Napredni C kurs

Namenski računarski sistemi

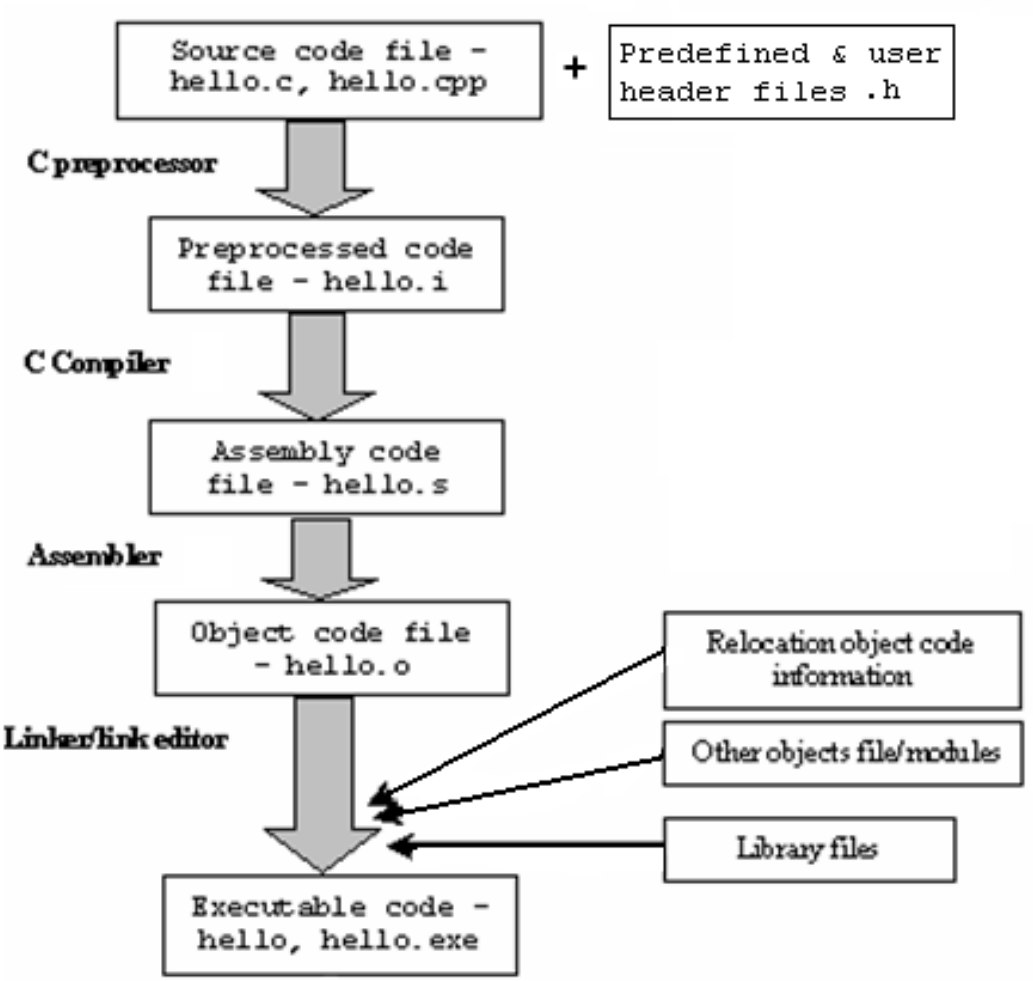
Struktura C programa

- Konstante i tipovi podataka
- Promenljive (lokalne, globalne, eksterne…)
- Izrazi (if, while, for…)
- Operatori (+, -, *, /,~…)
- Funkcije sa argumentima
  - Prenos po vrednosti
  - Prenos po adresi
  - Prenos po referenci (C#)
- Makro izrazi
- Preprocesorske direktive
- Biblioteke
- Komentari

```
#include <stdio.h>
#include "mydef.h"

#define ....          /* lokalne definicije */
int gvInt;            // globalna promenljiva
extern ...            /* referenca na gvar */
int func1(...);       /* redeklaracija */

void main( void )
{
    int i;            /* lokalna promenljiva */
    izraz 1;          /* komentar */
    func1(i, &i);     /* poziv funkcije */
    izraz 2;
    .....
}
int func( int k, int *pk )
{
    izraz3;           /* telo funkcije */
    return 0;
}
```

Prevođenje C programa

- file_name.c
  - C izvorni kod
- file_name.h
  - C header file
- file_name.i
  - Preprocesirani kod
- file_name.s, .asm
  - Asemblerski kod
  - Asemblerski kod koji se mora preprocesirati
- file_name.o, .obj
  - Objektni kod
- file_name.exe
  - Izvršni kod



Faze prevođenja C programa

- Izvorni kod -> Asemblerski kod -> Objektni kod -> Izvršni kod

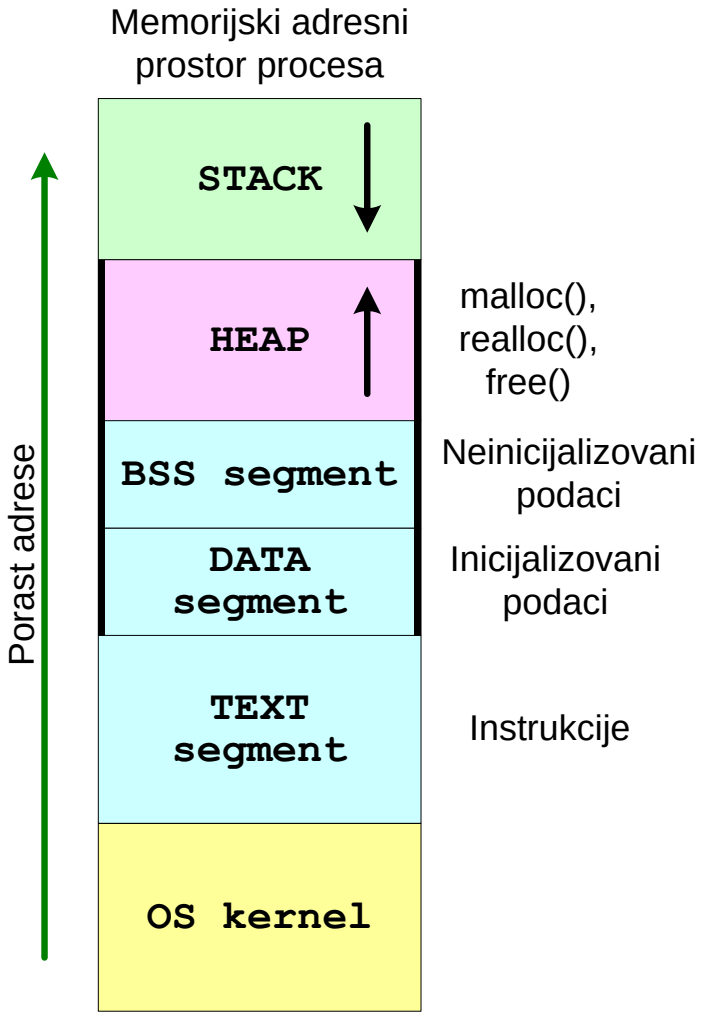
- Preprocesor – prvi prolaz prevođenja C koda
- Kompajler – drugi prolaz, generiše asemblerski kod
- Asembler – generiše objektni kod i asemblerski listing
- Linker – final faza gde se
  - Više .obj i .lib modula kombinuju u jedan izvršni (.exe)
  - Rešavaju reference na spoljne simbole
  - Vrši konačno dodeljivanje adresa funkcijama/promenljivim (relokacija)
- U IDE kompajlerima ove procedure su integrisane

Programski model

- U krajnjoj posledici programski model je
  - Organizacija promenljivih i funkcija, i načina međusobnog povezivanja, u okviru izvršnog programa
  - Iako zavisan od HW platforme, sličan u većini implementacija
- Postaje značajan u krajnjim fazama C prevođenja, zato pod programskim modelom podrazumevamo
  - Konvencije generisanja asemblerskog koda
    - Memorijska alokacija promenljivih
    - Registarske konvencije i korišćenje stack-a
  - Garancija međusobne kompatibilnosti
    - Povezivanje programa različitih kompajlera
    - Aplikativnih programa i operativnih sistema

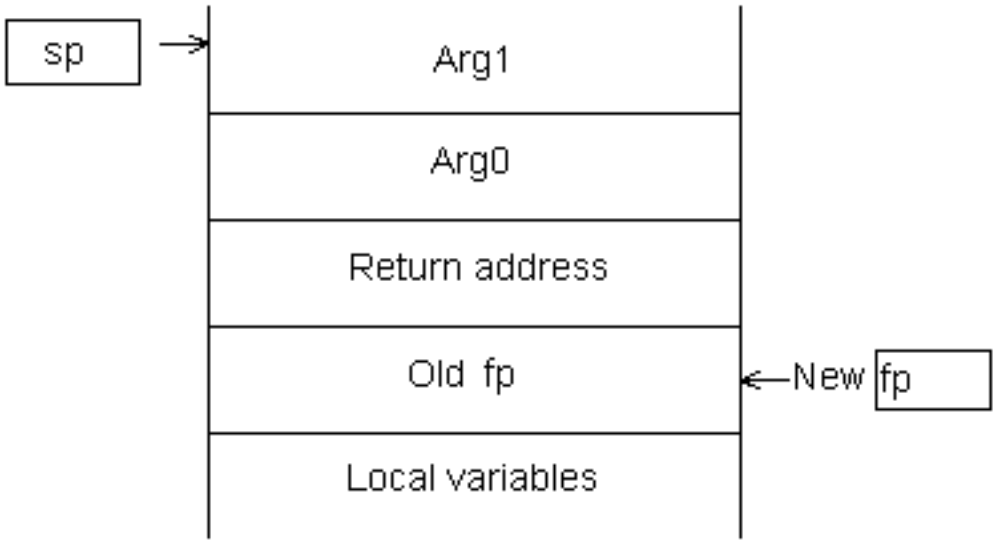
Memorijski segmenti programa

- Svaki program dobija svoj prostor pri punjenju
- TEXT sekcija može biti deljena
  - reentrant code – programs
- DATA – non-zero init global i static
  - .rdata – read only (const)
- BSS (Block Started by Symbol) – u izvršavanju DATA, u .exe ga nema
- HEAP – dinamička memorija
  - Pristup samo poreko pokazivača
  - Kontrola opsega (kompajler, run-time)
- STACK – lokalne promenljive, poziv funkcija i prenos argumenata
  - Stack frame, stacksize



Korišćenje stack-a pri pozivu funkcija

- Stack frame – osnovna struktura
  - Formira se i briše pre/po svakom pozivu funkcije na tekućoj lokaciji stack-a
  - Sadrži argumente, povratnu adresu i lokalne promenljive

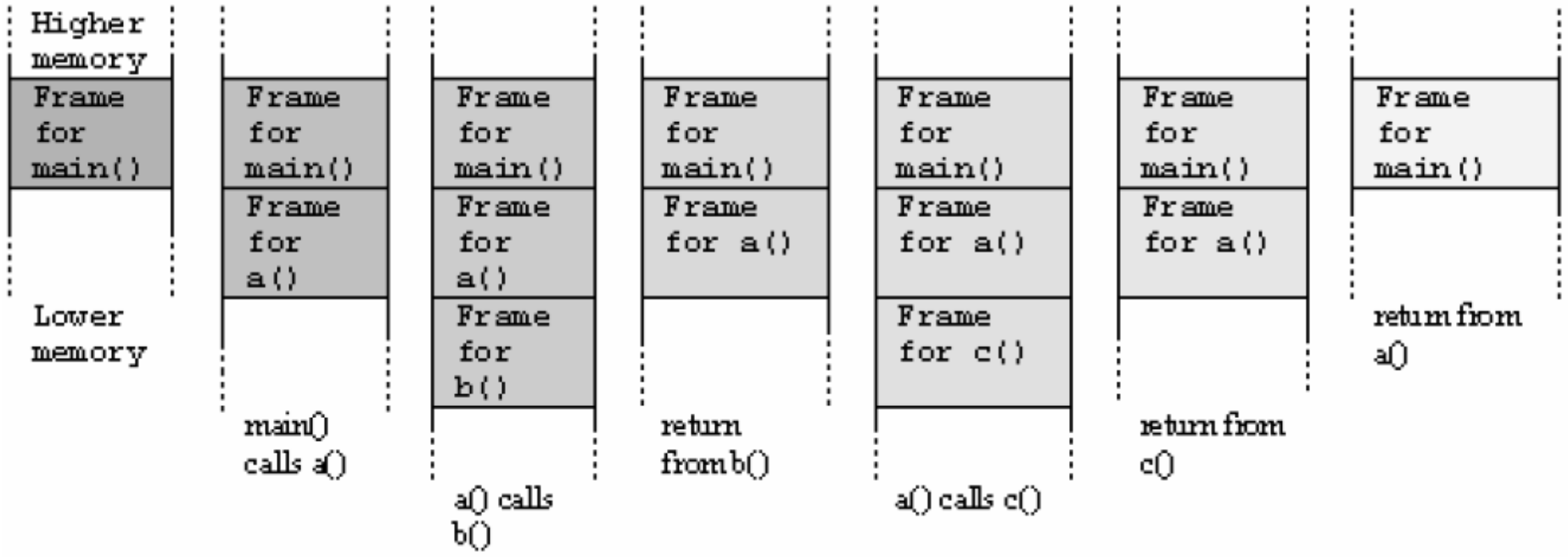


Izgled stack-a pri pozivanju funkcija

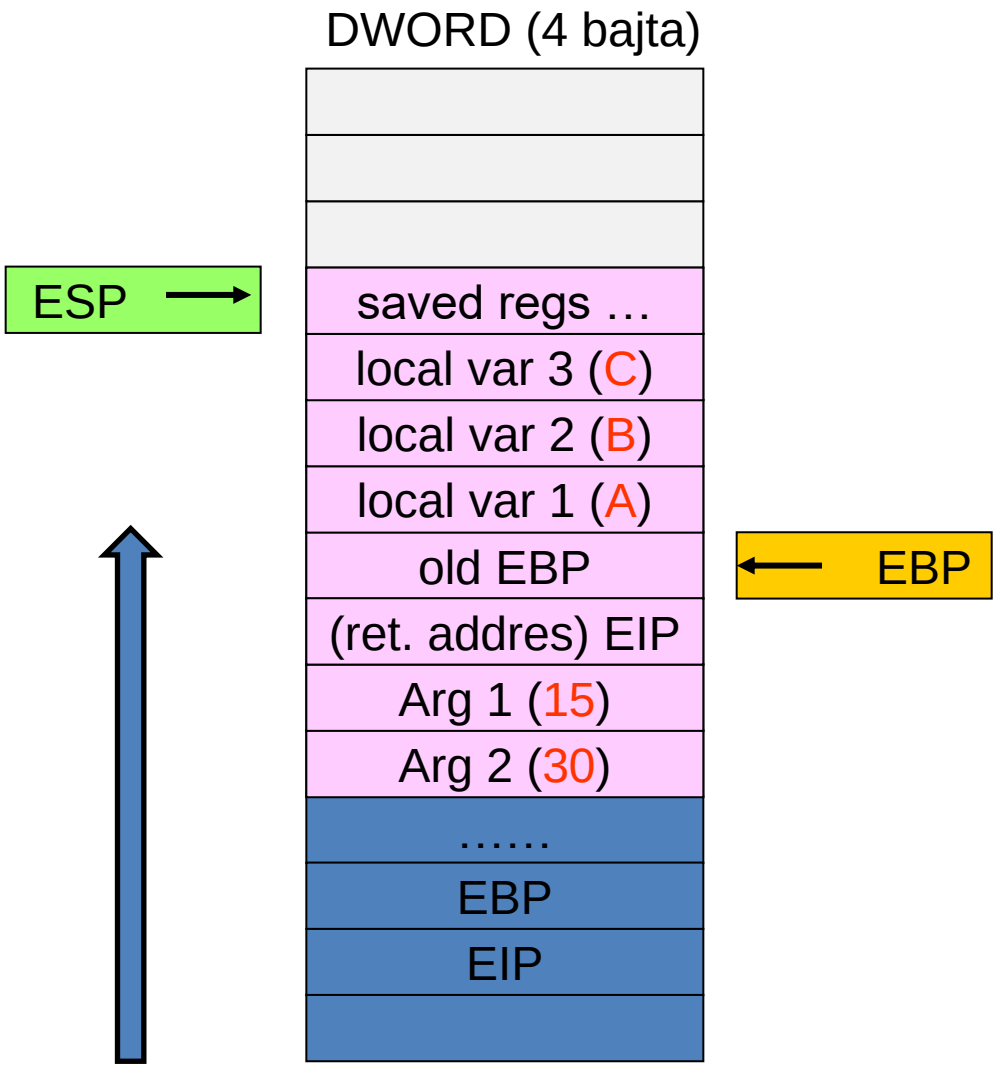
```
int main()
{
    a();
    return 0;
}
```

```
int a()
{
    b();
    c();
    return 0;
}
```

```
int b()
{ return 0; }

int c()
{ return 0; }
```



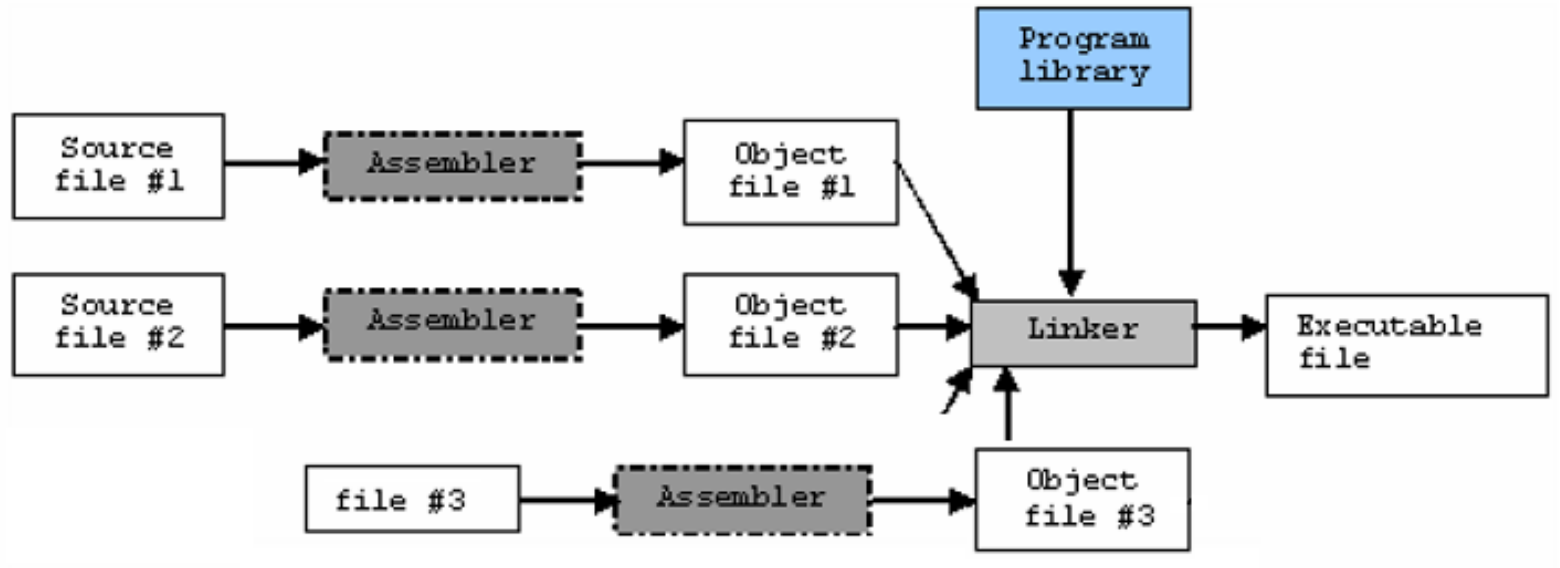
Stack frame




```
void main(void)
{
……
    ret = funkcija(15, 30);
}

int funkcija(int Arg1, int Arg2)
{
int A, B, C;
    [pristup Arg1 == EBP + 8]
    [pristup Arg2 == EBP + 12]
    [pristup loc.var B == EBP - 8]
……
    return B;
}
```

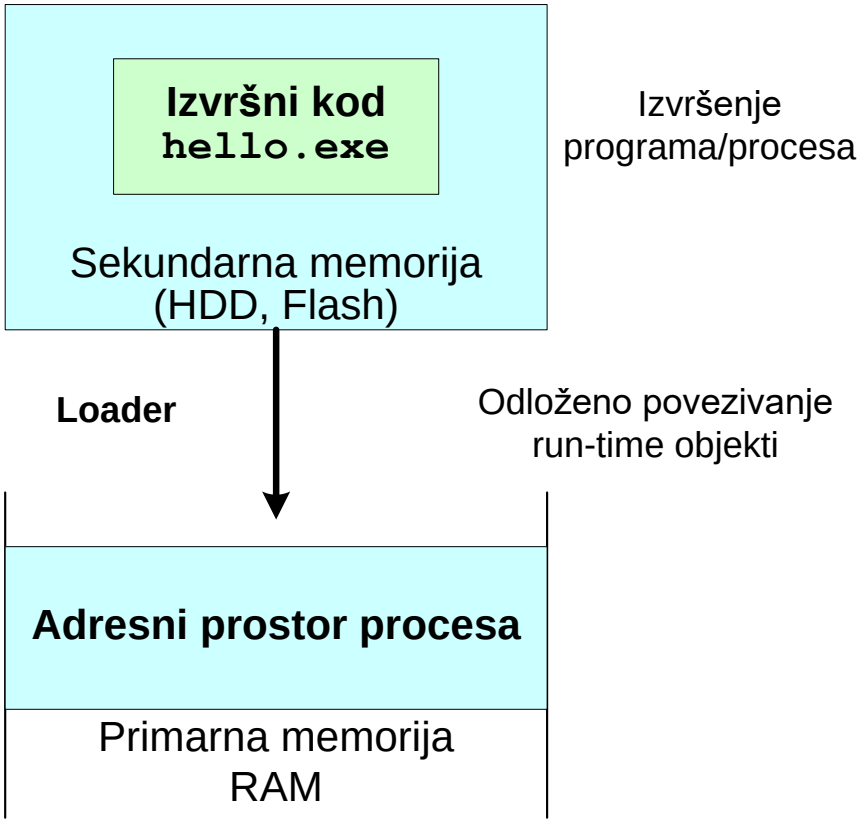
Linker

- Povezuje u izvršni kod, rešavanjem međusobnih adresnih referenci
  - Statičko – sve potrebno u izvršni fajl
  - Dinamičko – deljeni izvršni moduli (.dll)



Punjač (Loader)

- Pre izvršavanja se puni u operativnu memoriju
  - Funkcija OS
  - Sa diska
    - Sve se kopira
  - Iz Flash-a
    - Sve se kopira (max.var)
    - Samo inicijalizovane promenljive (min.var)
- Dinamičko povezivanje
  - Deffered linking
  - Run-time moduli/biblioteke



Proces punjenja

- Verifikacija izvršnog programa i računanje memorijskih zahteva
- Verifikacija raspoložive memorije i prava pristupa
- Alokacija potrebne memorije i prenos sa sekundarne memorije
- Formiranje sekcija podataka (DATA + BSS + Heap sekcija)
- Inicijalizacija stack-a i argumenata za main() funkciju
- Pozivanje main() funkcije

Direktive C Preprocesora

- Uključivanje drugih datoteka u fajl za prevođenje
  - #include <stdlib.h>, #include “MyDef.h”
  - Koristiti / (ne \) npr. #include “Common/MyDef.h”
- Definicija konstanti, tipova podataka i makroa
  - #define PI 3.14
  - #define AREA(a,b) ((a)*(b))
- Kontrola prevođenja (conditional compilation)

```
#if !defined(NULL)
        #define NULL 0
#endif

#ifdef DEBUG
        printf(“Var x= %d”, x);
#endif
```

Predefinisani makroi

Simbolička konstanta	Opis
__DATE__	Datum prevođenja
__LINE__	Broj linije u .c datoteci
__FILE__	Ime datoteke izvornog koda
__TIME__	Vreme prevođenja
__STDC__	Označava ANSI C kompatibilnost

Tipovi podataka u C

Type Name	Bytes	Other Names	Range of Values
int	4	signed	–2,147,483,648 to 2,147,483,647
unsigned int	4	unsigned	0 to 4,294,967,295
__int8	1	char	–128 to 127
unsigned __int8	1	unsigned char	0 to 255
__int16	2	short, short int, signed short int	–32,768 to 32,767
unsigned __int16	2	unsigned short, unsigned short int	0 to 65,535
__int32	4	signed, signed int, int	–2,147,483,648 to 2,147,483,647
unsigned __int32	4	unsigned, unsigned int	0 to 4,294,967,295
__int64	8	long long, signed long long	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned __int64	8	unsigned long long	0 to 18,446,744,073,709,551,615
bool	1	none	false or true
char	1	none	–128 to 127 by default, 0 to 255 when compiled by using /J
signed char	1	none	–128 to 127
unsigned char	1	none	0 to 255
short	2	short int, signed short int	–32,768 to 32,767
unsigned short	2	unsigned short int	0 to 65,535
long	4	long int, signed long int	–2,147,483,648 to 2,147,483,647
unsigned long	4	unsigned long int	0 to 4,294,967,295
long long	8	none (but equivalent to __int64)	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long	8	none (but equivalent to unsigned __int64)	0 to 18,446,744,073,709,551,615
enum	varies	none	/
float	4	none	3.4E +/- 38 (7 digits)
double	8	none	1.7E +/- 308 (15 digits)
long double	same as double	none	Same as double
wchar_t	2	__wchar_t	0 to 65,535

Promenljive

- Deo memorije koji čuva određeni tip podataka
  - tip_pod ImePromenljive;
  - sizeof(ImePromenljive) == sizeof(tip_pod)
- Tipovi promenljivih:
  - lokalne – vidljive unutar jedne funkcije
    - Smeštene na stack-u
  - globalne – definisane iznad tela funkcije
  - extern – spoljne (drugi modul)
  - static – dostupne samo funkcijama u jednom modulu
  - const – konstante (read only)
  - register – čuvaju se u registru
  - volatile – podložna promeni iz prekida

Izvedeni tipovi podataka

- Direktive: struct, union, typedef, enum
- Sve izvedene tipove treba označiti typedef-om
- Prvi član enum-a uvek mora biti inicijalizovan
- Strukture su zgodan način za grupisanje i prenos podataka između funkcija
  - sizeof(struct) == Σ_i sizeof(član $_i$), za pack(1)
  - Pristup članu: ImeStruct.član, PtrStruct->član

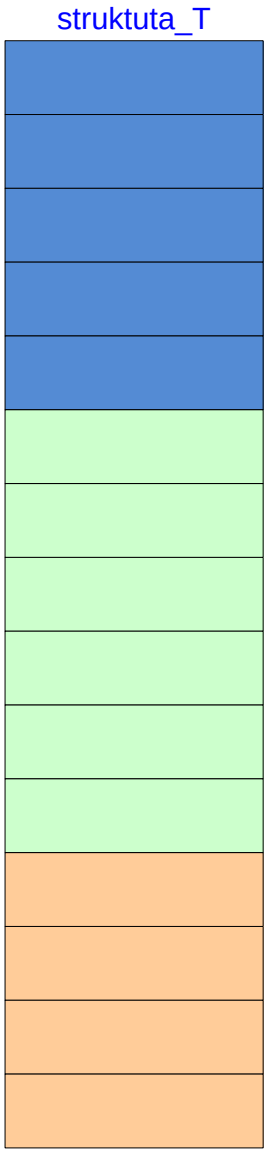
```
typedef struct {
   uint16_t      x;
   uint16_t      y;
} gdiPoint_t;

typedef enum { ILL = -1, GOOD, ...} boxStatus_t;

typedef struct listNode_t {
   struct listNode_t  *next;
   int                data;
} listNode_t;
```

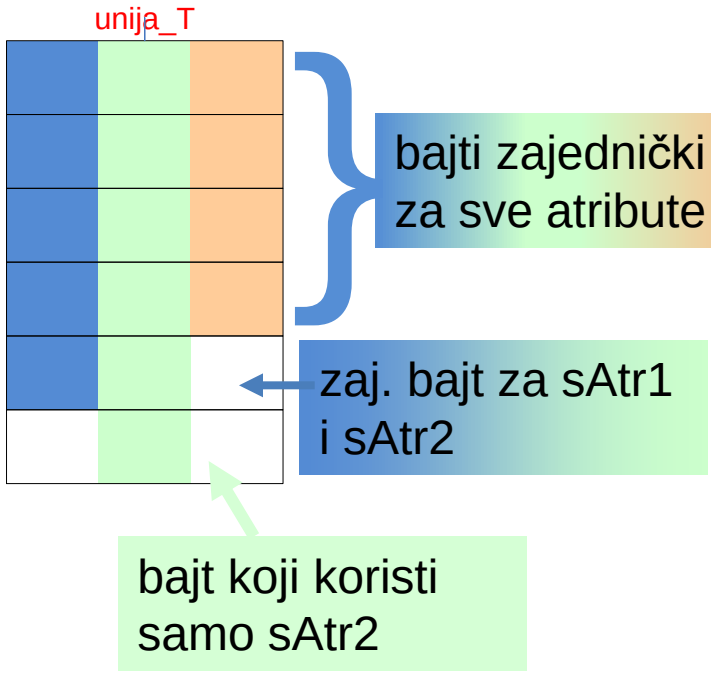
Izvedeni tipovi podataka

- union direktiva definiše tip podataka gde članovi dele isti prostor za smeštanje (storage space)
  - sizeof(union) = sizeof(najduži član)

```
union sample
{
int p;
float q;
};
…
union sample content = {234};
  ili
union sample content = {24.67};

  pristip clanovima unije

content.p = 37;
content.q = 1.2765;
```

Razlika između strukture i unije





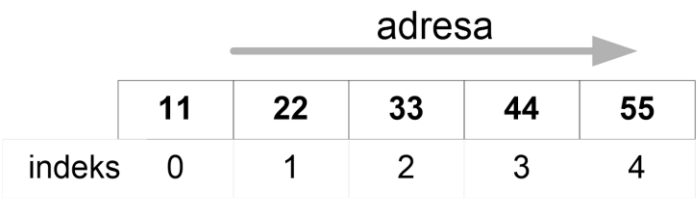

```
struct {
    char  sAtr1[5];
    short  sAtr2[3];
    long  sAtr3;
} struktura_T;
```

```
union {
    char  sAtr1[5];
    short  sAtr2[3];
    long  sAtr3;
} unija_T;
```

Nizovi

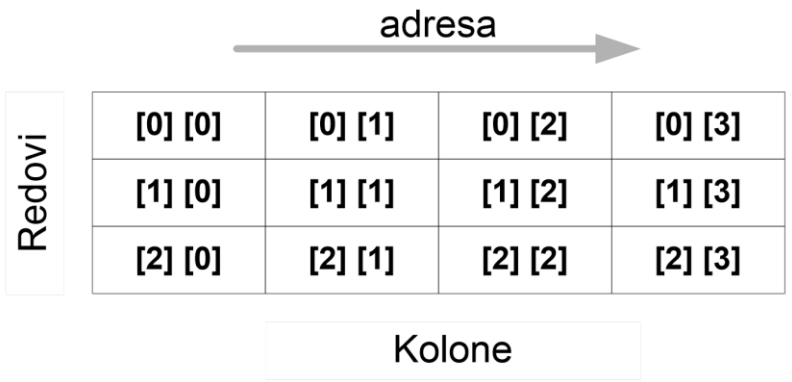
- Uređen skup podataka istog tipa, označen jednim imenom
- Podaci su smesteni kontinualno u memoriji
- Ime = adresa prvog elementa
- Pristup svakom od članova polja se ostvaruje pomoću jednog ili više indeksa, zavisno od dimenzije niza

```
int k[5] = {11,22,33,44,55};
```



k - ime označava početak niza (adresu)
k[i] - vrednost niza na odstojanju i

```
int a[3][4] =
{1,2,3,4,5,6,7,8,9,10,11,12};
...
{{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```



Pokazivači

- Pokazivač je promenljiva čija je vrednost adresa podatka smeštenog u memoriji, izražena u byte-ima
- Pokazivači mogu adresirati
  - Promenljive
  - Funkcije
- Deklaracija pokazivača
  - tip_pod *pnt;
- tip_pod deklariše format i veličinu podatka na toj adresi
- Uvećanje (++) i smanjenje (--) pokazivača menja njegovu vrednost za sizeof(tip_pod)
- Isto važi i za sabiranje i oduzimanje pokazivača sa celobrojnom vrednošću
  - [pnt ± n] = [pnt] ± n x sizeof(tip_pod)

Pokazivači na promenljive

- Postavljanje pokazivača
  - char *bp = &chr;
  - int *ip = NULL; (0L)
- * - sadržaj promenljive tip_pod na adresi iz pokazivača
  - char t = *bp;
- void * deklariše neoznačen pokazivač
  - samo adresa – odgovara unsigned char *
  - bez warning-a se može puniti pokazivač proizvoljnog tipa
- Pokazivači se mogu eksplicitno cast-ovati kao i promenljive
  - unsigned j = *(unsigned*) pnt
- Pokazivač na pokazivač
  - int **pnt; char **bp -> char *bp[];
- Niz pokazivača
  - int *pnt [20];

Nizovi i pokazivači

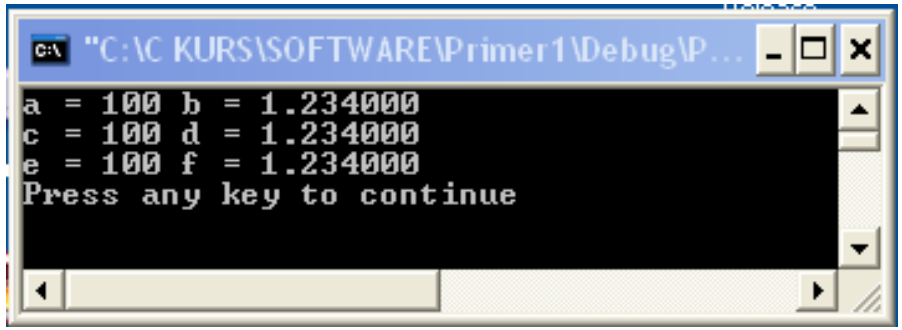
- element Niz[10];
- element *pokEl = Niz;

- Niz[3] == *(Niz + 3) == pokEl[3] == *(pokEl + 3)
  - pristup trećem (četvrtom) elementu niza Niz
- &Niz[5] == Niz + 5 == &pokEl[5] == pokEl + 5
  - adresa petog (šestog) elementa niza Niz
- (char*)Niz + 7 == (char*)pokEl + 7
  - adresa sedmog (osmog) bajta zauzetog memorijskog prostora
- *((char*)Niz + 9) == *((char*)pokEl + 9)
  - pristup devetom (desetom) bajta zauzetog memorijskog prostora

Pokazivači na funkcije

```c
#include <stdio.h>

/* functions' prototypes */
int fun1(int, double);
int fun2(int, double);
int fun3(int, double);

/* an array of a function pointers */
int (*p[3]) (int, double);

int main()
{
   int i;

   /* assigning address of functions */
   p[0] = fun1;
   p[1] = fun2;
   p[2] = fun3;

   /* calling with arguments */
   for(i = 0; i <= 2; i++)
   {
      (*p[i]) (100, 1.234);
   }
   return 0;
}
```

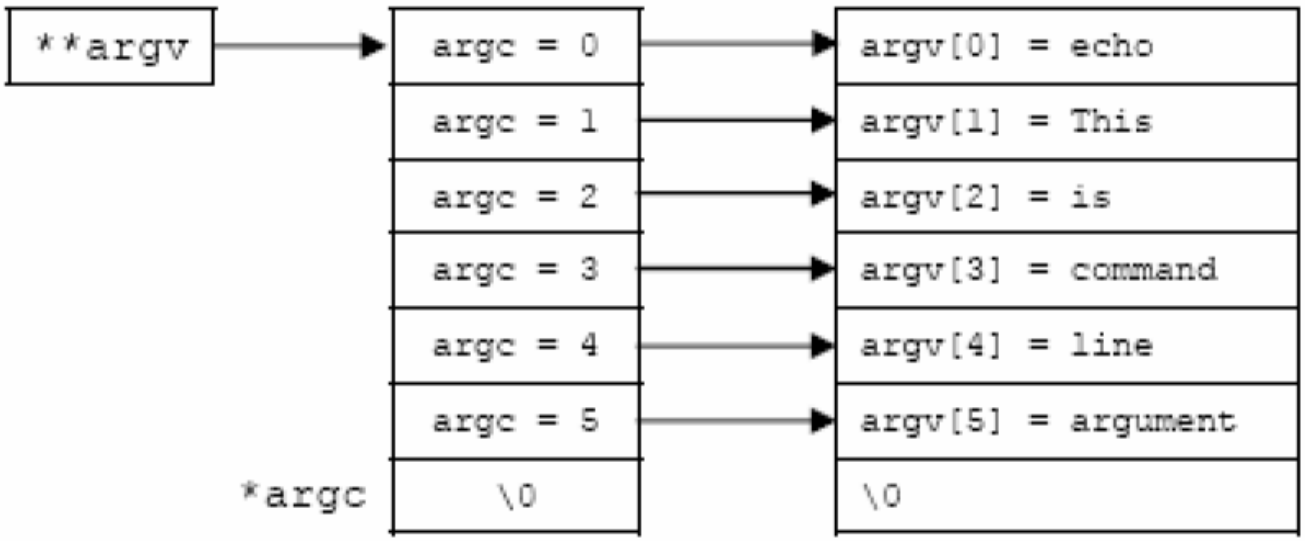
```c
/* functions' definition */
int fun1(int a, double b)
{
   printf("a = %d b = %f", a, b);
   return 0;
}

int fun2(int c, double d)
{
   printf("\nc = %d d = %f", c, d);
   return 0;
}

int fun3(int e, double f)
{
   printf("\ne = %d f = %f\n", e, f);
   return 0;
}
```



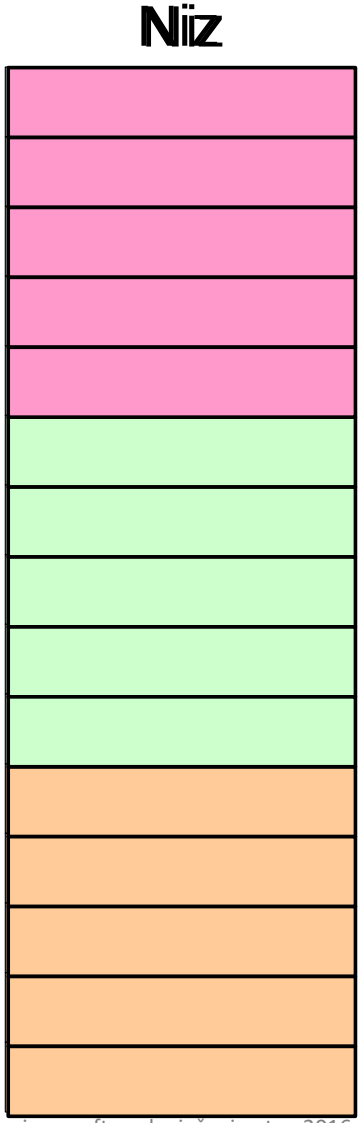
Main funkcija

- Ulazna tačka u programu
- int main(int argc, char **argv)

```
C:\>echo This is command line argument
This is command line argument
```



Višedimenzionalni nizovi



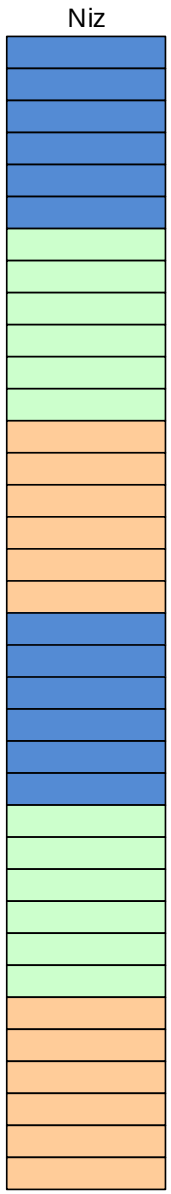

```
char Niz[3][5];
char **pokNiz  = (char**)Niz;
```

pokNiz[0] - pokazivač na char

Regularna sekvenca:

```
enum{DIM1=3, DIM2=5};
char Niz[DIM1][DIM2];

char *pokNiz  = (char*)Niz;
……..
Niz[2][3]  ==  *(pokNiz + 2 * DIM2 + 3)
```

Višedimenzionalni nizovi



long Niz[2][3][6]

```
long Niz[2][3][6];
long ***pokNiz  = (long***)Niz;
```

OVO NE VALJA

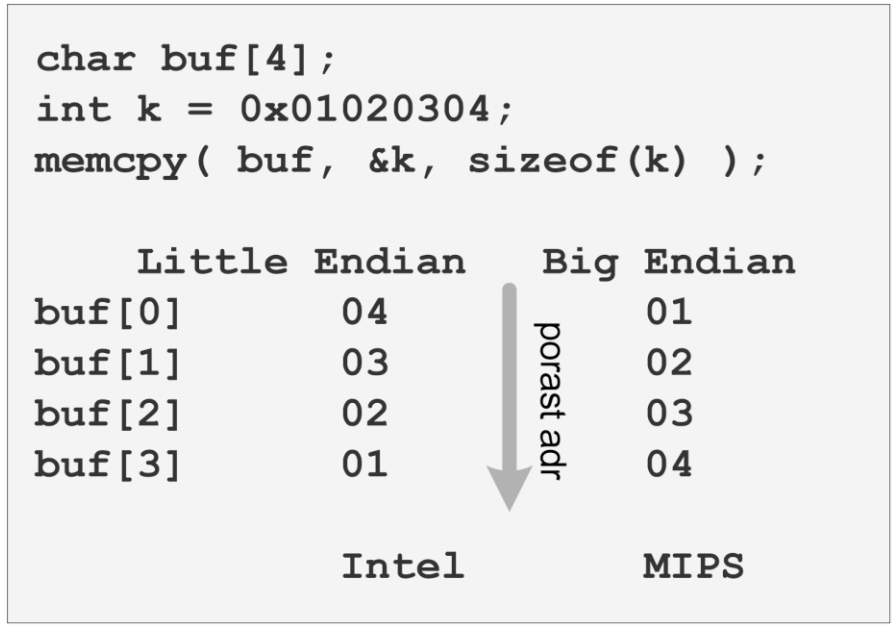
pokNiz[0] - pokazivač na pokazivač na long

```
Niz[1][2][3]  !=  pokNiz[1][2][3];
```

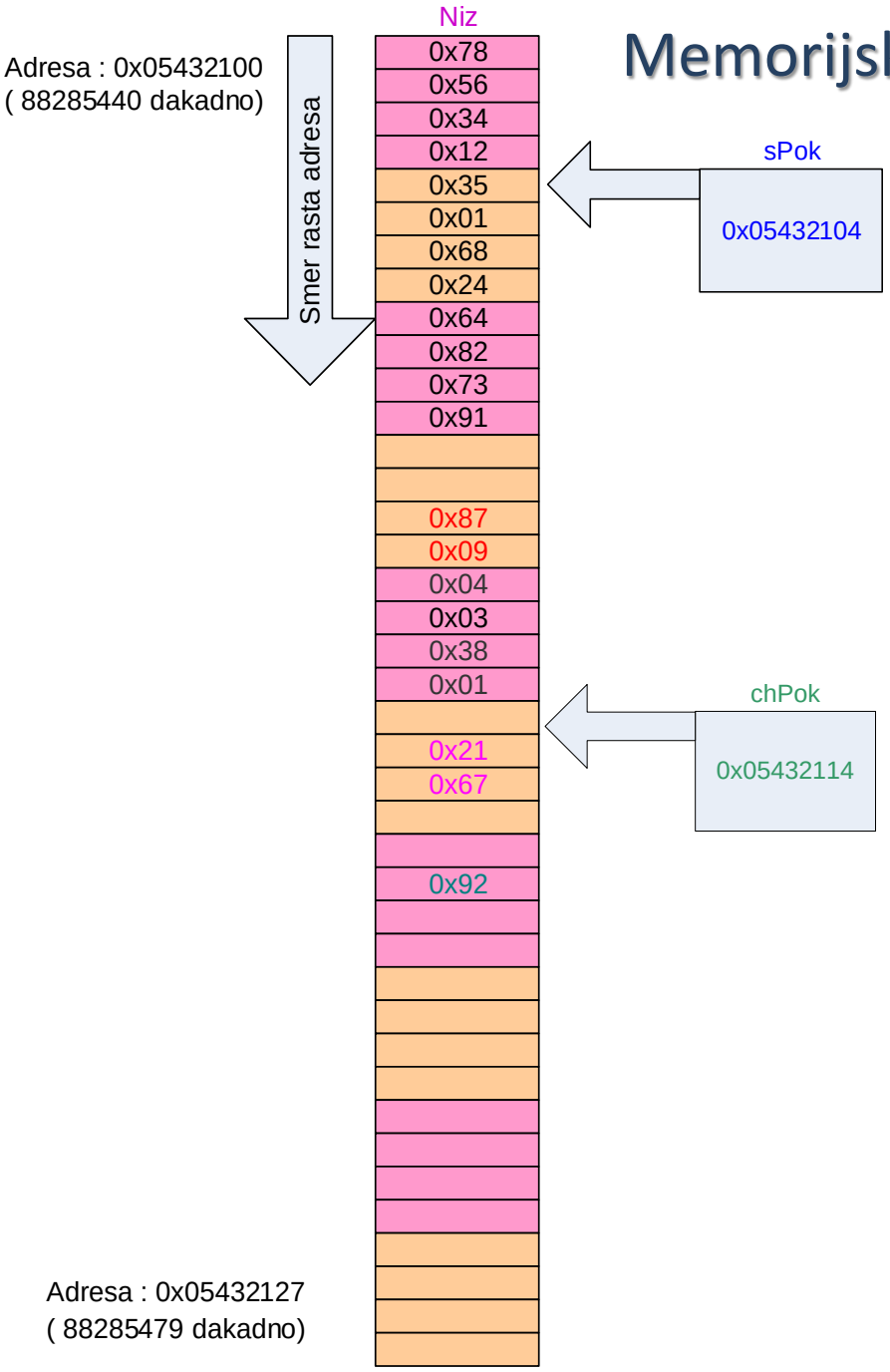
Big i Little Endian

- Redosled slaganja byte-ova u memoriji
- BE: najviši zadnji
- LE: najniži zadnji
- Bi-Endian: obe mogućnosti
- Zavisno od HW platforme
  - Intel je LE
  - MIPS je BE (ili LE – opciono)
- Problem je prisutan i kod
  - Čitanja binarnih datoteka
  - Komunikacione razmene podataka
- Ne postoji sistemsko rešenje, tj mora se programirati konverzija podataka
- Ograničava prenosivost koda

```
char buf[4];
int k = 0x01020304;
memcpy( buf, &k, sizeof(k) );
```

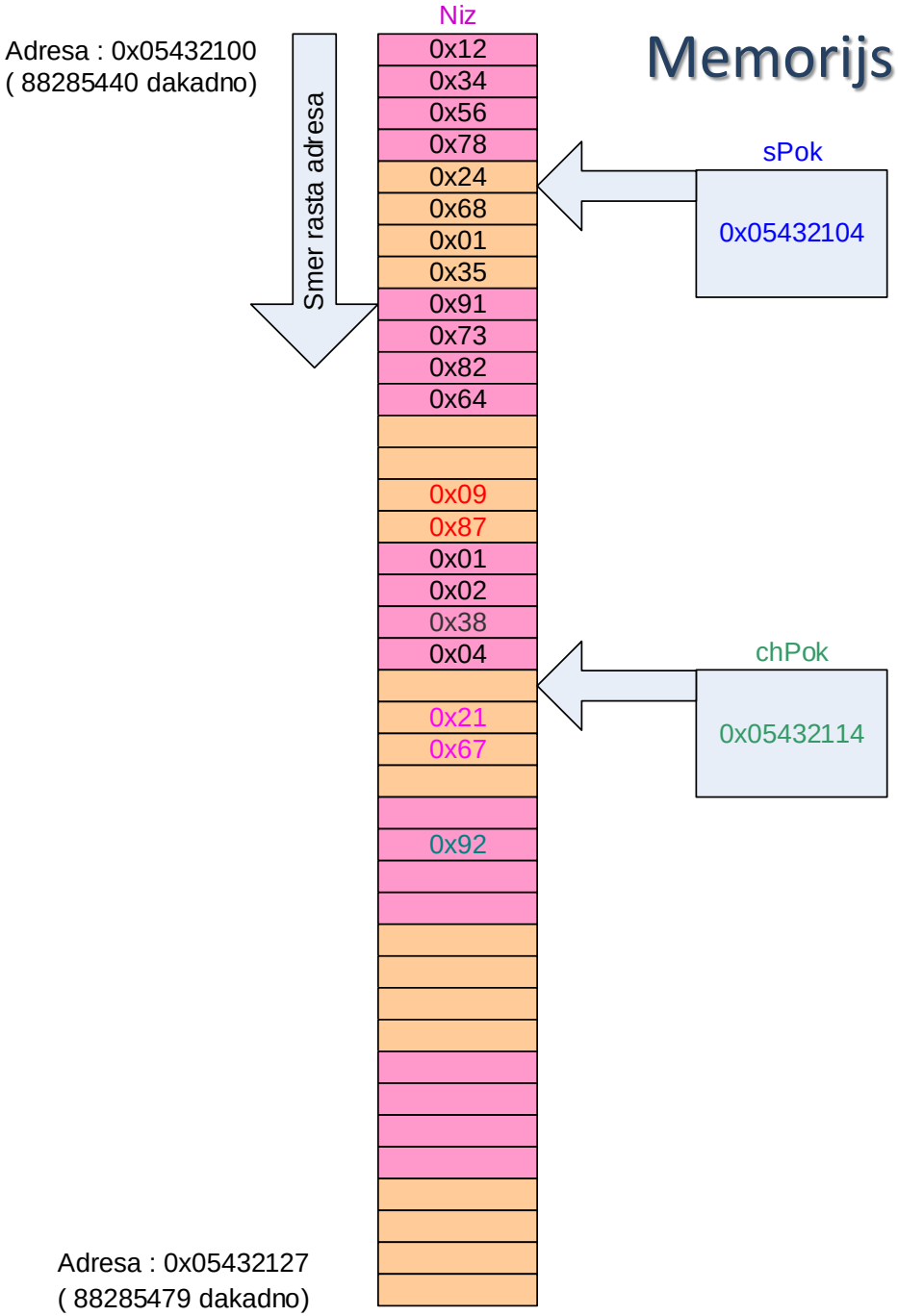


Memorijska slika – Little Endian




```
long Niz[10];
Niz[0] = 0x12345678;
Niz[1] = 0x24680135;
Niz[2] = 0x91738264;
short *sPok = (short*)&Niz[1];
sPok[5] = 0x0987;
char *chPok = (char*)(sPok + 8);
chPok[1] = 0x21;
chPok[2] = 0x67;
*(chPok+5) = 0x92;
*(Niz + 4)  = 0x01020304;
*((char*)Niz + 18)  = 0x38;
Niz[4]  ==  0x01380304
```

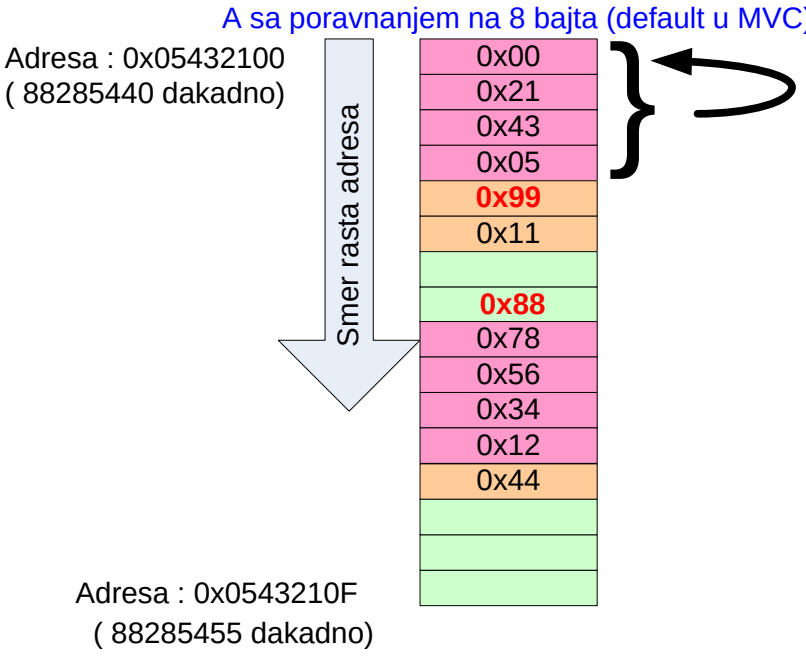
Memorijska slika – Big Endian



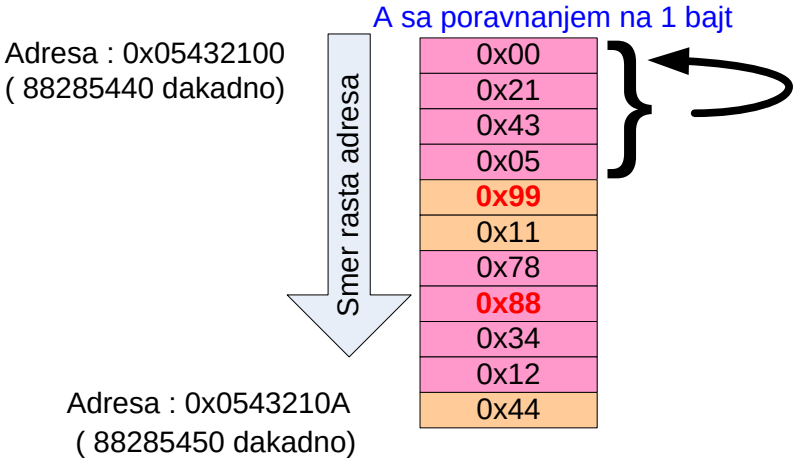

```
long Niz[10];
Niz[0] = 0x12345678;
Niz[1] = 0x24680135;
Niz[2] = 0x91738264;

short *sPok = (short*)&Niz[1];
sPok[5] = 0x0987;

char *chPok = (char*)(sPok + 8);

chPok[1] = 0x21;
chPok[2] = 0x67;
*(chPok+5) = 0x92;

*(Niz + 4)  = 0x01020304;
*((char*)Niz + 18)  = 0x38;
```

Niz[4] == 0x01023804

Poravnavanje strukture



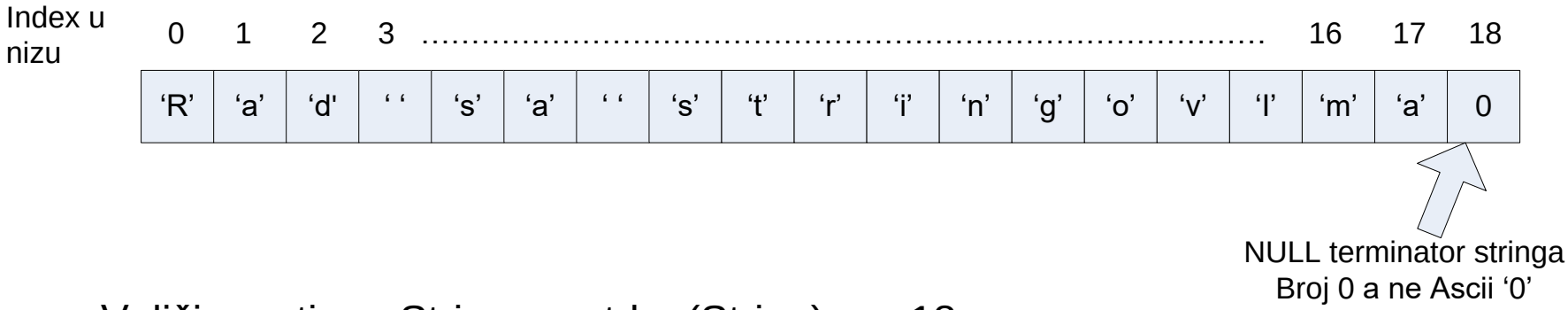

```
typedef struct
{
        unsigned char*   p1;

        unsigned short   p2;
        unsigned long    p3;
        unsigned char    p4;
} struktura_T;

struktura_T   A;
A.p1  =  (unsigned char*)&A.p1;
A.p2  =  0x1122;
A.p3  =  0x12345678;
A.p4  =  0x44;
```




```
A.p1[4]  =  0x99;
*(A.p1 + 7)  =  0x88;
```

Stringovi

char String[] = “Rad sa stringovima”;



Veličina srtinga String == strlen(String) == 18

Veličina zauzetog memorijskog prostora za String == strlen(String) + 1 == 19

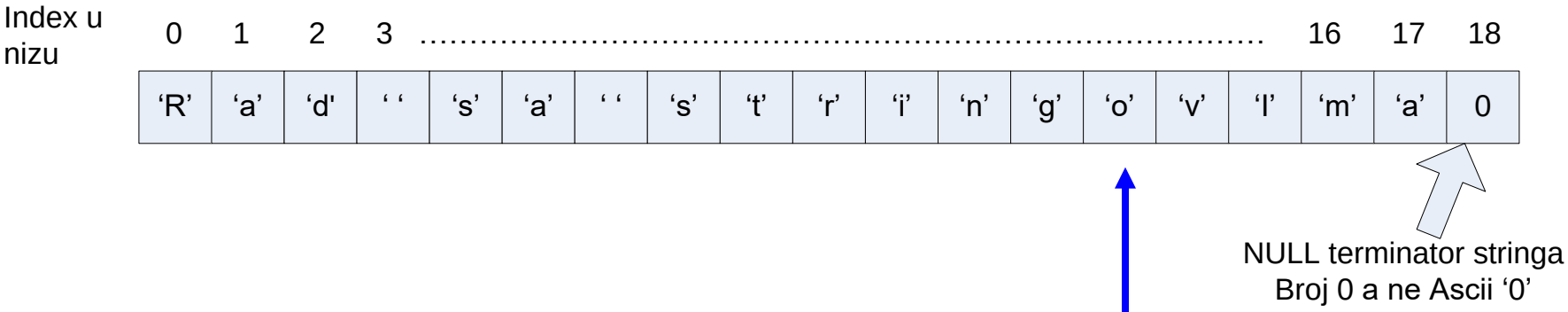
char String2[40] = “Rad sa stringovima”;

Veličina srtinga String2 == strlen(String2) == strlen(String) == 18

Veličina zauzetog memorijskog prostora za String2 == 40

Stringovi

char String[] = “Rad sa stringovima”;



char *string3 = String + 4;

string3 == “sa stringovima”

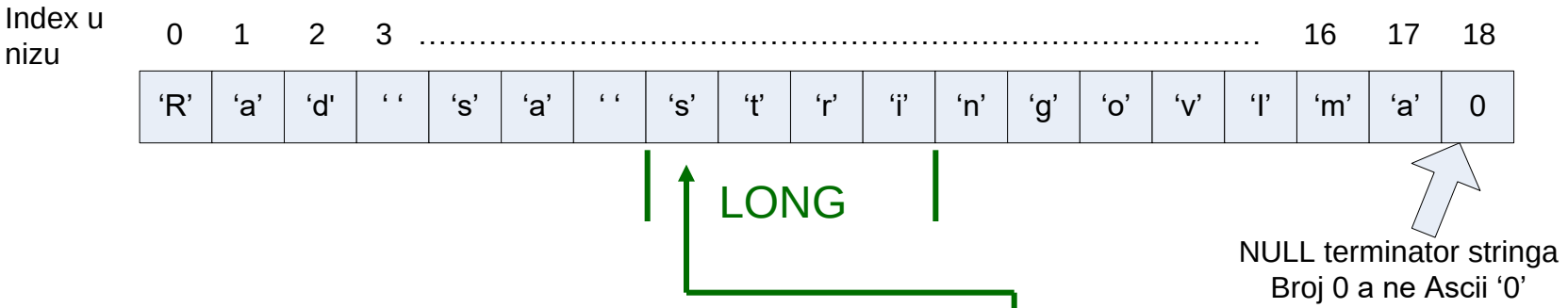
string3[9] = 0;

NULL terminator stringa

string3 == “sa string”

String == “Rad sa string”

Stringovi



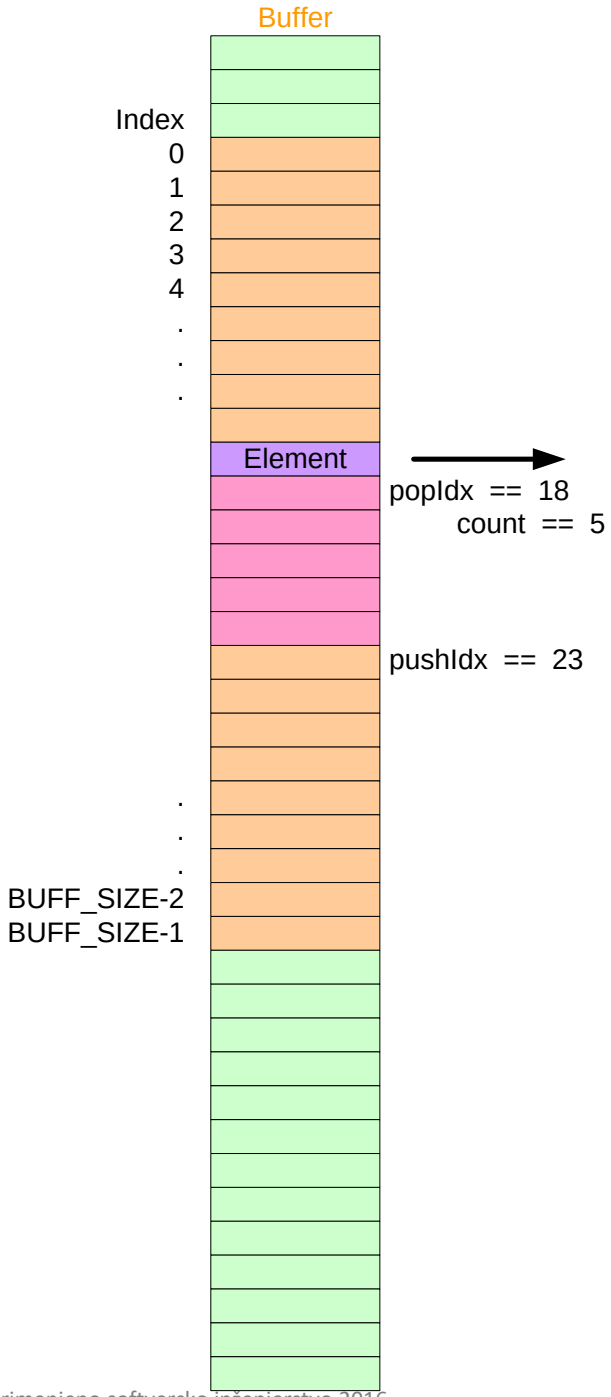
Svaki znak stringa je jedan bajt i kako ćemo mi njega videti je samo stvar intertpretacije

long *Lstr = (long *)(String + 7);

‘s’ == 0x73
‘t’ == 0x74
‘r’ == 0x72
‘i’ == 0x69

*Lstr == 0x69727473

Lstr[2] == 0x00616D69



Kružni bafer

```
element   Buffer[BUFF_SIZE];
int   pushIdx = 0;
int   popIdx = 0;
int   count = 0;

.....
```

Dodavanje elementa u kružni bafer:

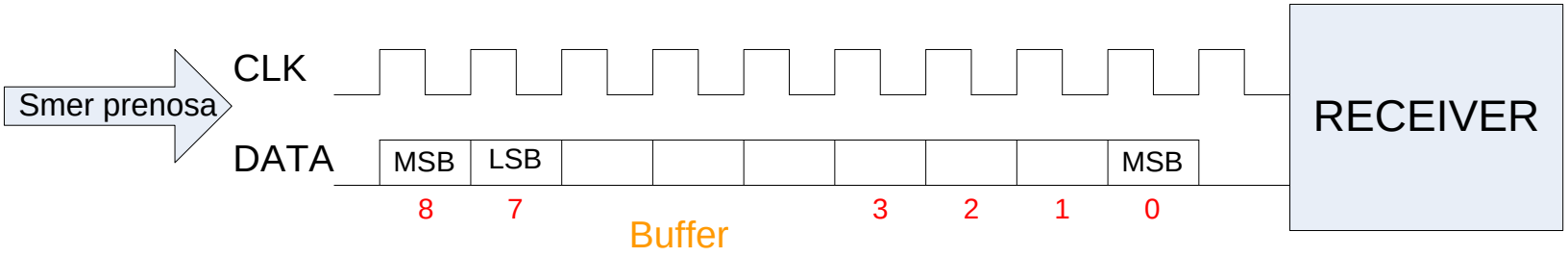
```
if(count == BUFF_SIZE)
        return false;
Buffer[pushIdx] = newEl;
pushIdx++;
if(pushIdx == BUFF_SIZE)
        pushIdx  =  0;
count++;
```

Uzimanje elementa iz kružnog bafera:

```
if(count == 0)
        return false;
Element = Buffer[popIdx];
popIdx++;
if(popIdx == BUFF_SIZE)
        popIdx  =  0;
count--;
```

Bitski bafer

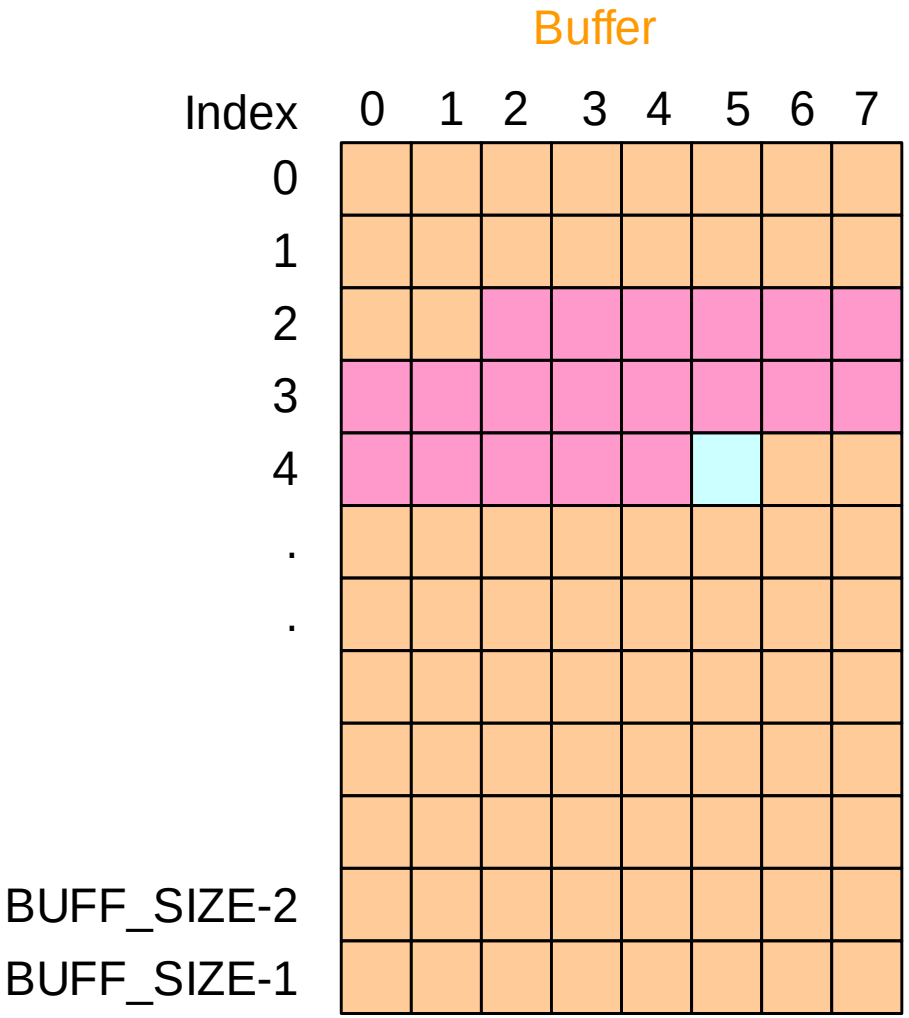
Sinhrona serijska komunikacija



Buffer

Index	7	6	5	4	3	2	1	0
0	0	1	2	3	4	5	6	7
1	8	9	.	.	.			
2	16	.	.	.				
3	24							
.	.							
.	.							
.	.							
BUFF_SIZE-2								
BUFF_SIZE-1								

Bitski bafer



popIdx == 18
count == 20

pushIdx == 38
== 0x26
== 0010 0110 b

bajtIdx == 00100 b
== 4

bitIdx == 0x110 b
== 6

```
#define BUFF_SIZE   256
#define BUFF_BIT_SIZE   2048
BYTE   Buffer[BUFF_SIZE];
int   pushIdx = 0;
int   popIdx = 0;
int   count = 0;
.....
```

odavanje bita u kružni bafer:

```
if(count == BUFF_BIT_SIZE)
        return false;
int bajtIdx = pushIdx >> 3;
int bitIdx = pushIdx & 0x00000007;
BYTE  bitMask  = 0x80 >> bitIdx;
if(Bit  !=  0)
        Buffer[bajIdx]  |=  bitMask;
else
        Buffer[bajIdx]  &=  ( ~bitMask );
pushIdx++;
if(pushIdx == BUFF_BIT_SIZE)
        pushIdx  =  0;
count++;
```