су елементи типа char и последњи елемент има вредност ‘\0‘
❖NULL знак = ‘\0‘

❖Дужина ниске = број знакова, не рачунајући завршни NULL знак

Презентација 7

❖Вишедимензиони низ (*multidimensional array*) је низ чији су елементи такође низови.
❖Тродимензиони низ:

```
int 3DNiz [3] [10] [5];
```

❖У низу 3DNiz има 3 x 10 x 5 = 150 елемената. Први елемент је 3DNiz[0][0][0], а последњи елемент је 3DNiz[2][9][4].

❖Дводимензиони низ се назива матрица (*matrix*). Често се користи. Елементи матрице распоређени су у редове (*rows*) и колоне (*columns*). На пример, матрица од три реда и пет колона:

```
int matrica[3][5];
```

❖Три елемента *matrica[0], matrica[1]* и *matrica[2]* су редови матрице *matrica*. Сваки ред представља низ од пет елемената типа *int*.

	0	1	2	3	4
matrica[0]	10	11	12	13	14
matrica[1]	11	12	13	14	15
matrica[2]	12	13	14	15	16

Презентација 8

❖Показивач (*pointer*) је референца (упутница) на податак или функцију
❖Показивач на податак који има вредност ***NULL*** значи да не показује ни на шта
❖Функција која враћа ***NULL*** значи (најчешће) да је дошло до неуспеха

❖Пошто објекти типа ***void*** не постоје, тип ***void **** се користи као тип показивача за све намене (генерички показивач, показивач на било шта)
❖Другим речима, показивач на тип ***void*** може да садржи адресу било ког објекта, али без „знања“ о типу тог објекта. Зато, да би се приступило објекту у меморији, показивач на тип ***void*** мора да се прво претворити у показивач на неки конкретан тип објекта, па тек онда може да се користи. Пример:

```
int i = 11, *intPok;
void *voidPok;
intPok = &i; // OK
voidPok = &i; // OK
printf("\n*intPok = %d\n", *intPok); // OK
printf("\n*voidPok = %d", *voidPok); // ERR
printf("\n*voidPok = %d\n", *( (int *) voidPok) ); // OK
```

❖Може и обрнуто; тада се врши имплицитна конверзија конкретног показивача у **void ***. Пример:

```
void * memset(void * s, int c, size_t n);
void * malloc(size_t size);
```

❖Ф-ја *memset()* додељује вредност ***c*** сваком од ***n*** бајтова у меморији у блоку који почиње на адреси ***s***. На пример, у наредном позиву ф-је додељује се вредност 0 сваком бајту у променљивој низ:

```
int niz[5];
memset(niz, 0, sizeof(niz));
//ILI: memset(&niz[0], 0, sizeof(niz));
```

❖Декларација показивача може да садржи идентификаторе типа:

- *const* (сталан, непроменљив) – текући процес не може мењати објекат после његове дефиниције),
- *volatile* (несталан, променљив) – текући процес, али и други процеси или догађаји, могу мењати објекат после његове дефиниције) и/или
- *restrict* (ограничен) – током животног века показивача, објекат на који показује показивач се не може мењати нити му се може приступити на неки други начин, сем помоћу тог ограниченог показивача.

❖Константни показивачи и показивачи на константне објекте:

- Када се дефинише константан показивач, мора се и иницијализовати, јер касније неће моћи да се мења,
- Константан показивач не мора да показује на константан објекат.

```
int var; // Objekat tipa int
const int c_var = 100; // Konstantan objekat tipa int
const int *ptr_2_c; // Pokazivač na konst.obj.tipa int
ptr_2_c = &c_var; // OK: ptr_2_c pokazuje na c_var
var = *ptr_2_c + 11; // OK
ptr_2_c = &var; // OK: ptr_2_c pokazuje na var
if ( c_var < (*ptr_2_c) ) // OK: pristup samo za čitanje
*ptr_2_c = 33; // ERR: ne može se menjati var pomoću ptr_2_c, iako var nije
konstanta
```

Презентација 12

Tranzientni i perzistentni podaci.

CPU ↔ Cache ↔ Memory ↔ Disk / CD / DVD / Flash Mem ↔ Traka (arhivska memorija)

Hijerarhija memorije: brzina, kapacitet, cena, trajnost

Datoteka sadrži semantički srodne podatke.

Pristup podacima u datoteci:
- sekvencijalni pristup (sequential access)
- direktan pristup (random access)

Datoteke po formatu podataka:
- tekstne
- binarne
  - tipizirane,
  - netipizirane

Datoteteke po organizaciji podataka:
- redna
  - uređenost: hronološka
  - pristup : sekvencijalan
- serijska (= sortirana redna)
  - uređenost: semantička (po vrednosti ključa)
  - pristup : sekvencijalan
- rasuta (direktna ili hash)
- indeksna
- indeks-sekvencijalna
- ostale org.dat.

Operacije nad datotekama:
- kreiranje (create, make)
- otvaranje (open)
- čitanje (read)
- upisivanje (write)
- ispitivanje statusa datoteke
- zatvaranje (close)
- manipulacija (preimenovanje, kopiranje, premeštanje, poređenje, brisanje)

Vrste tokova podataka (u PJ C):
1) tekstni tokovi,

2) binarni tokovi.

Datoteka (file) je imenovana sekvenca bajtova koja se čuva na trajnom (perzistentnom) memorijskom nosiocu (disk, CD/DVD, flash mem, itd).
Funkcije za rad sa datotekom: čitanje, pisanje, pozicioniranje, obrada greške.

Bafer (prihvatna memorija):
Kada se koriste datoteke, obično nije efikasno čitati ili upisivati znakove pojedinačno. Zato tok podataka ima bafer za čuvanje znakova koji se prenose kao blokovi iz uli u datoteku. Međutim, upotreba bafera nije uvek pogodna. Prenošeje sadržaja toka podataka u odnosu na bafer može se odvijati na tri načina:
- po punjenju bafera (fully buffered),
- posle znaka za novi red (line buffered),
- bez baferovanja (unbuffered).

Znakovi iz izlaznog bafera toka podataka moguće je direktno upisati u pridruženu datoteku upotrebom funkcije:
fflush()
Bafer se sam prazni (flush) kada se zatvori tok podataka. Na kraju programa prazne se svi baferi svih tokova podataka otvorenih u programu.

Način pražnjena bafera je moguće izmeniti upotrebom funkcija:
1) setbuf() i
2) setvbuf().

Standardni tokovi podataka
Svakom programu (na jeziku C) od njegovog početka su dostupna tri standardna toka podataka:

1) stdin - standardni ulazni tok, uobičajeno je to tastatura;
2) stdout - standardni izlazni tok, uobičajeno je to ekran;
3) stderr - standardni izlazni tok greške, uobičajeno je to ekran.

Otvaranje i zatvaranje datoteka
Način (režim) pristupa (access mode):
- čitanje (read),
- čitanje ili pisanje (upisivanje) (read/write).

Funkcije za otvaranje datoteke:
1) fopen(),
2) freopen() i
3) tmpfile().

Rezim pristupa:
Moguće vrednosti za tekstnu datoteku:
1) "r" - otvaranje postojeće tekstne datoteke za čitanje, FPI = 0;
2) "w" - otvaranje nove tekst.dat. za upisivanje; ukoliko datoteka već postoji biće obrisana, a ako ne postoji

biće kreirana nova, FPI = 0;

3) "a" - otvaranje tekst.dat. za pisanje; ako datoteka postoji, novi podaci biće dodati na kraj datoteke, FPI = Length(File); ako datoteka ne postoji, biće kreirana nova, FPI = 0;
4) "r+" - otvara tek.dat. za ažuriranje, tj. i za čitanje i za upisivanje, FPI = 0;
5) "w+" - otvaranje nove tekst.dat. za ažuriranje (i čitanje i upisivanje); ukoliko datoteka već postoji biće obrisana, a ako ne postoji biće kreirana nova, FPI = 0;
6) "a+" - otvaranje tekst.dat. za ažuriranje (i čitanje i upisivanje); ako datoteka postoji, novi podaci biće dodati na kraj datoteke, FPI = Length(File); ako datoteka ne postoji, biće kreirana nova, FPI = 0.

Moguće vrednosti za binarnu datoteku:

1) "rb" -
2) "wb" -
3) "ab" -
4) "r+b" -
5) "w+b" -
6) "a+b" –

Vrednosti za origin:

SEEK_SET	0	početak datoteke
SEEK_CUR	1	tekuća pozicija u datoteci
SEEK_END	2	kraj datoteke

АТП (апстрактни тип података) - листа

Листа (списак, попис; енгл. *list*) представља потпуно уређену линеарну структуру података; то значи да у листи од n елемената:

$e_1, e_2, ..., e_{i-1}, e_i, e_{i+1}, ..., e_n$, где $i, n \in N$

Сваки елемент листе има по једног непосредног претходника и следбеника, сем првог елемента који нема претходника и последњег елемента који нема следбеника. Листа без елемената је листа, такође. За означавање празне листе користи се константа *Null*. Садржај елемента листе (тј. његова вредност, податак који елемент садржи) може бити било ког типа.
Уколико приликом извршења операција над листом дође до грешке, враћа се константа *ErrorList*.

Домен АТП Листа је коначан скуп елемената такав да важи:

- Празан скуп елемената је листа. Ова листа назива се празна листа и означава се константом ***Null***.
- Ако је ***e*** елемент, а ***lst*** листа, онда је уређени пар ***(e, lst)*** такође листа. У овом случају, ***e*** је глава листе ***(e, lst)***, а ***lst*** је реп те листе.