Основи програмирања 1
Лекција 9

др Зоран Бањац

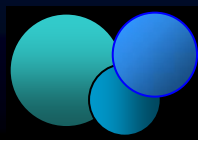
zoran.banjac@viser.edu.rs

Висока школа електротехнике и рачунарства
струковних студија
Београд

Садржај

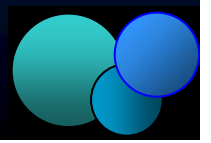
Функције

- дефиниција
- елементи
- прослеђивање аргумената



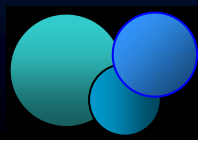
Подела програма

- Приликом развоја алгоритма могу се уочити логичке целине, односно поступци обраде које се понављају више пута, најчешће над различитим скупом података.
- Такве целине треба издвојити у посебне програмске целине - потпрограме.
- Потпрограм: део кôда унутар већег програма којим се извршава неки специфичан задатак.



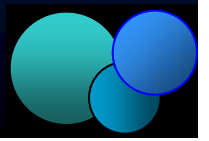
Потпрограм

- Структура потпрограма најчешће обухвта:
  - кôд који се извршава након позива потпрограма
  - вредности које се прослеђују потпрограму приликом позива
  - вредност коју потпрограм враћа на место одакле је позван
- Потпрограм може/не мора да има повратну вредност



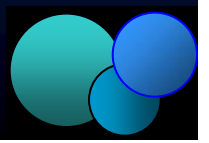
Потпрограм

- У програмским језицима обично се разликују две врсте потпрограма:
  - Процедурални потпрограми (*procedure)*, немају директну повратну вредност
  - Функцијски потпрограми (*function*), имају директну повратну вредност
- Обе врсте потпрограма могу да изазову бочне ефекте *(side effect)*
- Бочни ефекат: мењање вредности изван потпрограма независно од директне повратне вредности
  - промена глобалне променљиве, промена вредности прослеђених аргумената,…



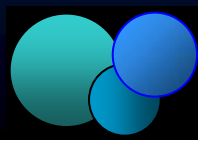
процедуре : функције

- Програмски језик C познаје само функцијске потпрограме, скраћено функције
- Синоними: потпрограми *(subprogram, subroutine*), методе (*method)* и процедуре.
- Програмски језик C дозвољава постојање функција које немају повратну вредност



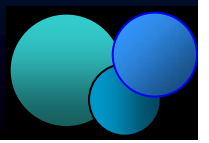
Зашто функције?

- повећава се прегледност и читљивост програма,
- смањује се величина програма,
- постиже се универзалност кода,
- омогућава се лакше отклањање грешака.
- омогућава се скривање дела кода



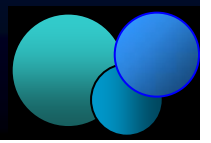
Зашто функције?

- Писање мањих целина које се лакше контролишу
- Функције - градивни елемент програма Једном написана функција се користи у више различитих програма
- Понављање извршавања истог кôда у програму



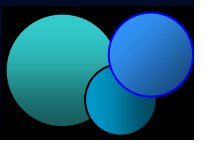
Функције

- Функција се обично дефинише као потпрограм који на основу одређеног броја улазних вредности (аргумената) даје један резултат (повратну вредност).
- Податак који функција враћа главном програму или другој функцији назива се повратна вредност функције.
- Тип функције је тип податка који враћа функција.
- Функција не мора да има повратну вредност (понаша се као процедура), односно може да даје само бочне ефекте.



Врсте функција

- ***C*** програм је најчешће комбинација два типа функција:
  - **корисничке функције:**
    функције које пише сам програмер
  - **библиотечке функције** :
    функција које постоје у стандардним библиотекама
    `printf, scanf, sin, malloc, free,` и друге.

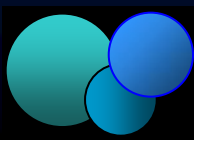


Особине Функције

- Функције у математици

$$f(x) = 2x + 5$$

- Тип функције (тип повратне вредности)
- Пареметри/Аргументи функције
- Тип параметара функције



Библиотечке функције

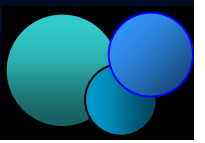
- Готове функције које постоје у оквиру програмског језика

```
#include <math.h>
#include <stdio.h>
main()
{
    double y;
   /* poziv bibliotecke funkcije*/
    y = floor(2.8);
    printf( "The floor of 2.8 is %f\n", y );
    y = floor( -2.8 );
    printf( "The floor of -2.8 is %f\n",y );
}
```

```
The floor of 2.8 is 2.000000
The floor of -2.8 is -3.000000
```

Корисничке функције

- Корисничке функције су функције које дефинише програмер
- Могу да се поделе у две категорије:
  - Функције које имају тип података тј. враћају неку вредност
  - Функције које не враћају вредност, **void** функције

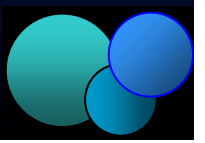


Корисничке функције

- Својства функције :
  1. Име функције
  2. Број аргумената функције
  3. Тип податка сваког аргумента
  4. Тип повратне вредности (тип функције)

---

5. Шта ради та функција, (њен кôд).



Дефиниција функције

Име функције
Идентификатор у складу са синтаксом језика

Листа формалних аргумената
Параметри кроз које функција прима податке (вредности или адресе)

```
tip   ime  (tip1 arg1, tip2 arg2, ... , tipN argN)
{
  def_lokalnih_promjenljivih;
  programski_iskazi;
  return (izraz);
}
```

Локалне промењљиве
које се користе у функцији,
Нису видљиве изван функције

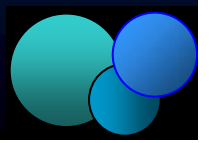
Тип функције
Ако се не наведе подразумева се `int`
Ако функција не враћа податак, тип је `void`

Излаз из функције и враћа вредност датог типа
У телу функције може бити више return исказа
Ако функција не враћа ништа, return може да се изостави.

Позив функције

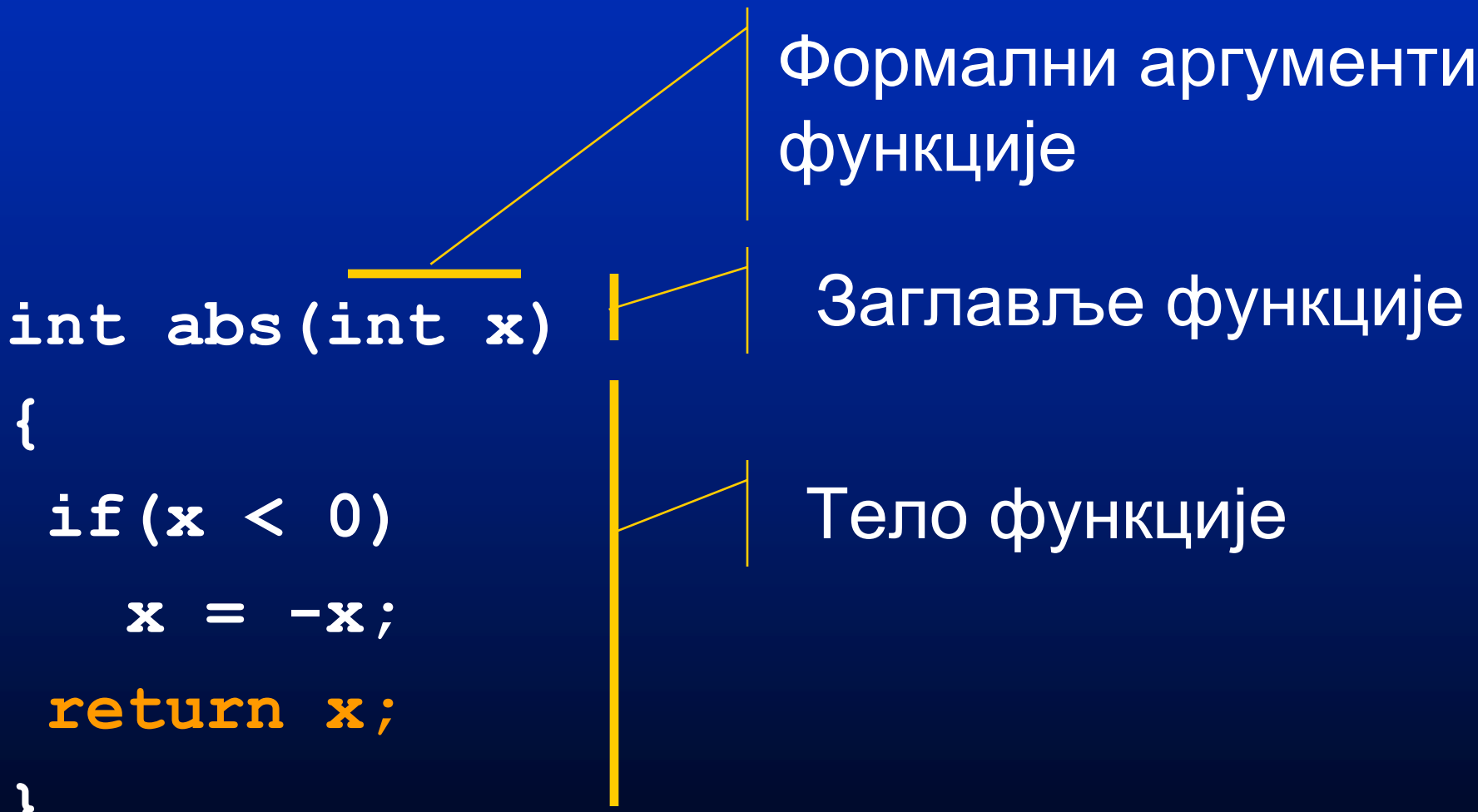
```
tip prom;
...
prom = ime(sarg1, sarg2, ... , sargN);
...
```

- Приликом позива наводе се вредности аргумената **(sarg1, sarg2, ... , sargN)** или имена променљивих (типови се не наводе) чије вредности треба да се пренесу у функцију
- Ови аргументи се називају СТВАРНИ аргументи
- Ако функција враћа вредност та вредност може да се додели променљивој (**prom**)
- Препорука је да тип те променљиве буде једнак типу повратне вредности



Функције које враћају вредност

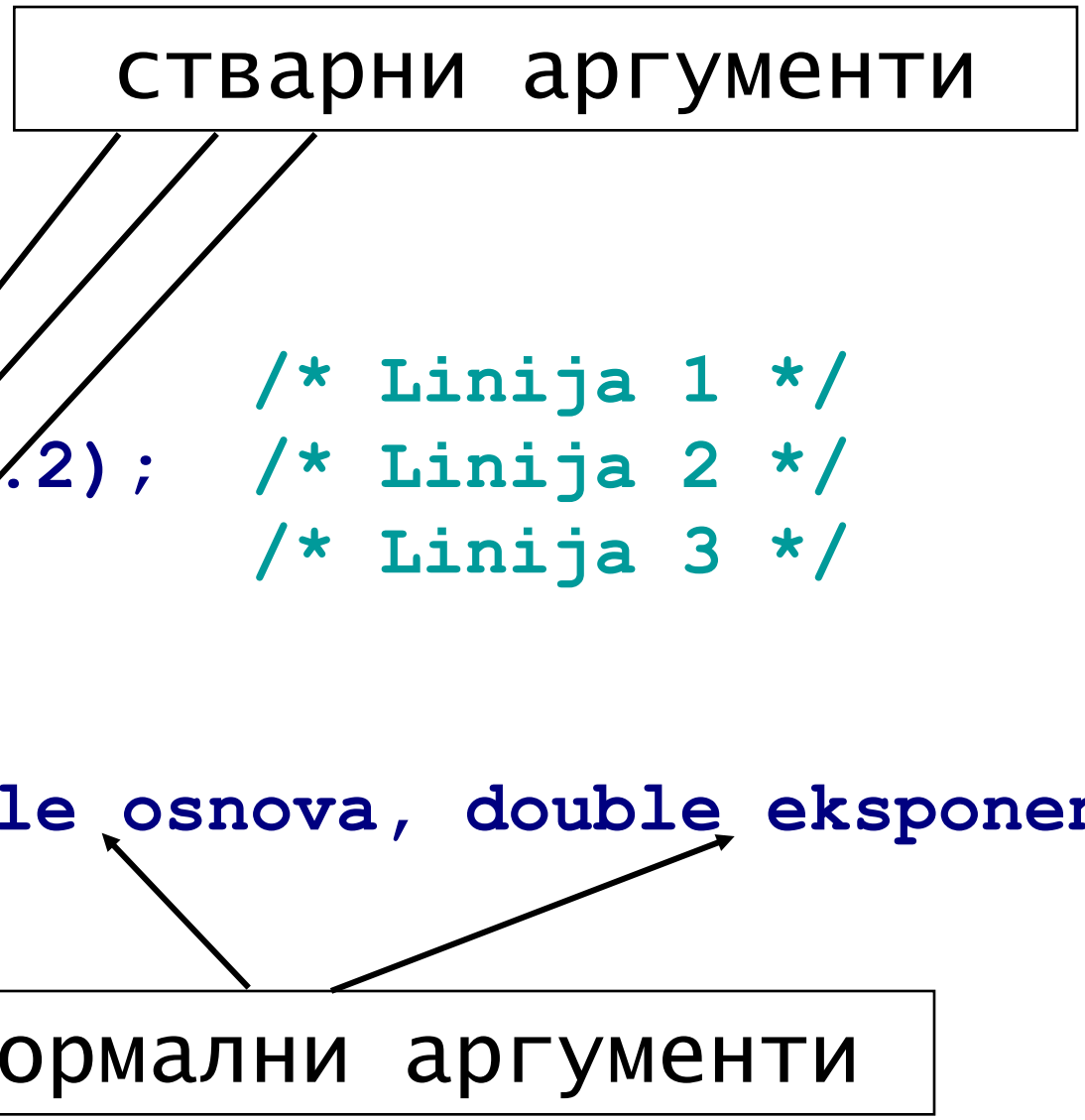
- Заглавље и тело функције




```
int abs(int x)
{
 if(x < 0)
    x = -x;
 return x;
}
```

Формални и стварни аргументи

```
main()
{
 double u = 2.5;
 double v = 3.0;
 double x, y, w;
 x = stepen(u, v);          /* Linija 1 */
 y = stepen(2.0, 3.2);      /* Linija 2 */
 z = stepen(v, 7);          /* Linija 3 */
}

double stepen(double osnova, double eksponent)
{
   ...
}
```



Дефинисање функција

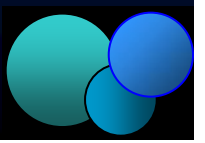
- Општа синтакса за дефинисање функције је:

```
tip_funkcije ime_funkcije(lista form argumenata)
{
 naredbe
}
```

- Тело функције
  - Део синтаксе између витичастих заграда
  - Наредбе:

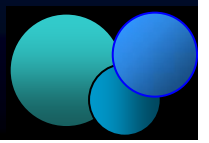
    дефинисање променљивих (декларативне)

    извршне наредбе



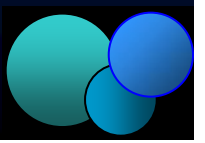
Тип функције

- Тип вредности коју враћа функција.
- Назива се и тип повратне вредности (*return type*) функције.
- Уколико се изостави подразумева се тип **`int`**.
- За функције које не враћају вредност резервисан је тип **`void`**.



Тип функције

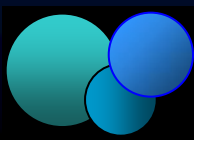
```
int ime_funkcije(lista form. argumenata)

double ime_funkcije(lista form. argumenata)

void ime_funkcije(lista form. argumenata)
```

Име функције

- Правила за доделу имена идентификатора.
- Испред имена функције може да се стави *
  Повратна вредност функције: показивач на податке чији је тип наведен као tip_funkcije

```
double *treci_stepen(lista form. argumenata)

void *moja_fun(lista form. argumenata)
```

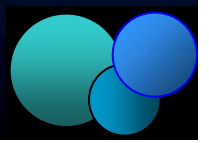



Листа формалних аргумената

- Наводи се унутар малих заграда
- Потребно је навести тип података и његово име.
- Уколико постоји више аргумената они се међусобно раздвајају зарезом.

```
tip_funkcije ime_funkcije(tip_prom1
  ime_prom1, tip_prom2 ime_prom2,...
                    ,tip_promN ime_promN)
```

```
float funPrimer(int a, double b)
```

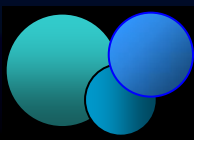
Листа формалних аргумената

- Аргумената не мора бити али и у том случају се морају навести мале заграде.

```
/* definicija funkcije */
tip_funkcije ime_funkcije() /* zaglavlje */
{
   ...
}
```

- У случају да је листа формалних параметара празна, онда приликом позива те функције не треба наводити листу стварних аргумената.

```
ime_funkcije(); /* poziv */
```

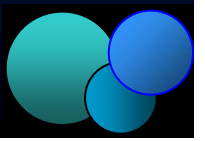
Наредба return

- Вредност коју функција треба да врати се враћа помоћу наредбе **`return`**.

- Синтакса

```
return izraz;
```

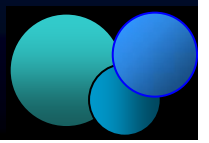
- **`izraz`**
  - променљива,
  - константа или
  - неки израз чијим израчунавањем се добија жељена вредност.



Наредба return

- Након извршења наредбе **`return`** функција се тренутно напушта, а контролу преузима део програма из ког је функција позвана.

- Наредба којом је позвана функција се „замењује” са вредношћу коју враћа функција, тј. са вредношћу поред наредбе **`return`**.



Наредба return

```
float manji(float x,float y)
{
 float pom;
 if(x <= y)
      pom = x;
 else
     pom = y;
 return pom;
}
```

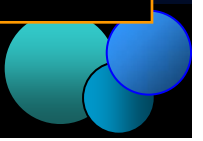
1

```
float manji(float x, float y)
{
  if(x <= y)
       return x;
   else
       return y;
 }
```

2

```
float manji(float x, float y)
{
  if(x <= y)
      return x;
   return y;
 }
```

3



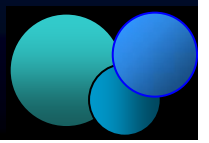
Наредба return

```
#include<stdio.h>
main( )
{
 float prvi, drugi, MinBroj;

 printf("Manji broj od 5 i 6 je %f\n",
                                  manji(5,6));

 printf("Unesite dva broja\n");
 scanf("%f %f", &prvi ,&drugi);
 MinBroj = manji(prvi, drugi);
 ...
```

Наредба return

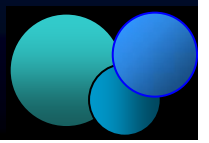
```
 printf("Manji od %f i %f je %f\n", prvi,
                                 drugi, MinBroj);
 printf("Manji od %f i %f je %f\n", 0.9,
                              drugi, manji(0.9, drugi));
}
```

```
Manji od 5 i 6 je 5.000000
Unesite dva broja
8 2
Manji od 8.000000 i 2.000000 je 2.000000
Manji od 0.900000 i 2.000000 je 0.900000
```

Прототип функције

- Идентификатор мора да се декларише пре прве употребе.
- Обично се прво пише функција **`main`** пре свих осталих корисничких функција.
- Проблем: како позвати функцију пре дефиниције
- Решење: **Прототип функције** пре дефинисања било које функције (укључујући и функцију **`main`**)



Прототип функције

- **Прототип функције** представља заглавље функције (без навођења тела функције).
- Прототип функције се обавезно завршава са знаком тачка-зарез (;)

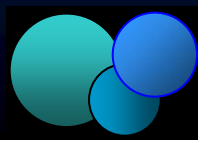
```
tip_fun  ime_fun(lista formalnih arg.);
```

- прототип за функцију `manji` је

```
float manji(float x, float y);
```

- Имена променљивих нису неопходна, тип јесте

```
float manji(float , float );
```

Прототип функције

- Прототип или декларација функције представља њен ”опис за спољашњи свет”
- Из прототипа се види како се комуницира са датом функцијом, јер садржи:
  - тип функције,
  - име функције,
  - типове (а може и имена) аргумената.

```
int zbir( int a, int b )
{  return (a+b);  }
```

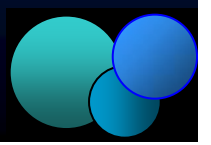
→

```
int zbir( int , int );
```

```
void poruka( )
{  printf(“zdravo!”);  }
```

→

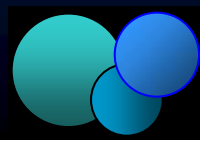
```
void poruka( );
```



Функције

Структура C програма

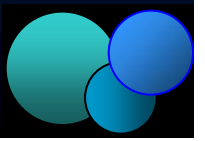
- Уобичајено
  - претпроцесорске директиве
  - прототип функција
  - main
  - дефиниција функција
- Могуће
  - претпроцесорске директиве
  - дефиниција функција
  - main
- Погодно када постоји мали број функција
- Ако функција позива неку другу функцију, функција која се позива мора претходно бити дефинисана



Пример

```
#include <stdio.h>
void poruka ();
main()
{
    poruka();
}

void poruka ()
{
  printf("Zdravo!");
}
```

```
#include <stdio.h>

void poruka ()
{
  printf("Zdravo!");
}

main()
{
    poruka();
}
```

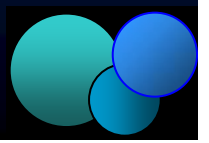



void функције

- Функције које не враћају вредност

- Наредба **`return`** не мора да постоји
- **`return`** може али без израза поред ње
- **`return`** се користи се за напуштање функције пре њеног краја.

- **`void`** функције могу али не морају да имају аргументе

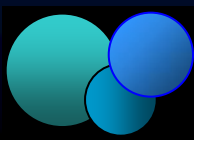


Прослеђивање вредности аргумената

- Приликом позива функције у малим заградама се наводи листа аргумената.
- По начину преношења вредности, уопштено гледано, постоји две врсте аргумената :

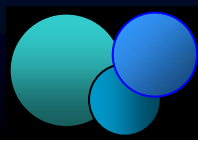
1. Аргументи који се прослеђују преко вредности (***value arguments***)

2. Аргументи који се преносе помоћу адресе (***reference arguments***)



Пример 1:

Учитати дужине страница правоугаоника, а затим израчунати и исписати обим и површину. Обим треба да се рачуна у функцији obim, а површина у функцији povrs.



Пример 1: 1/2

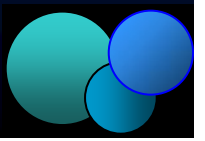
```
#include <stdio.h>
float obim(float, float);
float povrs(float s1, float s2);


main()
{
   float a, b, o, p;
   printf("Stranica a: ");
   scanf("%f", &a);
   printf("\nStranica b: ");
   scanf("%f", &b);
```

Прототип функције `obim`

Наведени само типови аргумената

Прототип функције `povrs`

Наведени типови и имена аргумената



Пример 1: 2/2

```
o = obim(a,b);
p = povrs(a,b);
```

Позив функције `obim`
Позив функције `povrs`

Шаљу се стварни подаци `a` и `b`

```
 printf("Obim: %7.3f\n", o);
 printf("Povrsina: %7.3f\n",p);
}   /* kraj funkcije main */
```

```
float obim(float x, float y)
{  return ( 2*x + 2*y ); }
```

Дефиниција функције obim

Функција прихвата прослеђене вредности (`a` и `b`) кроз формалне аргументе `x` и `y`

Функција израчунава обим `2*x+2*y` и то враћа у главни програм

```
float povrs(float c, float d)
{ float s;
  s = c*d;
  return ( s );
}
```

Пример 2:

```
#include <stdio.h>
void kvadrat ( int );
main()
{
    int i,n;
    do{
        printf("n = "); scanf("%d", &n);
    }while (n<1);
    for (i = 1; i <= n; i++)
       kvadrat(i);
} /* kraj funkcije main */
void kvadrat (int k)
{
    int kv;
    kv = k * k;
    printf("%5d %5d\n", k, kv);
}
```

```
n = 4
....1 ....1
....2 ....4
....3 ....9
....4 ..16
```

Позив функције `kvadrat`

Функција `kvadrat` не враћа вредност (`void`)

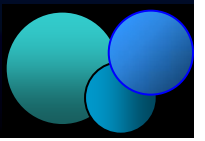
Оваква функција понаша се као процедурални потпрограм

Пример 3: 1/2 Испис *n* простих бројева

```
#include <stdio.h>
#include <math.h>
int prost(int); /* deklaracija funkcije */
main()
{
  int i = 1, broj = 1, n;
  do{
      printf("n = "); scanf("%d", &n);
  }while (n < 1);
  printf("Prosti brojevi:\n");
  while (i <= n)
  {
    if ( prost(broj) ) /* poziv funkcije prost */
    {  printf(" %d", broj); i++; }
    broj++;
  }
} /* kraj funkcije main */
```

Пример 3: 2/2

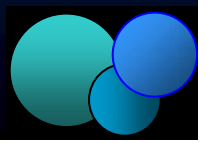
```
/* definicija funkcije */
int prost(int b)
{
   int d = 3, pom;
   pom = (b % 2) || (b == 2);
   while (pom && d<=sqrt(b))
   {
      pom = b % 2;
      d += 2;
   }
   return(pom);
}
```

Пренос параметра у функцију преко ВРЕДНОСТИ

1. приликом позива функције шаље се вредност
   - као стварни аргумент може да се наведе:
     - име променљиве,
     - нека константа или
     - израз (израчунава се вредност израза и та вредност се шаље у функцију)

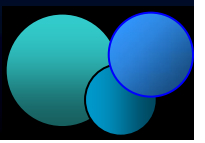
2. функција прихвата послате вредности у формалне аргументе
   - формални и стварни аргументи морају да се слажу у:
     - броју,
     - редоследу и
     - типу



Пренос параметра у функцију преко ВРЕДНОСТИ

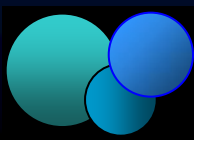
- формални аргументи (променљиве) аутоматски настају приликом уласка у функцију и преузимају прослеђене вредности

3. по изласку из функције формални аргументи аутоматски нестају
   - аутоматски настају и нестају па се називају аутоматске променљиве



Пренос параметра у функцију преко ВРЕДНОСТИ

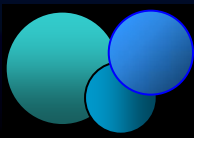
- У функцији се формира слика стварних аргумената
- Користи се та слика, а не стварне променљиве из главног програма
- Зато након изласка из функције променљиве у главном програму остају непромењене!!!



Пример 4:

```
#include <stdio.h>
void f(int);/* deklaracija funk. */
main()
{
   int i = 10;
   printf("main: %d\n",i);
   f(i);
   printf("main: %d\n",i);
}
/* definicija funkcije */
void f(int k)
{
   printf("f: %d\n",k);
   k = 20;
   printf("f: %d\n",k);
}
```

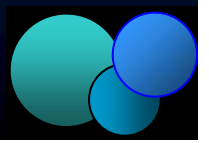
```
main:10
f:10
f:20
main:10
```



Пренос параметра у функцију преко АДРЕСЕ

Приликом позива функције шаље се адреса

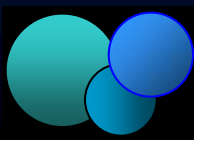
- као стварни аргумент може да се наведе:
  - име показивача,
  - нека константа (вредност се тумачи као адреса)
  - израз
- Познавање адресе омогућава промену ведности стварног аргумента!



Пример

```
#include <stdio.h>
void fa(int *p);/* deklaracija funk.*/

main()
{
   int i = 10;
   printf("main: %d\n",i);
   fa( &i);
   printf("main: %d\n",i);
}
/* definicija funkcije */
void fa(int *k)
{
   printf("fa: %d\n", *k);
   *k = 20;
   printf("fa: %d\n",*k);
}
```

```
main:10
fa:10
fa:20
main:20
```

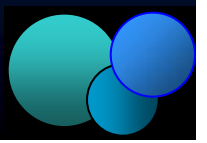



Пример 1/29

```
#include<stdio.h>
void funJedan(int a, int *b, char v);
int funDva(int *x, int y, char *w);
main( )
{
    int num1, num2;
    char ch;

    num1 = 10;          /* Linija 1*/
    num2 = 15;          /* Linija 2*/
    ch = 'A';           /* Linija 3*/
```

Пример 2/29

```
    ...
      printf("Linija  4: .... ");
      funJedan(num1, &num2, ch);
      printf("Linija  6: ....");
      num1 = funDva(&num2, 25, &ch);
      printf("Linija  8: ... ");
}
```

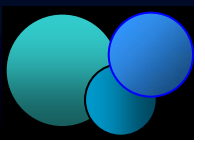



Пример 3/29

```
void funJedan(int a, int *b, char v)
{
 int jedan;
 jedan = a;          /* Linija  9 */
 a++;                /* Linija 10 */
 *b = (*b) * 2;      /* Linija 11 */
 v = 'B';            /* Linija 12 */
 printf("Linija 13: Unutar funJedan
  a=%d, *b=%d, v=%c\n",a,*b,v);
}
```

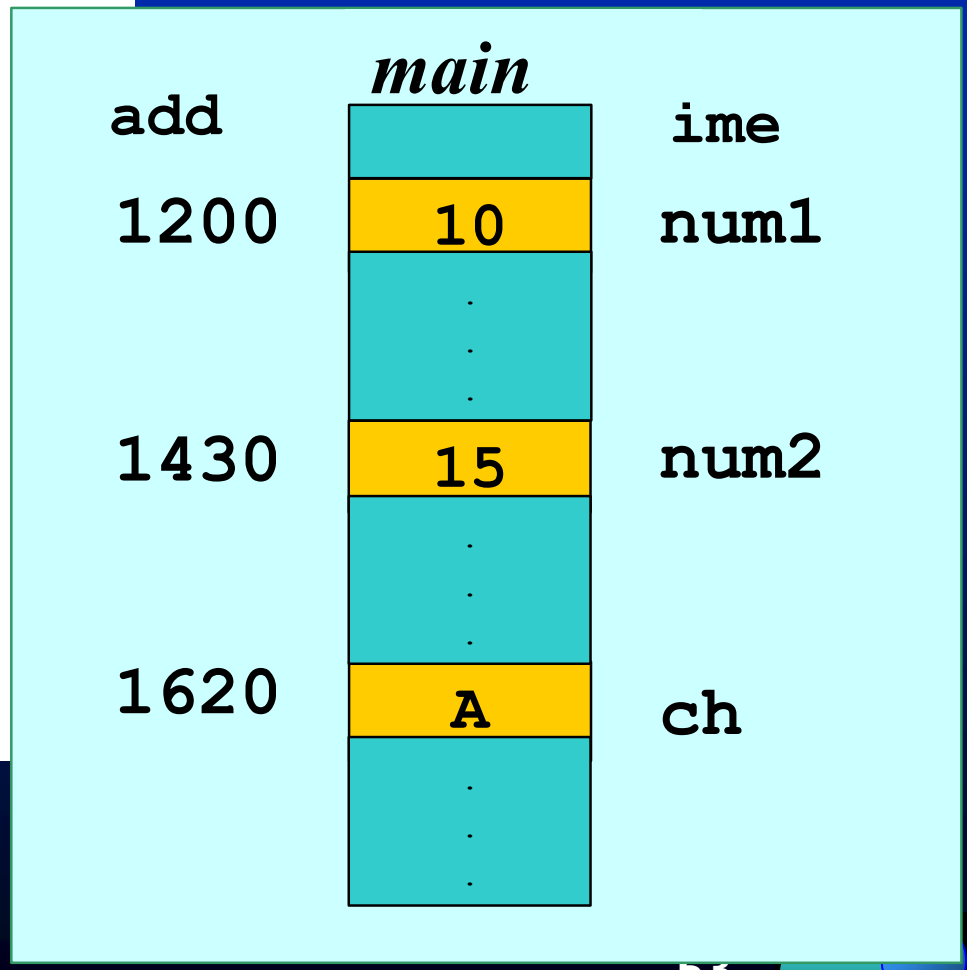
Пример 4/29

```
int funDva(int *x, int y, char *w)
{
   (*x)++;          /* Linija 14*/
   y = y * 2;       /* Linija 15*/
   *w = 'G';        /* Linija 16*/

   return y;
}
```

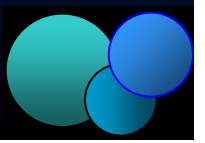
Пример 5/29

```
main( )
{
 int  num1, num2;
 char ch;

 num1 = 10; /*L1*/

 num2 = 15; /*L2*/

 ch = 'A';  /*L3*/
```



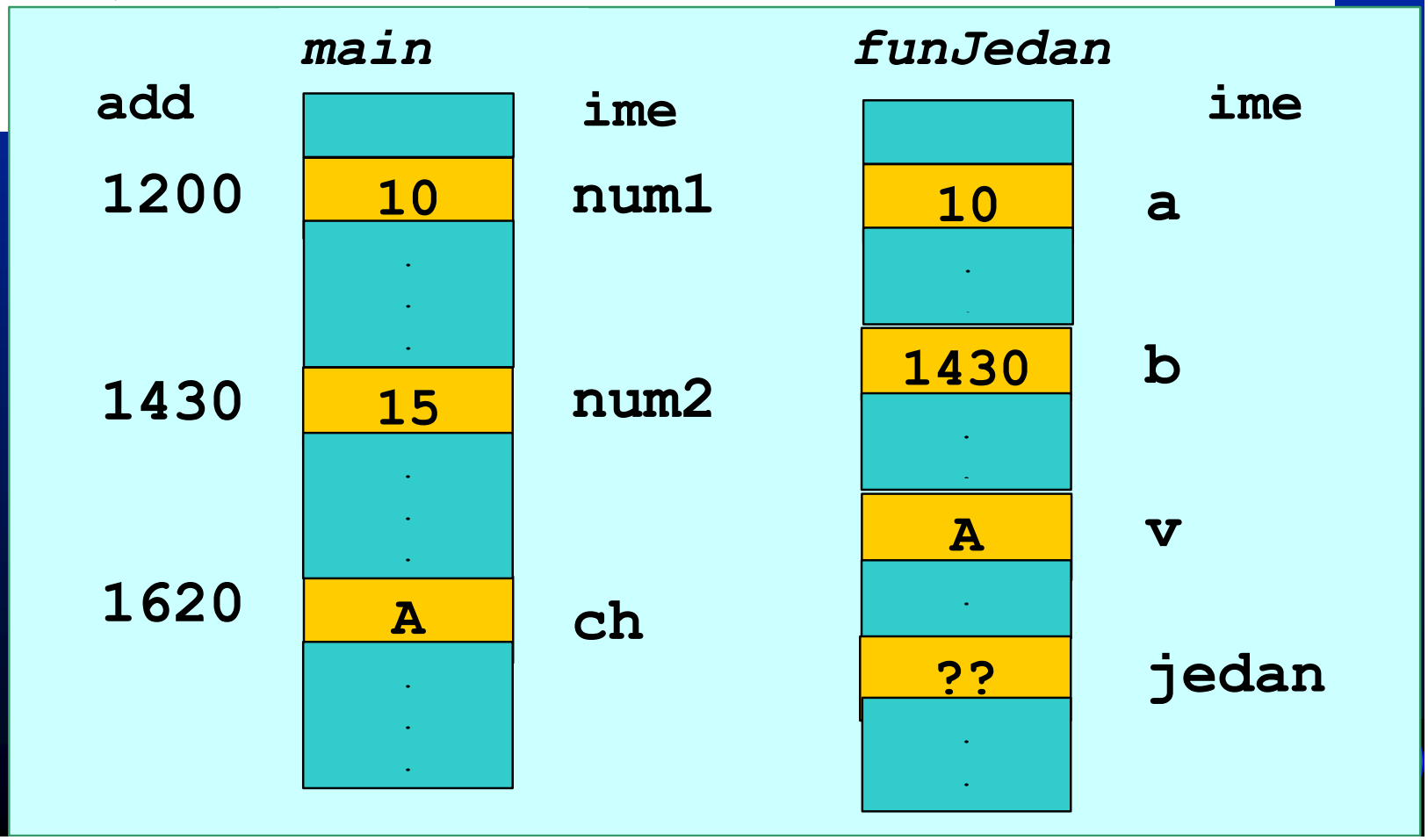
Пример 6/29

```
main( )
{
 int num1, num2;
 char ch;

 num1 = 10;                          /* L1*/
 num2 = 15;                          /* L2*/
 ch = 'A';                           /* L3*/
 printf("Linija  4: ....);
 funJedan(num1, &num2, ch);
```

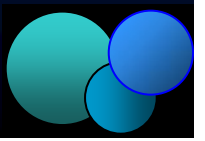
Пример 7/29

```
funJedan(num1, &num2, ch); /* L5 poziv*/
 void funJedan(int a, int *b, char v)
 {
  int jedan;
  ...
```



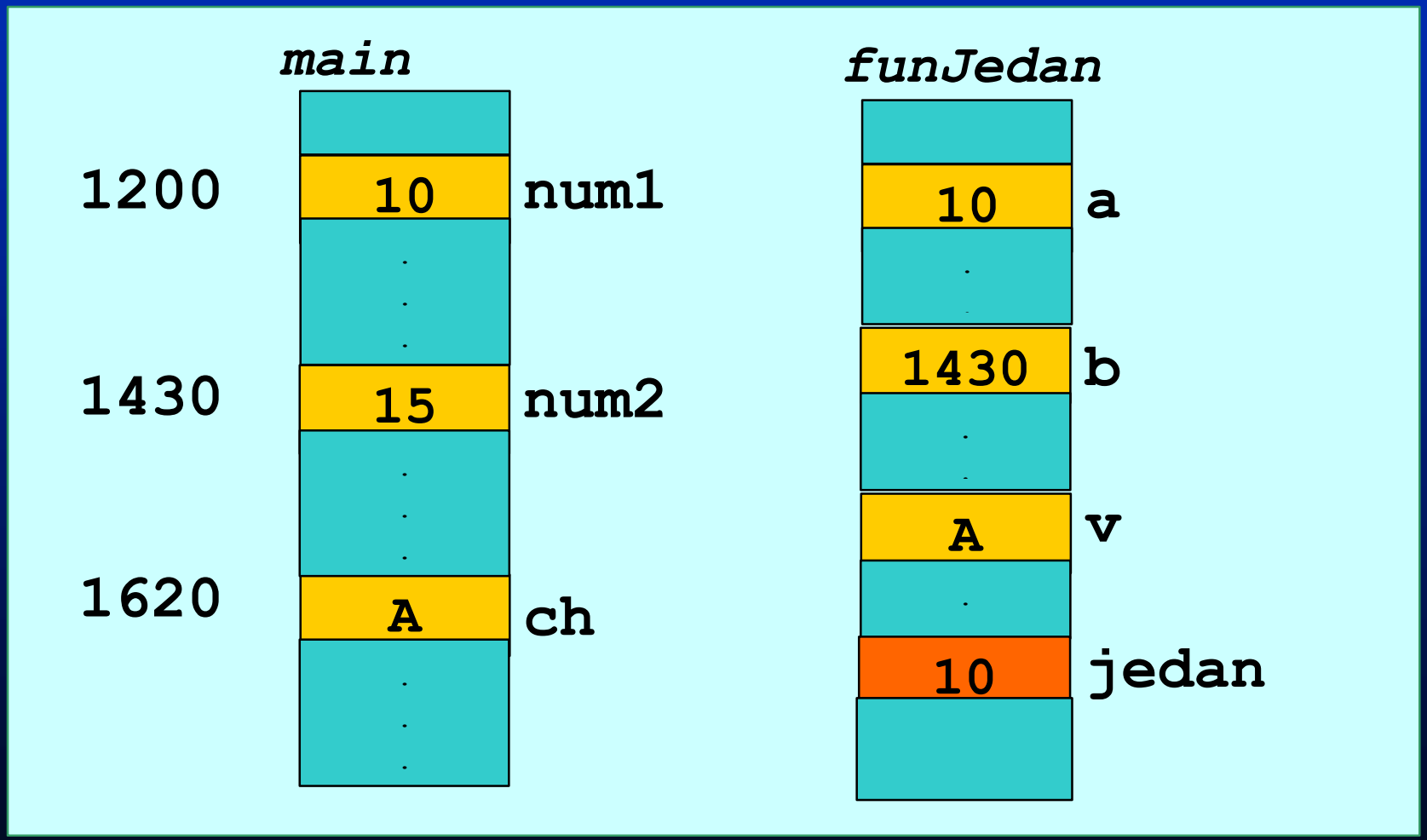
Пример 8/29

```
void funJedan(int a, int *b, char v)
{
 int jedan;
 jedan = a;              /* L 9 */
 a++;                    /* L10 */
 *b = (*b) * 2;          /* L11 */
 v = 'B';                /* L12 */
 printf("Linija 13: Unutar funJedan
  a=%d, *b=%d, v=%c\n",a,*b,v);
}
```

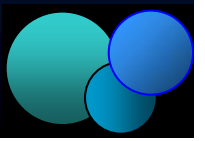
Пример 9/29

- У линији 9, након наредбе `jedan=a;`



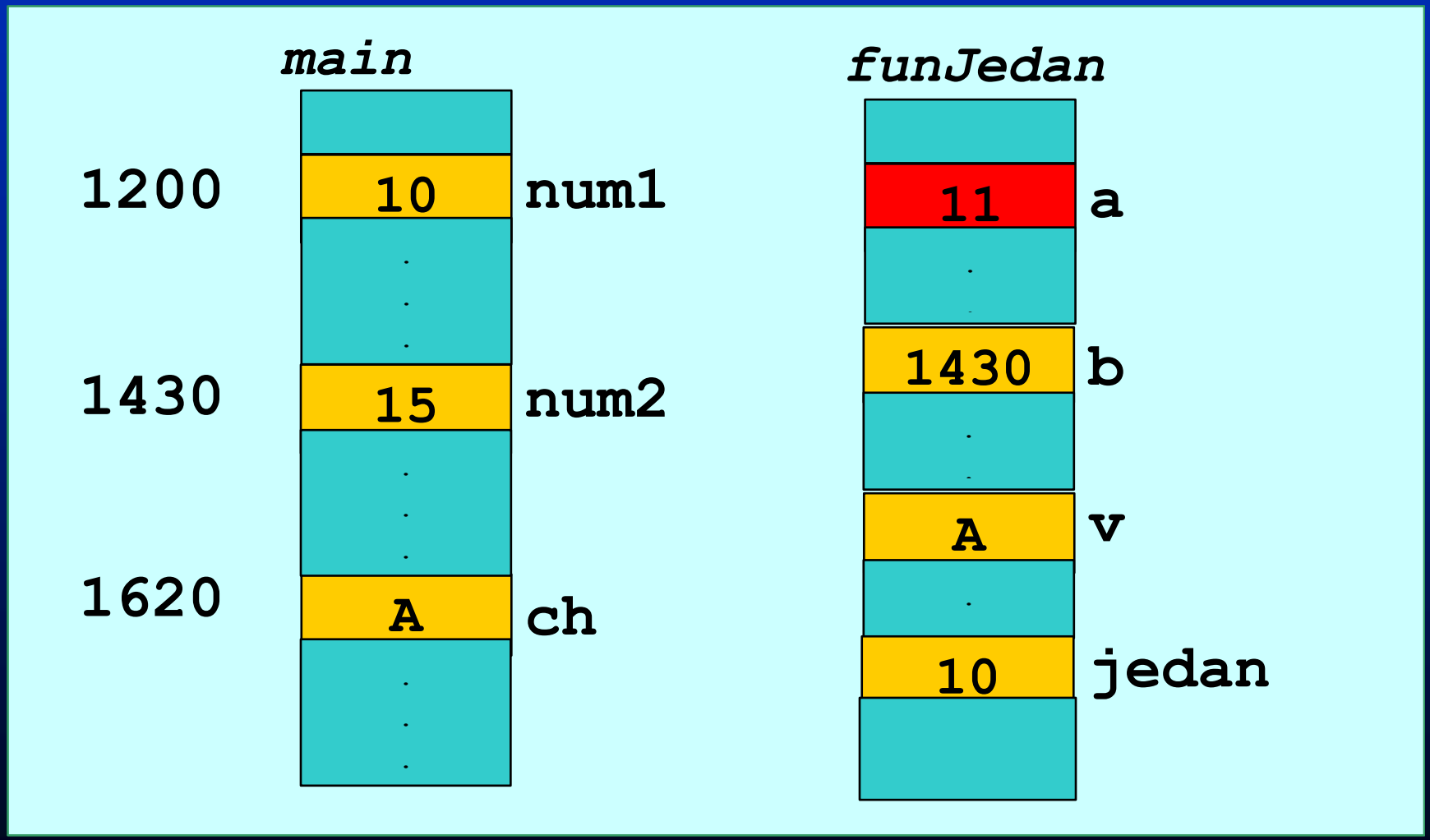
Пример 10/29

```
void funJedan(int a, int *b, char v)
{
 int jedan;
 jedan = a;          /* L 9 */
 a++;                /* L10 */
 *b = (*b) * 2;      /* L11 */
 v = 'B';            /* L12 */
 printf("Linija 13: Unutar funJedan
  a=%d, *b=%d, v=%c\n",a,*b,v);
}
```

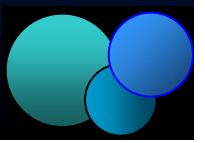
Пример 11/29

- Након линије 10 `(a++)`:



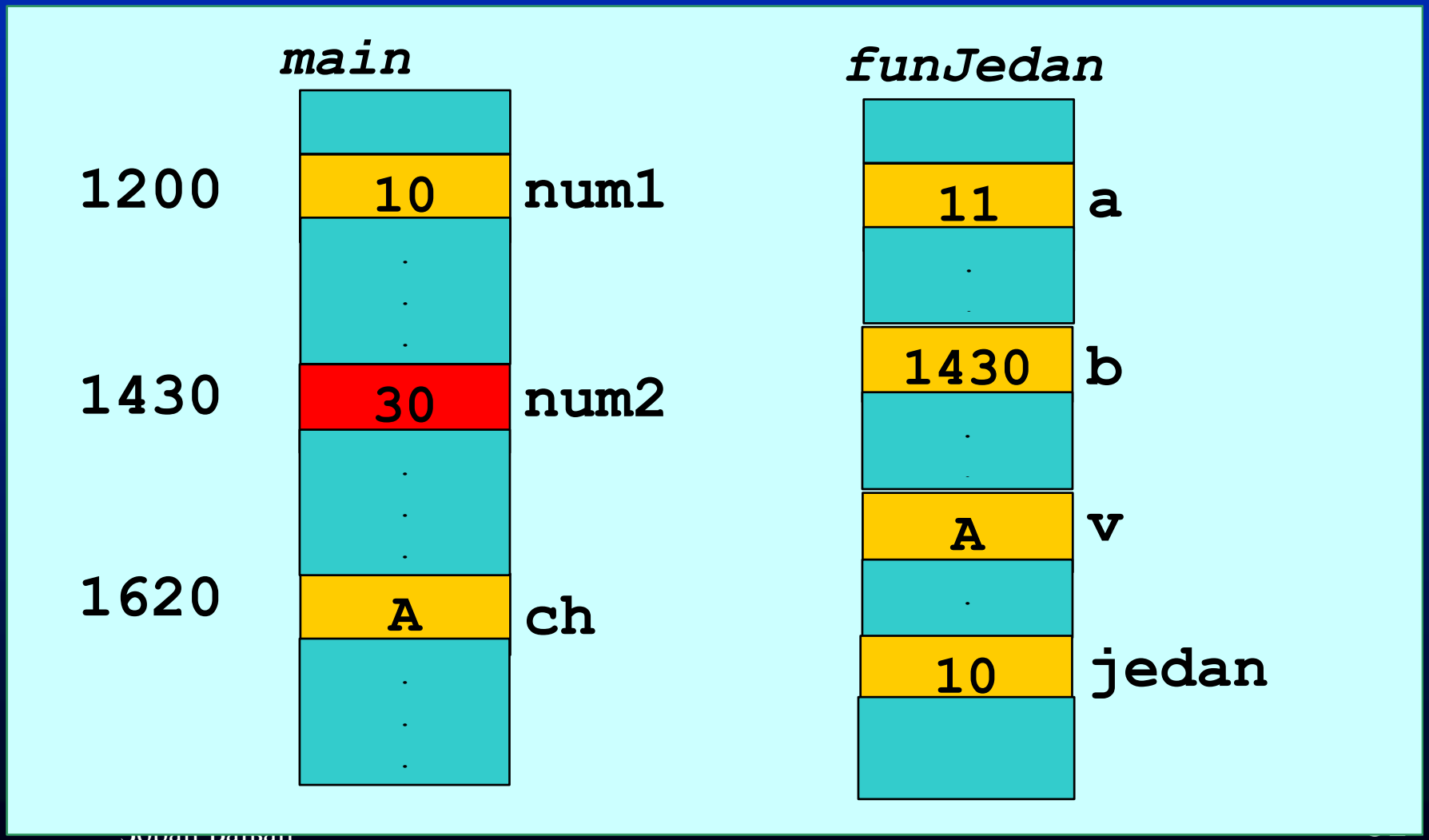
Пример 12/29

```
void funJedan(int a, int *b, char v)
{
 int jedan;
 jedan = a;          /* L 9 */
 a++;
 *b = (*b) * 2; /* L11 */
 v = 'B';            /* L12 */
 printf("Linija 13: Unutar funJedan
  a=%d, *b=%d, v=%c\n",a,*b,v);
}
```

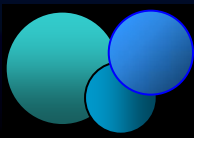
Пример 13/29

- Након наредбе `*b = (*b) * 2;` линија 11



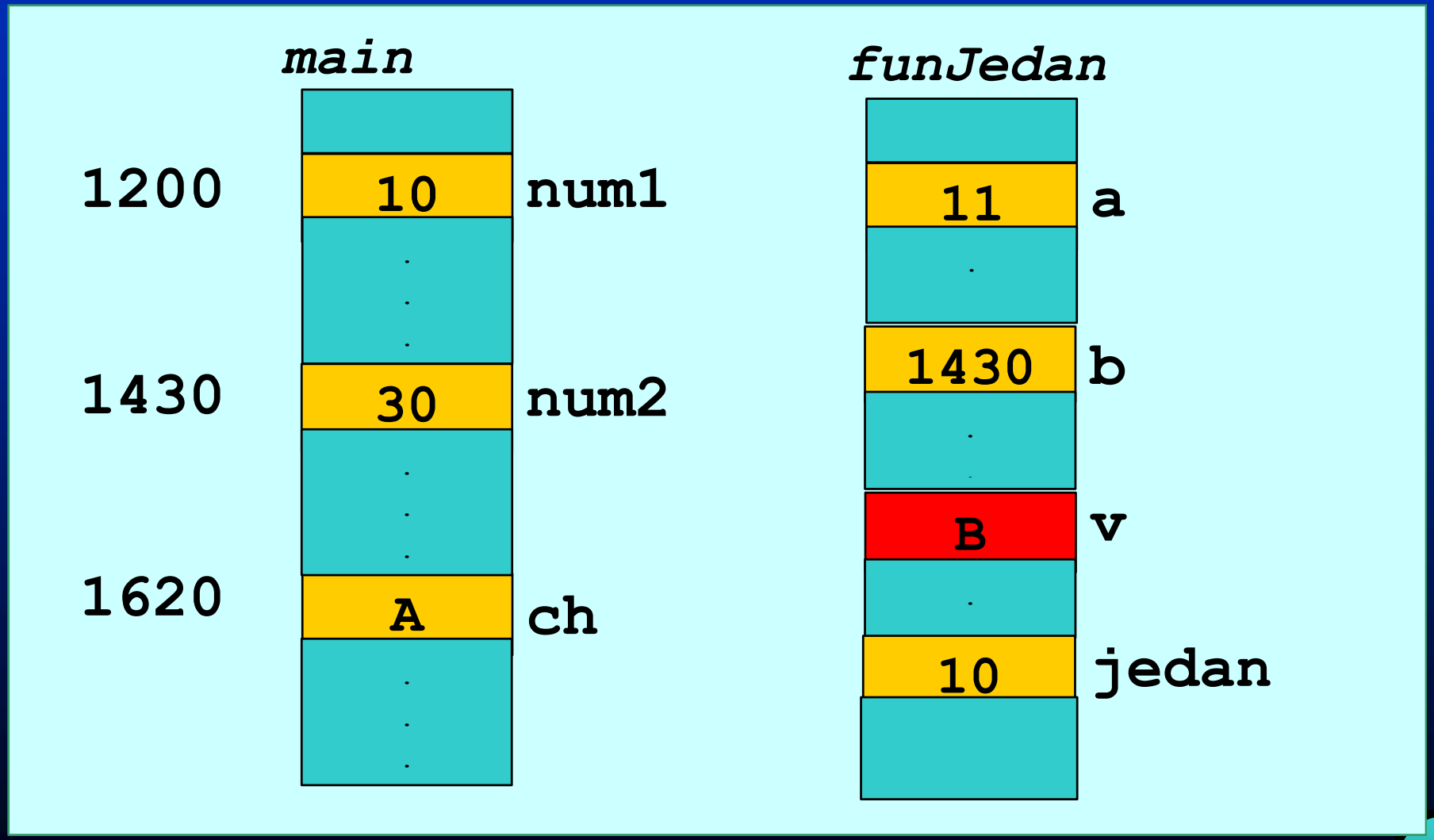
Пример 14/29

```
void funJedan(int a, int *b, char v)
{
 int jedan;
 jedan = a;          /* L 9 */
 a++;                /* L10 */
 *b = (*b) * 2;      /* L11 */
 v = 'B';            /* L12 */
 printf("Linija 13: Unutar funJedan
  a=%d, *b=%d, v=%c\n",a,*b,v);
}
```

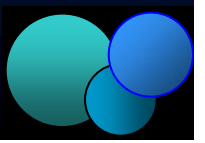
Пример 15/29

- Након наредбе `v = 'B'` у линији 12



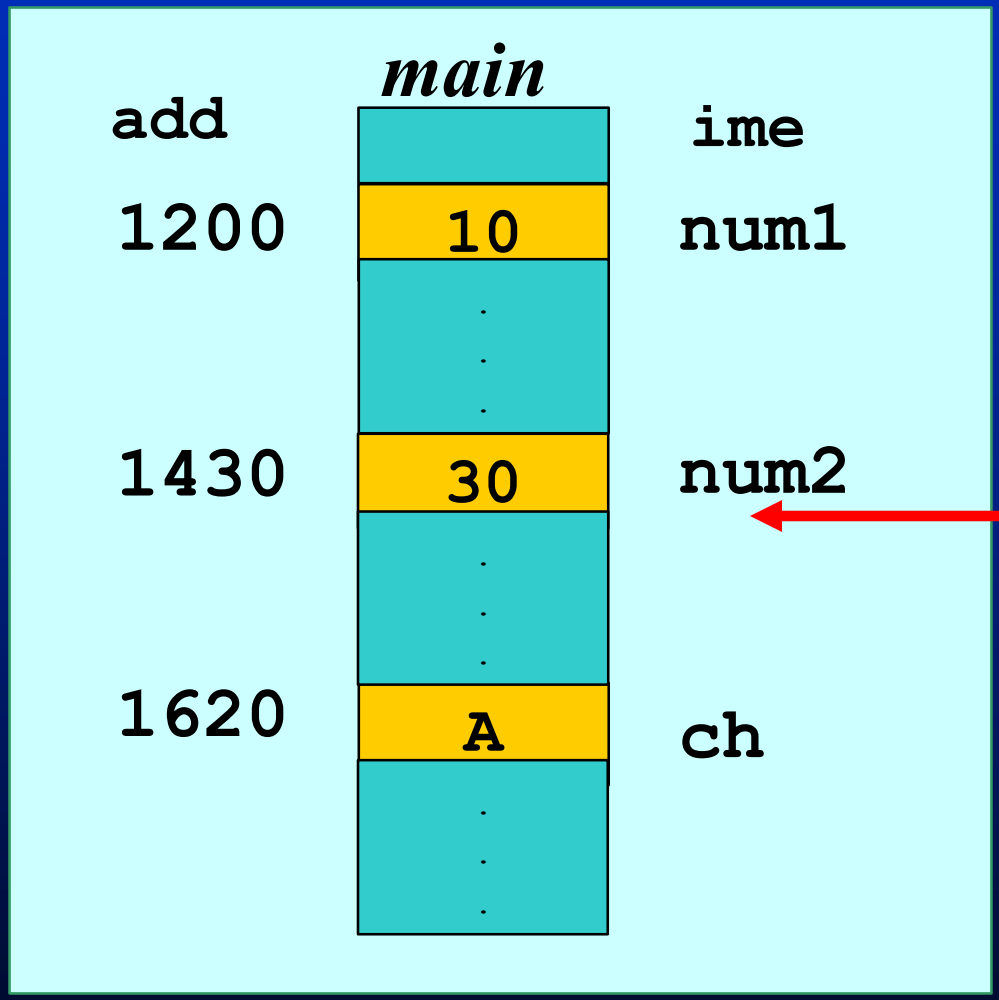
Пример 16/29

```
void funJedan(int a, int *b, char v)
{
 int jedan;
 jedan = a;          /* L 9 */
 a++;                /* L10 */
 *b= (*b) * 2;       /* L11 */
 v = 'B';
 printf("Linija 13: Unutar funJedan
  a=%d, *b=%d, v=%c\n",a,*b,v);
}
```

Пример 17/29

- Након наредбе у линији 13
- Напушта се функција **`funJedan`**
- Програм наставља да се извршава у L6 функције **`main`**

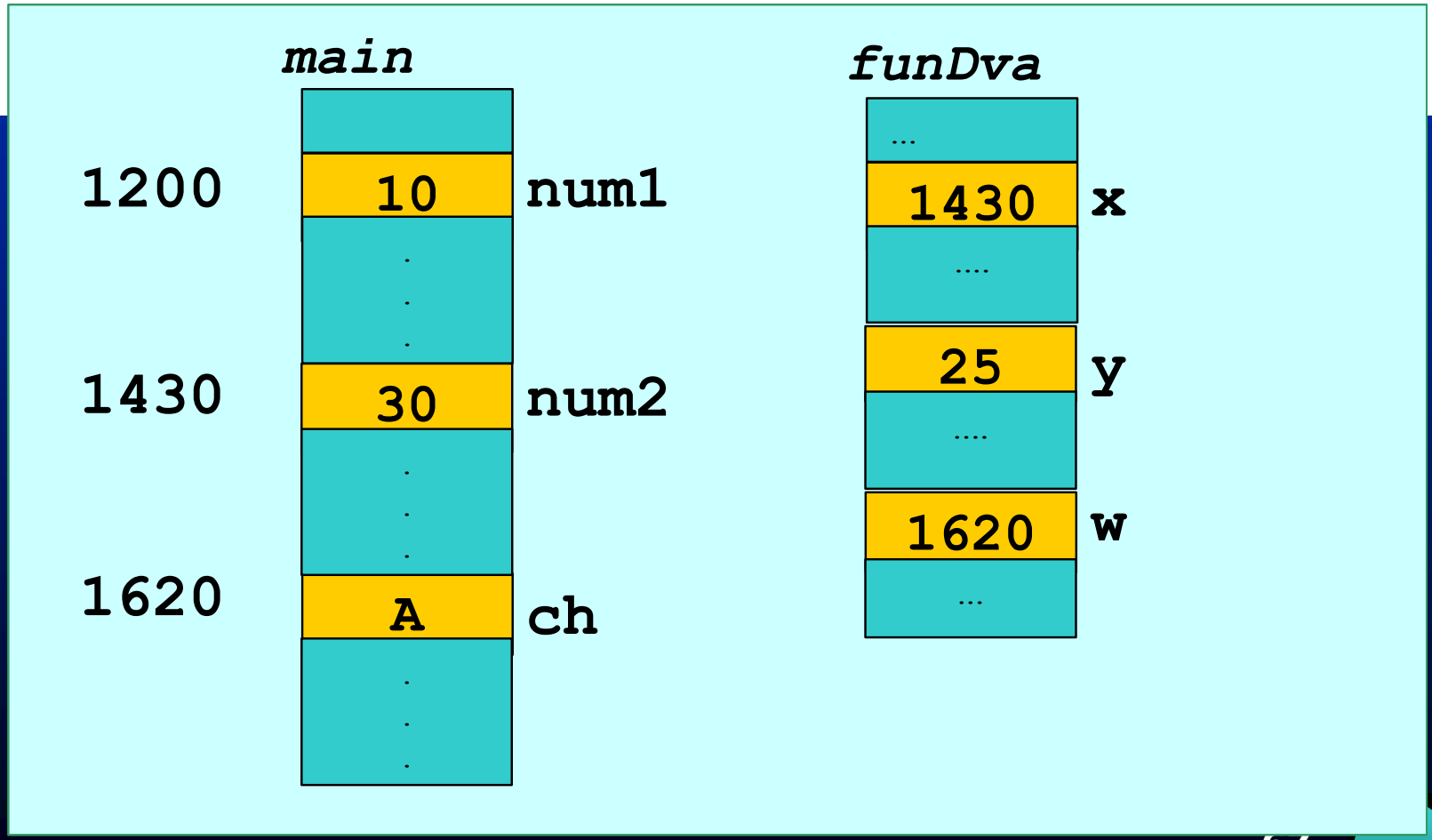


Пример 18/29

```
main( )
{
 int num1, num2;
 char ch;

 num1=10;                                /* L 1 */
 num2=15;                                /* L 2 */
 ch='A';                                 /* L 3 */
 printf("Linija  4: ....);
 funJedan(num1, &num2, ch);     /* L 5 */
 printf("Linija  6: Unutar...);
 num1 = funDva(&num2, 25, &ch); /* L 7*/
 printf("Linija  8: ...);
}
```

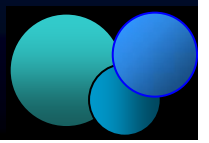
Пример 19/29

```
num1 = funDva(&num2, 25, &ch) /* L7  poziv*/
int funDva(int *x, int y, char *w);
{
  . . .
```



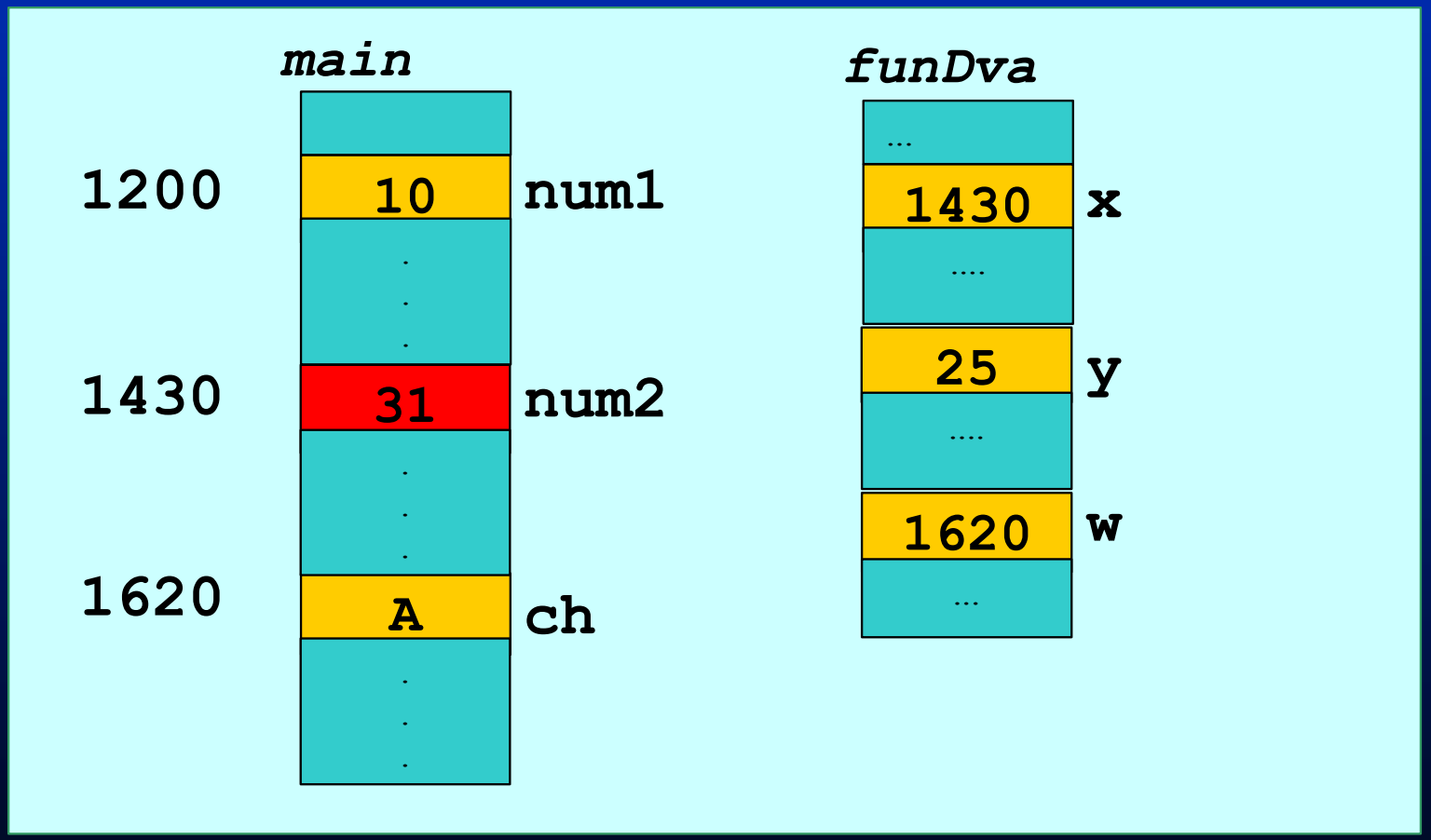
Пример 20/29

```
int funDva(int *x, int y, char *w)
{
 (*x)++;                /* L14*/
 y = y * 2;             /* L15*/
 *w = 'G';              /* L16*/

 return y;
}
```

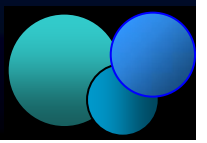
Пример 21/29

- Након извршавања наредбе у линији 14, `(*x)++`



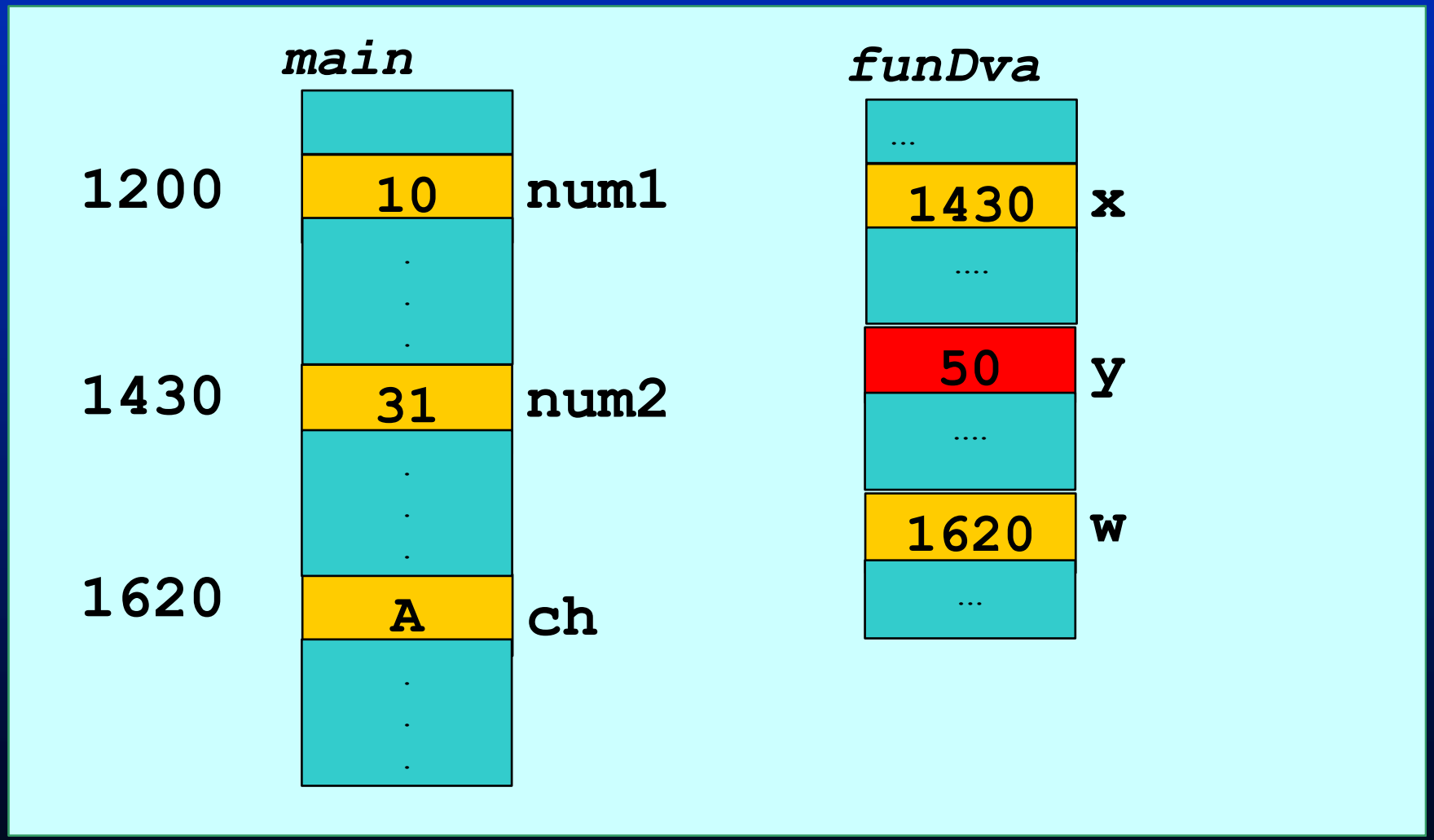
Пример 22/29

```
int funDva(int *x, int y, char *w)
{
 (*x)++;
 y = y * 2;              /* L15*/
 *w = 'G';               /* L16*/

 return y;
}
```

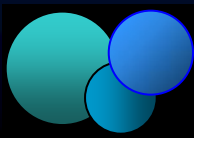
Пример 23/29

- Наредба у линији 15 `(y = y * 2)` даје



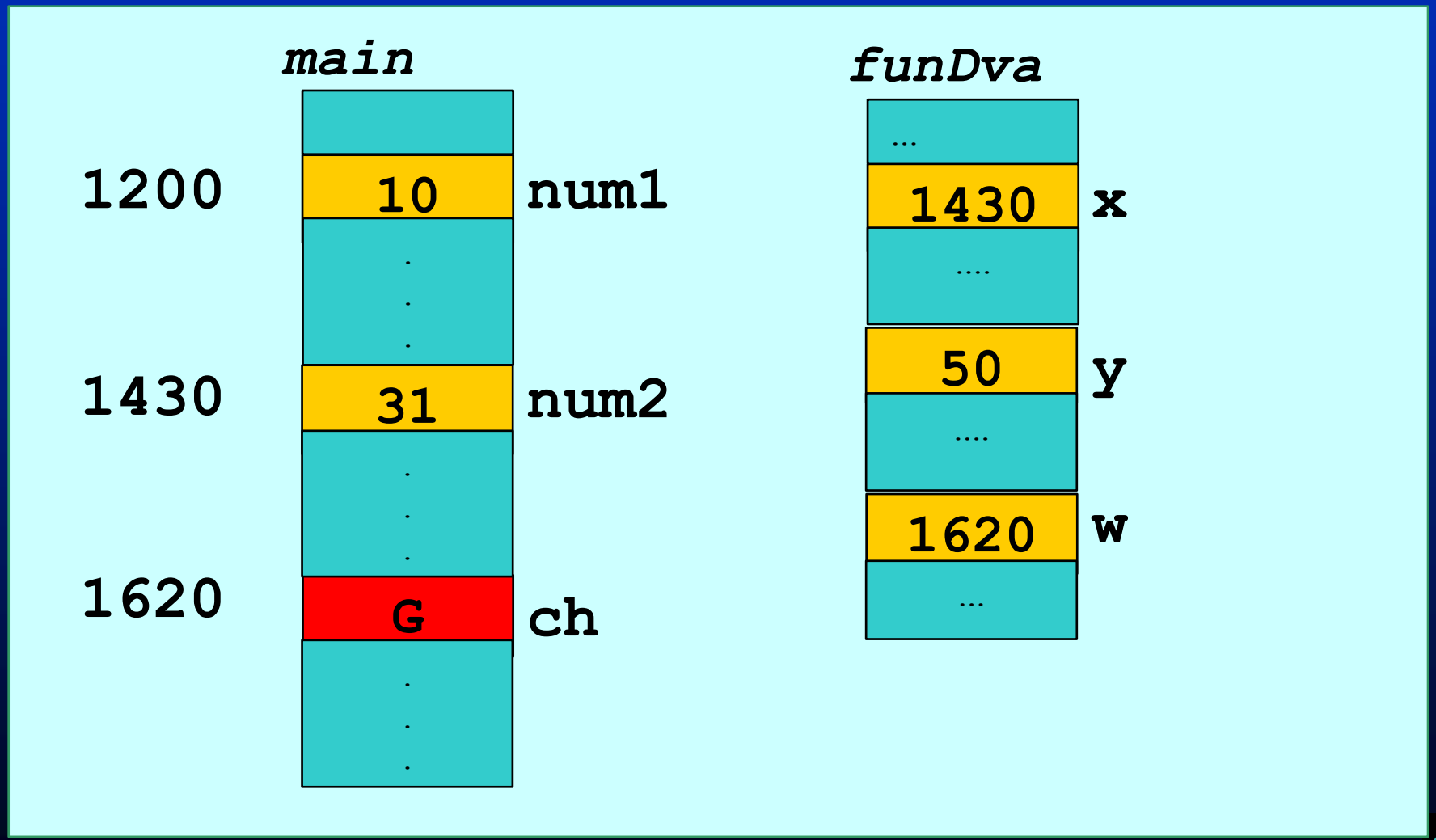
Пример 24/29

```
int funDva(int *x, int y, char *w)
{
 (*x)++;                /* L14*/
 y = y * 2;
 *w = 'G';              /* L16*/

 return y;
}
```

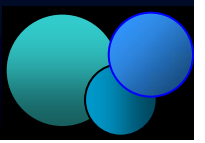
Пример 25/29

- Након линије 16 (`*w = 'G'`)



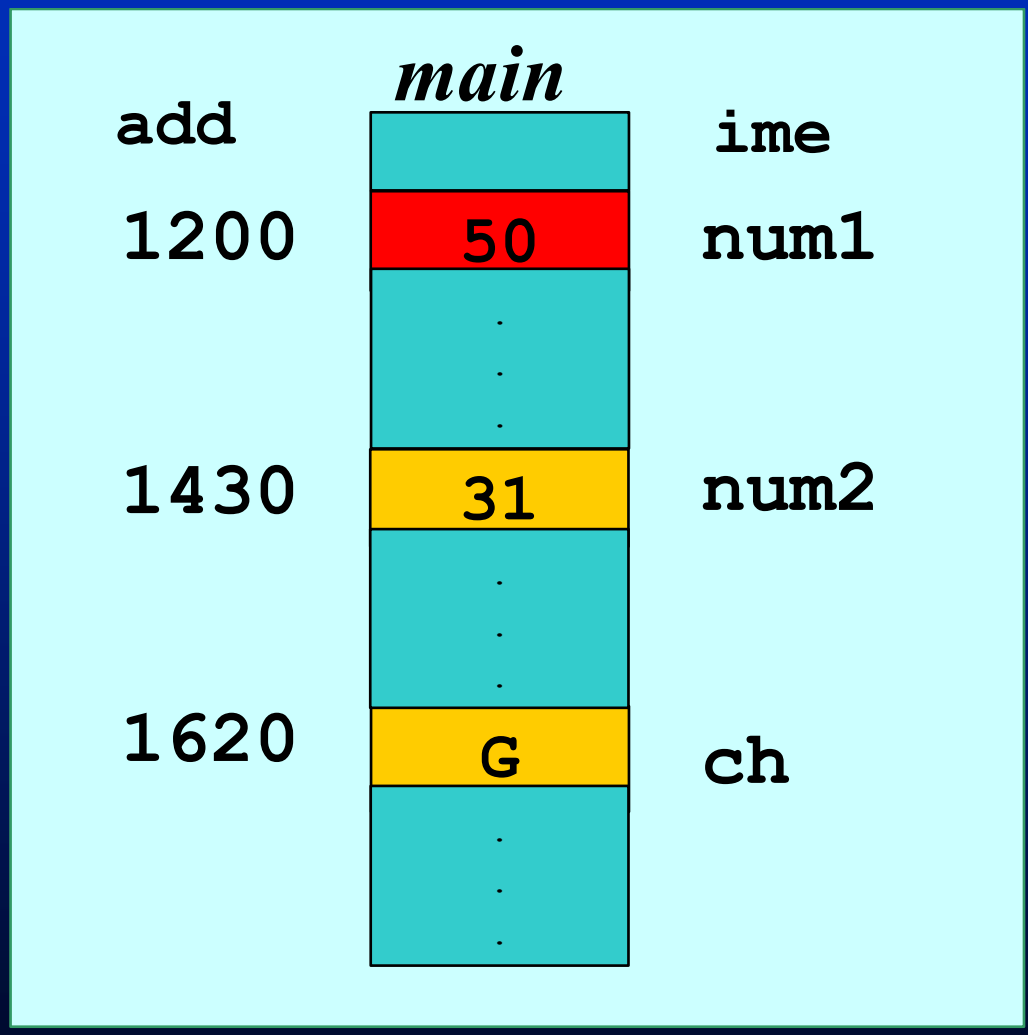
Пример 26/29

```
int funDva(int *x, int y, char *w)
{
 (*x)++;                /* L14 */
 y = y * 2;             /* L15 */
 *w = 'G';

 return y;
}
```

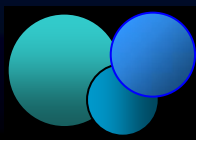
Пример 27/29

- Након наредбе у L17
- Напушта се функциа **`funDva()`**
- Враћа се у L7 функције **`main()`**
- Додељује се повратна вредност променљивој **`num1`**



Пример 29/29

- Након линије 8
- Прекида се извршавање програма
- Ослобађа се меморијски простор



Хвала на пажњи

Питања?