Основи програмирања 1
Лекција 10

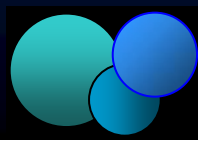
др Зоран Бањац

zoran.banjac@viser.edu.rs

Висока школа електротехнике и рачунарства
струковних студија
Београд

Садржај

- Функције и низови
- Рекурзивне функције
- Досег и меморијске класе променљивих
- Претпроцесорске наредбе



Структурно програмирање

- Програмски задатак је подељен на потпрограме
- могућ је независан дизајн, кодовање, и тестирање

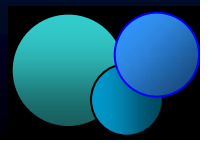
Програм који није структуиран

```
int main()
{
   /* uzmi podatke od korisnika */
   izraz1;
   izraz2;
   izraz3;
   /* Obrada podataka */
   izraz4;
   izraz5;
   izraz6;
   izraz7;
   /* Prikazivanje rezultata */
   izraz8;
   izraz9;
}
```

Структуиран програм

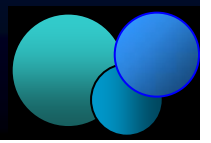
```
int main()
{
  /* poziv funkcije */
  prihvatPodataka(parametri);
  /* poziv funkcije */
  obradaPodataka(parametri);
  /* poziv funkcije */
  prikazRezultata(parametri);
}
```

- Функције могу да буду дефинисане у истом или неком другом фајлу
- Програм је много “читљивији” ако се имена функција пожљиво одабрана



Функције и низови

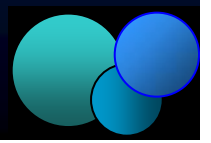
- Низови могу да се проследе у функцију као аргументи



Низ као аргумент функције

Низ се у функцију преноси путем адресе

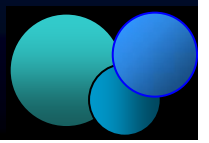
- **При позиву функције шаље се адреса низа**
  - **као стварни аргумент се наводи име низа (име низа је почетна адреса низа)**
- **Функција прихвата адресу низа**
  - **формални аргумент у дефиницији функције мора да садржи [ ]**
  - **број елемената низа не мора да се наводи**
  - **И у случају када се наведе број елемената низа, функција "не зна" број елемената низа.**



Низ као аргумент функције

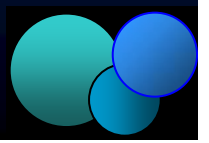
- Не ствара се копија низа већ се ради са стварним подацима

У функцији се манипулише стварним подацима, а не копијом низа. Зато функција може да промени низ, па по повратку из функције немамо оригинални него модификовани низ!



Пример

- Написати програм који учитава низ, позива функцију којој прослеђује тај низ. Функција проналази и враћа највећи елемент низа.
- Главни програм треба да прикаже враћену вредност



Пример: Низ као аргумент функције

```
#include <stdio.h>
int max( int niz[], int n);
int main()
{
   int a[] = {2,15,5,0,8}, naj;
   int n = 5;
   naj = max( &a[0], n );
   printf("Najveci: %d\n", naj);
   return 0;
}
int max( int niz[] , int n )
{
   int i, m = niz[0];
   for (i = 1; i < n; i++)
     if(niz[i] > m) m=niz[i];
   return (m);
}
```

Najveci: 15

Позив функције

Шаље се адреса низа

Формални аргумент

Декларација низа без димензије

Могло је и int niz[5]

Пример: Низ као аргумент функције

```
#include <stdio.h>
int max( int niz[], int n);
int main()
{
   int a[] = {2,15,5,0,8}, naj;
   int n = 5;

   naj = max( a    , n );
   printf("Najveci: %d\n", naj);
   return 0;
}

int max( int niz[], int n )
{
   int i, m = niz[0];
   for (i = 1; i < n; i++)
     if(niz[i] > m) m=niz[i];
   return (m);
}
```

Име низа је уједно и адреса елемента на индексу 0 (почетна адреса низа)

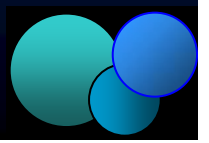
Пример: Низ као аргумент функције

```
#include <stdio.h>
int max( int niz[], int n);
int main()
{
  int a[] = {2,15,5,0,8};
  int n = 5;
  printf("Najveci: %d\n", max( a, n ) );
  return 0;
}

int max( int niz[], int n )
{
  int i, m = niz[0];
  for (i = 1; i < n; i++)
    if(niz[i] > m) m=niz[i];
  return (m);
}
```

Позив функције max() у оквиру позива функција printf()

Пример

- Написати програм који поред функције **`main`** садржи још две функције:
  - за испис вредности низа
  - за сортирање низа
- у главном програму исписати вредност низа пре сортирања (позив функције) позвати функцију за сортирање и након тога поново исписати вредности низа (позив функције) .



Пример: Низ као аргумент функције

```
#include <stdio.h>
void sort(int b[],int n);
void pisi(int b[],int n);
main()
{
 int a[] = { 2,15,5,0,8 };
 printf("Pre:");
 pisi(a, 5);
 sort(a, 5);
 printf("Posle:");
 pisi(a, 5);
}

void pisi(int b[], int n)
{ int i;
  for (i=0; i<n; i++)
     printf(" %d", b[i]);
   printf("\n");
}
```

```
void sort(int b[],int n )
{
 int i, j, pom;
 for (i = 0; i<n-1; i++)
 {
  for (j=0; j<n-1-i; j++)
    if (b[j] > b[j+1])
    {
       pom = b[j];
       b[j] = b[j+1];
       b[j] = pom;
    }
  }
}
```

Pre: 2 15 5 0 8
Posle: 0 2 5 8 15

Пример

- Написати функцију која проналази средњу вредност низа

```
float sredVred(int dim, float lista[])
{
 int i;
 float suma = 0.0;
 for (i=0; i < dim; i++)
   suma += lista[i];

 return(suma/dim);
}
```

димензија низа

Низ, код прослеђивања једнодинензионалног низа НЕ наводе се димензије

тип низа је једнак типу повратне вредности

Вишедимензионални низ као аргумент функције

Вишедимензионални низ се преноси помоћу адресе

У декларацији формалних аргумената:

- прва димензија не мора да се наведе,може само [ ] .
- остале димензије морају да се наведу

```
int funkcija (int a[][10][10], int p, int q)
int funkcija (int a[10][10][10], int p, int q)
```

У функцији се ради са стварним подацима,а не са копијом низа.

Пример: приказ 2д низа

```
#include <stdio.h>
/* prototip funkcije prikaziNiz */
void prikaziNiz(int dim1,int dim2,
                                float niz[][5]);
int main()
{
  float a[2][5] ={{2,15,5,0,8},{1,2,3,4,5}};
  int n = 2,m=5;

  /* poziv funkcije prikaziNiz */
  prikaziNiz(2, 5, a);

  return 0;
}
```

Пример: приказ 2д низа

```
void prikaziNiz(int dim1,int dim2, float
                                  niz[][5])
{
  int x, y;
  for ( x = 0; x < dim1; x++)
  {
    for ( y = 0; y < dim2; y++)
      printf("%f ", niz[x][y]);
    printf("\n");
  }
}
```

**Низ,
код прослеђивања
вушедимензионалног низа НЕ
наводи се прва димензија, али
се наводе све остале**

Пример: Низ као аргумент функције

Програм позива потпрограме за испис и транспоновање матрица.

```
#include <stdio.h>
void tran( int m[][3], int n);
void pisi( int m[3][3], int n);
main()
{
   int dim = 3;
   int a[3][3] = { {1,2},{3,4} };
   printf("Pre:\n");
   pisi(a, dim); /* poziv funk. za ispis */
   tran(a, dim ); /* poziv funk. za transp. */
   printf("Posle:\n");
   pisi(a, dim); /* poziv funk. za ispis */
}
```

```
Pre:
   1   2   0
   3   4   0
   0   0   0
Posle:
   1   3   0
   2   4   0
   0   0   0
```

Дефиниција функ. pisi()

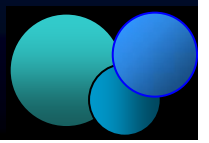
```
void pisi (int m[3][3], int n)
{
   int i,j;
   for (i = 0; i < n; i++)
   {
     for (j = 0; j < n; j++)
       printf(" %4d", m[i][j]);
     printf("\n");
   }
   return;
}
```

прва димензија може (не мора) да се наведе, друга се обавезно наводи

Дефиниција функ. tran()

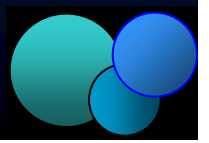
```
void tran(int m[ ][3], int n)
{
   int i, j, pom;
   for (i = 0; i < n; i++)
      for (j = i+1; j < n; j++)
      {
         pom = m[i][j];
         m[i][j] = m[j][i];
         m[j][i] = pom;
      }
   }
}
```

Рекурзивне функције



Рекурзивне функције

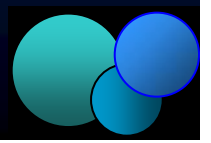
- Функције које могу да позову саме себе
- Своде сложен проблем на једноставнији проблем (исте природе) који могу да реше.



Рекурзивне функције

- Пример: рачунање факторијела

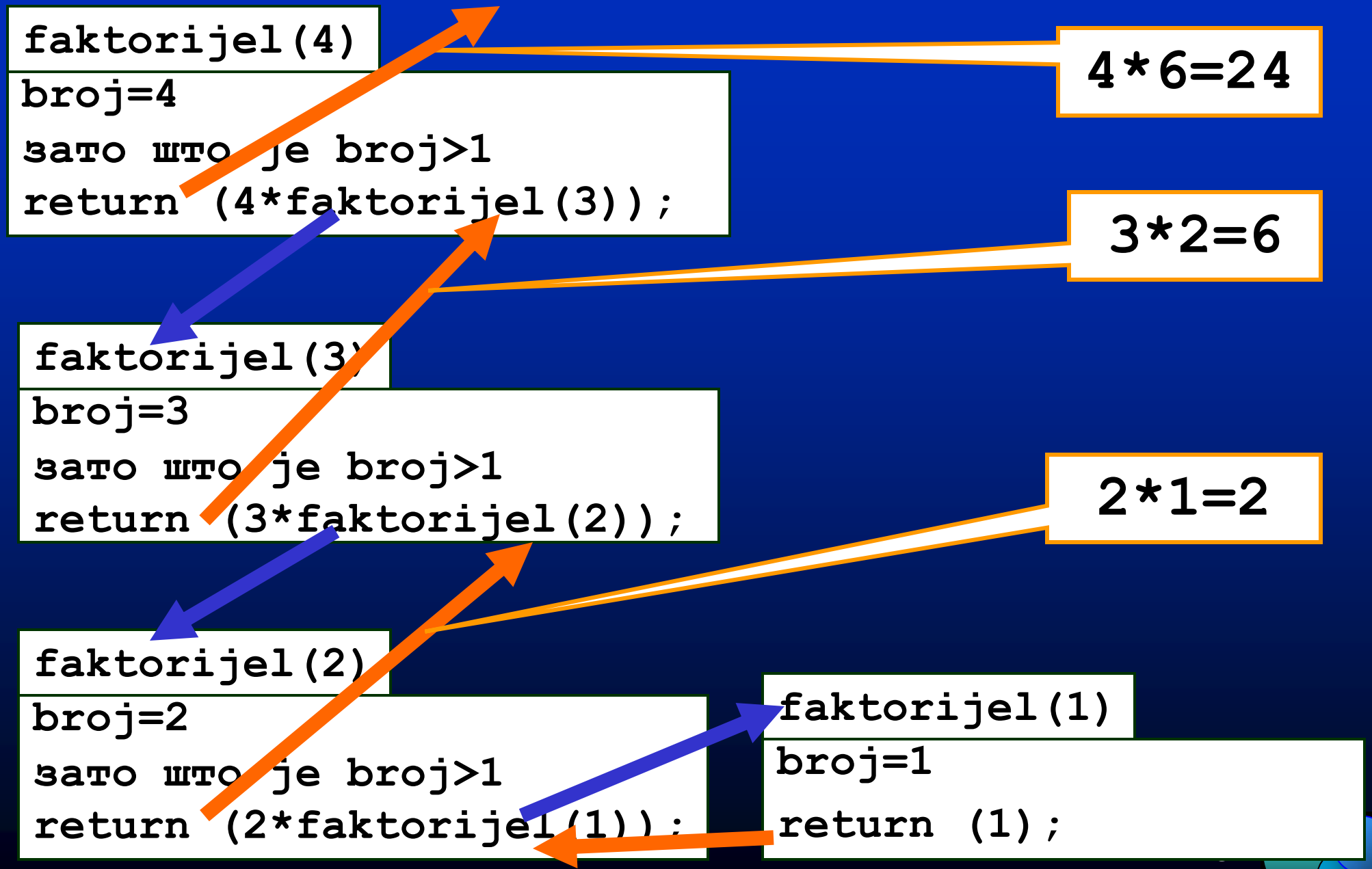
```
n! =n*(n-1)*(n-2)*(n-3)*...*1
1! = 1
0! = 1

4! = 4*3*2*1=4*(3*2*1) = 4*(3!)
3! = 3*2*1=3*(2*1)     = 3*(2!)
2! = 2*1=2*(1)         = 2*(1!)
1! = 1
```



Рекурзивне функције

```
/* prototip funkcije */
long faktorijel(long broj);

main()
{
   ...
   faktorijel(4); /* poziv funkcije */
   ...
}

long faktorijel(long broj)
{
 if( broj <= 1 )
    return 1;
 else
    return (broj * faktorijel(broj - 1) );
}
```

Рекурзивне функције



Рекурзивне функције

- Рекурзивна функција је функција која сама себе позива

  факторијел: $n! = n * (n-1)!, n>0;\ 0!=1$

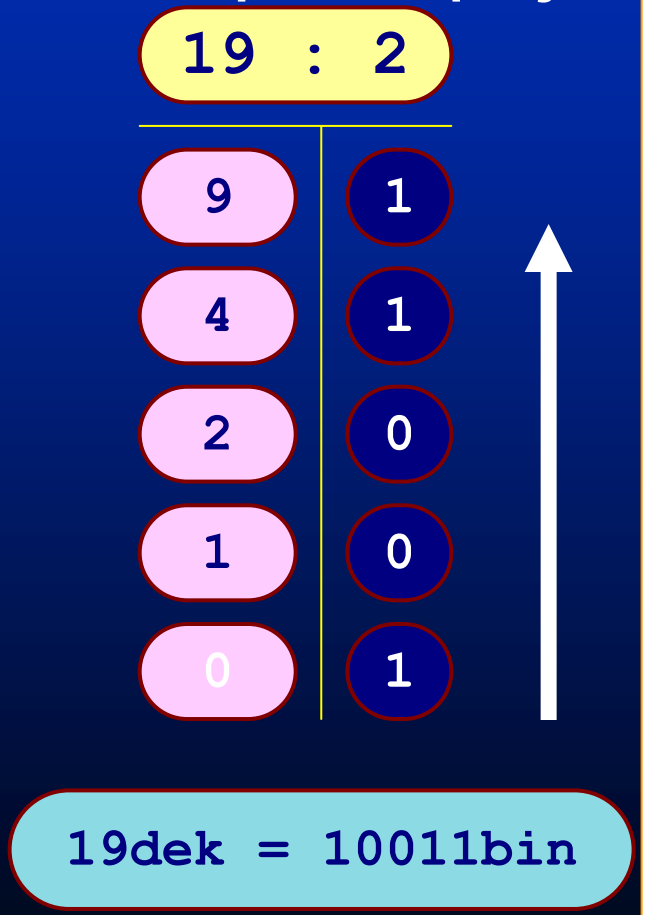
  степеновање: $x^n = x * x^{n-1}, n>0;\ x^0=1$

- Опште карактеристике:

- Проблем се своди на решавање поједностављеног проблема
- Функција позива саму себе док проблем не поједностави до тривијалног
- Рекурзивна решења су мање ефикасна (већи захтев за меморијом)
- Треба их избегавати, али постоје проблеми који су по природи рекурзивни

Пример рекурзивне функције

- Конверзија декадног у бинарни број




```
#include<stdio.h>
void kon (int b); /* prototip */
int main()
{ int a;
  printf("unesite broj:");
  scanf("%d", &a);
  kon(a); /* poziv funkcije */
}
void kon (int b)
{
  int cif;
  cif = b % 2;
  b = b/2;
  if (b > 0)
     kon(b); /* poziv funkcije*/
  printf("%d",cif);
}
```

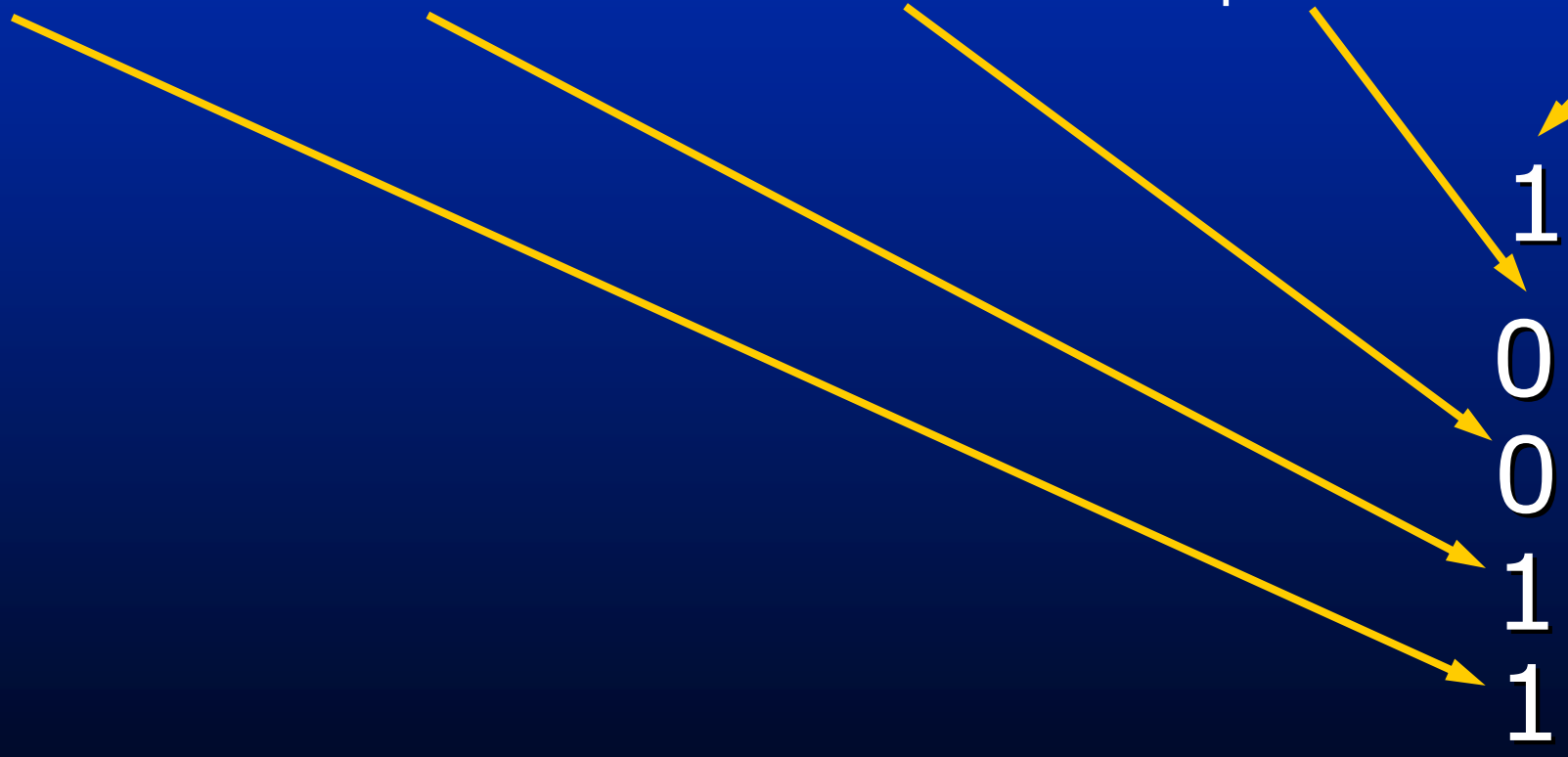
kon(19)
cif=19%2=1
b=19/2=9
kon(9)
ispisi cif

kon(9)
cif=9%2=1
b=9/2=4
kon(4)
ispisi cif

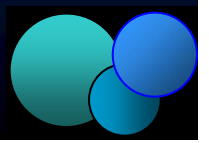
kon(4)
cif=4%2=0
b=4/2=2
kon(2)
ispisi cif

kon(2)
cif=2%2=0
b=2/2=1
kon(1)
ispisi cif

kon(1)
cif=1%2=1
b=1/2=0
ispisi cif

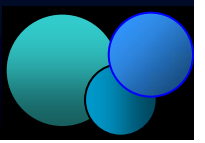


Досег и меморијске класе променљивих



Досег *(Scope)*

- **Досег** (област дефинисаности, област важења, видљивост) идентификатора (променљиве) је део програма у ком је могуће да се приступи идентификатору по имену
- Основно правило:
  - Идентификатор је доступан у блоку у којем је дефинисан, као и у свим угњежденим блоковима, осим ако у њима није маскиран другим идентификатором са истим именом!



Пример

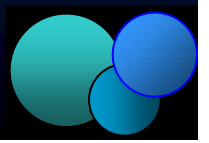
У угњежденом блоку је дефинисана променљива са истим именом, чиме је маскирана променљива x прве линије

```
#include <stdio.h>
main()
{
 int x = 1;
 printf("Pre bloka: %d\n", x);
 {
  int x = 0;
  printf("U bloku: %d\n", x);
 }
 printf("Posle bloka: %d\n", x);
}
```

```
Pre bloka: 1
U bloku: 0
Posle bloka: 1
```

Меморијске класе променљивих

- У програмском језику C се разликују 4 меморијске класе променљивих:
  - глобалне
  - статичке
  - аутоматске
  - регистрске



Меморијске класе променљивих

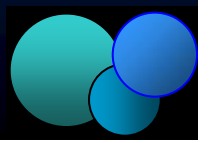
- Глобалне променљиве
  - дефинишу се изван свих функција, може им се приступити из било ког дела фајла у ком су декларисани.
- Статичке променљиве
  - дефинишу се помоћу кључне речи **`static`**

    `static tip ime = vrednost`
- Аутоматске променљиве
  - дефинишу се помоћу кључне речи **`auto`**

    `auto tip ime = vrednost`
  - подразумева се да је променљива аутоматска, ако се другачије не наведе
- Регистрске променљиве
  - дефинишу се помоћу кључне речи **`register`**

Аутоматске променљиве

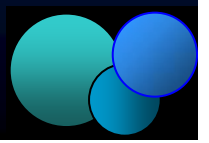
- Дефинишу се на почетку блока (главног програма или потпрограма). Често се користи термин локална променљива.
- Почетна вредност им се не подразумева. (није нула!)
- Могу да се користе само у функцији у којој су дефинисане. То значи да исто име можемо да користимо независно у више различитих функција. Те променљиве могу да буду и различитих типова.



Аутоматске променљиве

- Аутоматски настају приликом уласка у функцију, и аутоматски нестају након изласка из функције.
- Подразумева се да је променљива аутоматска ако се другачије не наведе.
- Није неопходно да се изричито наведе кључна реч auto, али може

```
auto tip ime = vrednost;
```



Пример: аутоматске промен.

```
#include <stdio.h>
void f1 ();
void f2 ();

main()
{
   int i = 0;
   printf("main: %d\n", i);
   f1();
   printf("main: %d\n", i);
   i = 5;
   f2();
   printf("main: %d\n", i);
}
```

```
void f1 ()
{
 float i = 1.0;
 printf("f1: %f\n", i);
}

void f2 ()
{
 int i = 2;
 printf("f2: %d\n", i);
 i = 10;
}
```

```
main: 0
f1: 1.0
main: 0
f2: 2
main: 5
```

Статичке променљиве

- Дефинишу се навођењем кључне речи **`static`**

```
static tip ime = vrednost
```

- Додела почетне вредности се обавља само први пут, без обзира на број позива функције
- Подразумева се да је почетна вредност нула, ако се другачије или ништа не наведе
- Имају трајан карактер
- Постоје у меморији од почетка до краја извођења програма
- Вредност остаје сачувана до следећег позива те функције

Пример: Статичке променљиве

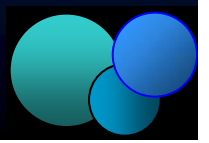
```
#include <stdio.h>
void f()
{
  static int brs = 1;
  auto int bra = 1;
  printf("static brs=%d auto bra=%d\n", brs, bra);
  brs++;
  bra++;

  return;
}
int main()
{
 int i;
 for (i = 1; i <= 3; i++)
    f();

 return 0;
}
```

static brs=1 auto bra=1
static brs=2 auto bra=1

static brs=3 auto bra=1

Глобалне променљиве

- Променљиве које су заједничке за већи број функција.
- Дефинишу се изван било које функције.
- Досег :
  од места где је дефинисан до краја фајла у ком се налази та наредба.



Пример: Глобалне променљиве

```
void f1()
{
 int i = 1;
 printf("f1: %d\n",i);
}

int i=0;

void f2()
{
 printf("f2: %d\n",i);
 i = 10;
}
```

Глобална променљива

У функцијама `main()` и `f2()` се користи глобална променљива `i`

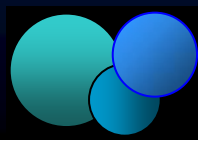
Глобална променљива није видљива у функцији `f1()` јер је дефинисана након ње

```
#include <stdio.h>
int main()
{
 printf("main:%d\n",i);
 f1();
 printf("main: %d\n",i);
 i = 5;
 f2();
 printf("main: %d\n",i);
 return 0;
}
```

```
main:0
f1:1
main:0
f2:5
main:10
```

Глобалне променљиве

- Функције које сачињавају програм не морају да се дефинишу у једном фајлу.
- Скуп тих фајлова: **пројекат**.
- Сваки фајл унутар пројекта се преводи независно.
- Након успешног превођења, преведени фајлови се повезују (*linking*) у извршни програм.

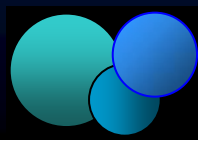


Глобалне променљиве

- Глобални идентификатори могу да имају досег на све фајлове у пројекту.
- У сваком од фајлова треба да постоји декларација глобалног податка или функције која ће се у њему користити.
- У само једном од фајлова треба да постоји дефиниција (функције и/или глобалне пром).
- У сваком фајлу у ком се користи глобална променљива, дефинисана у другом фајлу, мора постојати тзв. екстерн (*extern*) декларација.

Глобалне променљиве

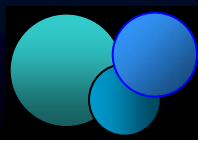
- Подразумевана почетна вредност глобалне променљиве је нула!
- Глобална променљива је доступна у свим функцијама које су дефинисане након ње.
- Могућност да више функција манипулише истим скупом података и да се ефикасно размењују подаци између функција (јер све функције приступају истим меморијским локацијама).



Глобалне променљиве

Лоше стране

- Треба их избегавати јер се губи универзалност функције и флексибилност примене на различите сетове података.
- Ако се у некој функцији дефинише променљива са истим именом као нека глобална, тада та локална променљива маскира глобалну и глобалној не може да се приступи из те функције.



Глобалне променљиве

FILENAME1.c

```
 1  extern int brojac; /*deklaracija gl. prom*/
 2  int rezultat = 2;  /*definicija gl. prom*/
 3  void funkcija_1(); /* prototip */

 4  int main()
 5  {
 6   int i = 4;
 7   printf("M7: %d, %d\n", brojac, rezultat);
 8   rezultat++;
 9   brojac = rezultat*i;
10   printf("M10: %d, %d\n", brojac, rezultat);
11   funkcija_1();
12   printf("M12: %d, %d\n", brojac, rezultat);
13   return 0;
14  }
```

```
M7: 0, 2
M10: 12, 3
F6: 12 3
F9: 1 5
M12: 1, 5
```

Глобалне променљиве

FILENAME2.c

```
1   #include<stdio.h>
2   extern int rezultat;
3   int brojac = 0;

4   void funkcija_1()
5   {
6    printf("F6: %d %d\n", brojac,rezultat);
7    rezultat = 5;
8    brojac = 1;
9    printf("F9: %d %d\n", brojac,  rezultat);
10   return;
11  }
```

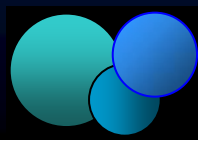
```
M7: 0, 2
M10: 12, 3
F6: 12 3
F9: 1 5
M12: 1, 5
```

Регистарске променљиве

- Регистарске променљиве су аутоматске променљиве које се држе у регистрима процесора а не у меморији, како би се добило на брзини
- Пожељно је да се користе код променљивих које често мењају вредност (нпр. бројачи)
- Не гарантује се да ће променљива стварно бити регистарска. То зависи од конкретног процесора и проводиоца, броја регистарских променљивих...
- Број регистарски променљивих је ограничен
- Декалпрација уз употребу речи **`register`**

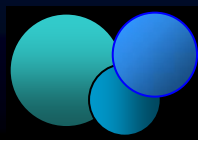
```
register tip ime = vrednost

register int i=10;
register float j;
```

- Претпроцесорске наредбе



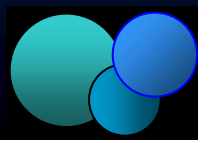
Претпроцесор

- Уметање садржаја фајла (наредба **`#include`**)
- Замена лексичких симбола (наредба **`#define`**)
  - Дефинисање симболичких константи
  - Дефинисање макро-а
- Условно превођење (наредба **`#if`**, **`#ifdef`**, ...)



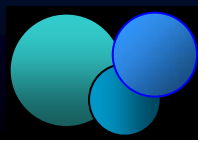
Претпроцесор

- Претпроцесорске наредбе не представљају део програмског језика *C*
- Претпроцесор је део преводиоца који треба да изврши предобраду кôда.
- Обавља одговарајуће трансформације текста којима се добија коначан облик изворног кôда који треба да буде преведен.



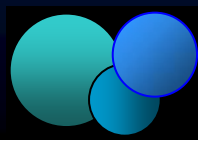
Претпроцесор

- Претпроцесорска наредба се пише у засебном реду и почиње са знаком **#**.
- Испред ње не могу бити друге наредбе.
- Пре превођења замењује текст у изворном кôду.



Претпроцесор

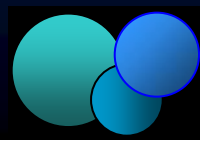
- Претпроцесорске наредбе се не завршавају знаком ;
- Могу да се уведу на било ком месту у програму.
- Дејство: од појављивања па надаље, док се посебном наредбом не укину или до краја програма.



Претпроцесор

- Неке од важнијих претпроцесорских наредби које препоручује *ANSI* стандард су следеће:

```
#include
#define
#if
#ifdef
#ifndef
#elif
#else
#endif
#undef
```

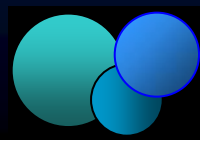


Уметање садржаја фајла

Наредба **#include**

```
#include "ime fajla"
#include <ime fajla>
```

- Тражени фајл може и сам да садржи наредбу **#include**.



Замена лексичких симбола

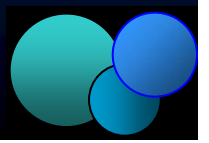
Наредба **#define**

- Користи се за дефинисање симболичких константи и макро-а.

```
#define SIMBOLICKA_KONSTANTA ZnakovniNiz
```

Пример:

```
#define PI  3.1415926
#define NASLOV "Programski jezik C"
#define ZAUVEK for( ; ; )
```



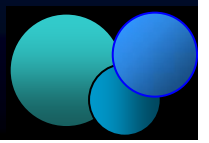
Дефинисање симболичке константе

- Претпроцесор претражује изворни кôд и након појављивања наредбе облика

```
#define SIMBOLICKA_KONSTANTA ZnakovniNiz
```

свако појављивање **`SIMBOLICKA_KONSTANTA`** замењује са **`ZnakovniNiz`**.

- у коментару или унутар неког другог знакованог низа неће доћи до замене.

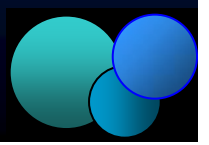


Дефинисање симболичке константе

```
#include<stdio.h>
#define PI 3.1415926

int main()
{
 float a;
 float r = 4.23;
 char ch[ ]="PILICI"; /* slogovi PI LI CI*/
 a = 2 * r * PI;
 printf("Niz ch je %s
       a obim a je %f\n ",ch,a);
 return 0;
}
```

Излаз

```
Niz ch je PILICI a obim a je 26.577874
```

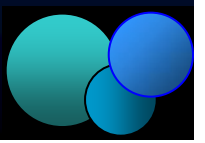



Дефинисање симболичке константе

```
#include <stdio.h>
#define WIDTH    80                    /* L1 */
#define LENGTH   ( WIDTH + 10 ) /* L2 */
int main()
{
 int var;
 var = LENGTH * 20; /* var=(80+10)*20;*/
 printf("var=%d\n",var)
 ...
 return 0;
}
```

излаз

```
var=1800
```

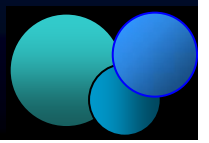
Дефинисање симболичке константе

- Значење претпроцесорске наредбе

```
#define SIMBOLICKA_KONSTANTA
```

се може укинути наредбом:

```
#undef SIMBOLICKA_KONSTANTA
```



Дефинисање симболичке константе

```
include <stdio.h>
#define WIDTH   80
int main()
{
 int x = 2, y;
 y = WIDTH * x;
 printf("Vrednost y = %d\n", y);
 #undef WIDTH
 y = WIDTH;
    /* GRESKA! 'WIDTH':undeclared identifier */
 return 0;
}
```

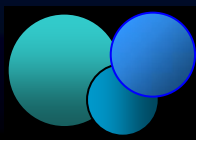
Условно превођење

Наредба **`#if, #ifdef, #ifndef...`**

- Основна наредба за почетак условног превођења :

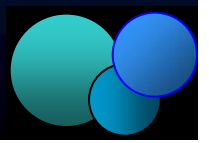
```
#if uslov
```

- Ако услов није испуњен,
  део кôда иза наредбе **`#if uslov`**
  неће бити преведен.



Условно превођење

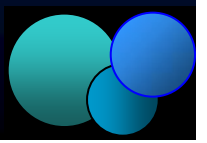
```
#define VERZIJA 1.1
#define TESTIRANJE

#ifdef TESTIRANJE
#ifndef TESTIRANJE
```



Условно превођење

Правила употребе `#if`, `#ifdef` и `#ifndef`

- Свака од ових наредби се мора завршити наредбом `#endif`.
- Између ове две наредбе може постојати произвољан број `#elif` наредби и највише једна `#else` наредба.
- Уколико постоји `#else` наредба она мора бити последња претпроцесорска наредба пре наредбе `#endif`.



Условно превођење

```
#define TST
#define DAN 2
int main()
{
 #ifdef TST
    int a = 5;
 #endif
 #if (DAN == 1)
  printf("Ponedeljak\n");
 #elif (DAN == 2)
  printf("Utorak\n");
 #else
  printf("Nije pon. ni
                 utorak\n");
 #endif
```

```
 #ifdef TST
   a = a + 7;
   printf("a = %d\n",a);
 #endif
 printf("DAN je %d",DAN);

 return 0;
}
```

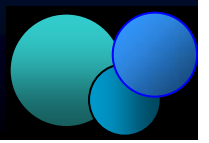
```
Utorak
a = 12
DAN je 2
```

Условно превођење

```
#define DAN 2
int main()
{
 #ifdef TST
    int a=5;
 #endif
 #if(DAN = = 1)
   printf("Poned.\n");
 #elif(DAN = = 2)
   printf("Utorak\n");
 #else
   printf("Nije poned.
          ni utorak\n");
 #endif
```

```
 #ifdef TST
   a=a+7;
   printf("a= %d\n",a);
 #endif
 printf("DAN je %d" ,DAN);
 return 0;
}
```

```
Utorak
DAN je 2
```



Хвала на пажњи

Питања?