Универзитет “Св. Кирил и Методиј” во Скопје
Факултет за електротехника и информациски технологии

ПРОГРАМИРАЊЕ И АЛГОРИТМИ

ФУНКЦИИ

- Програмски јазик C –

НОВ ПРОБЛЕМ

Проблем: Да внесат три броеви од тастатура и да се пресмета **производот** од нивните **факториели**.

a! * b! * c!

a! = 1*2*3*…*a

Чекори:

1. Внеси ги трите броеви a, b и c
2. Пресметај факториел на a
3. Пресметај факториел на b
4. Пресметај факториел на c
5. Пресметај го производот на факториелите
6. Прикажи го производот
7. Крај

НОВ ПРОБЛЕМ

Проблем: Да внесат три броеви од тастатура и да се пресмета производот од нивните факториели.

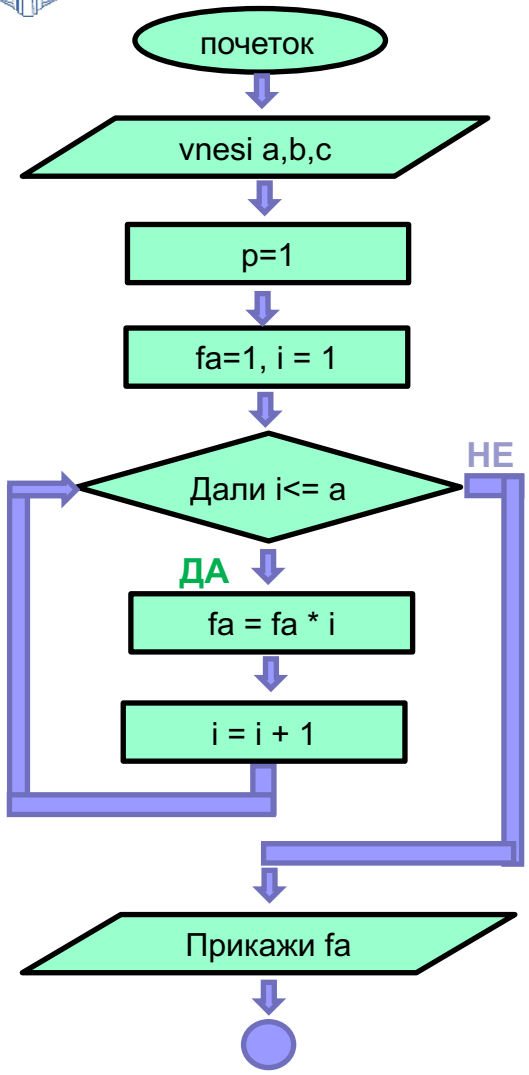
a! * b! * c!

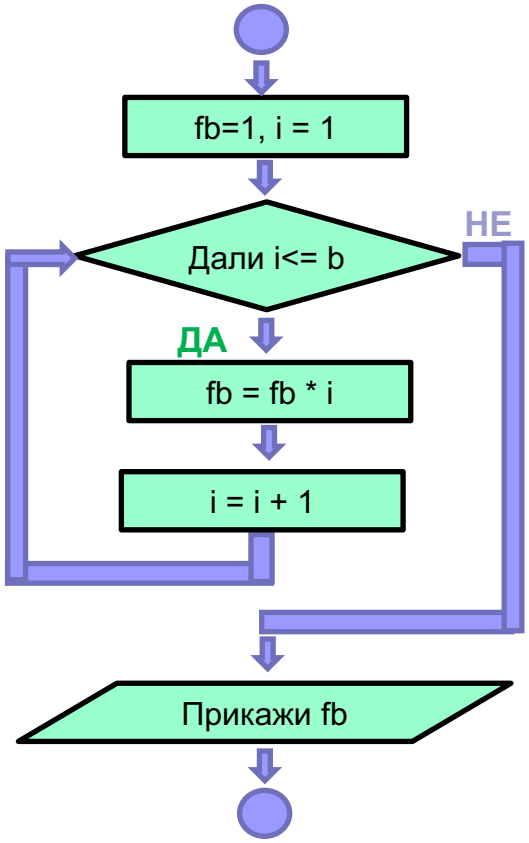
a! = 1*2*3*…*a

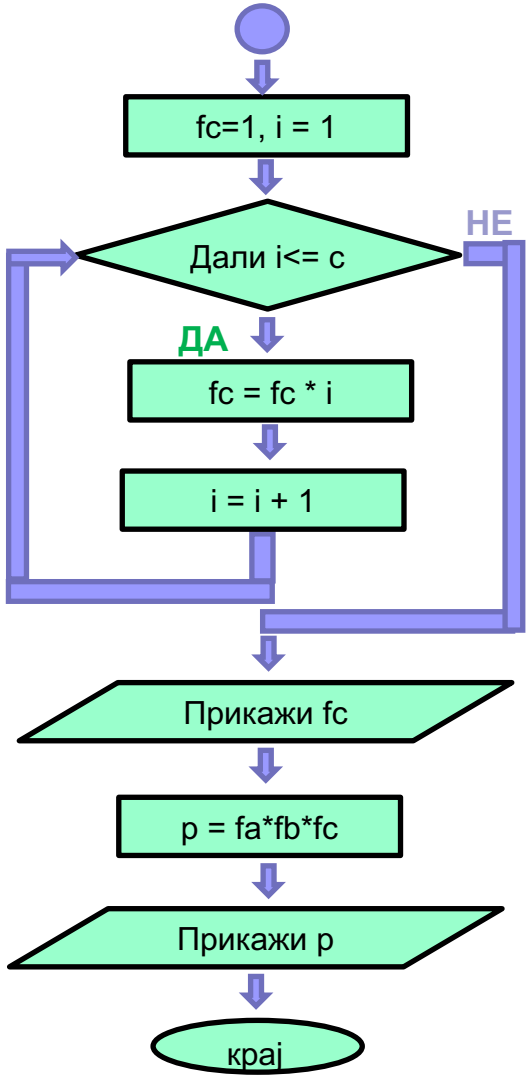
Пресметај факториел на а:

1. Постави fa=1, i=1
2. Се додека i<=a оди на чекор 3, инаку оди на чекор 6 (крај на вртење)
3. fa=fa*i
4. Зголеми го i за еден
5. Оди на чекор 2
6. Прикажи го факториелот (fa)

АЛГОРИТАМ


почеток
vnesi a,b,c
p=1
fa=1, i = 1
Дали i<= a
НЕ
ДА
fa = fa * i
i = i + 1
Прикажи fa

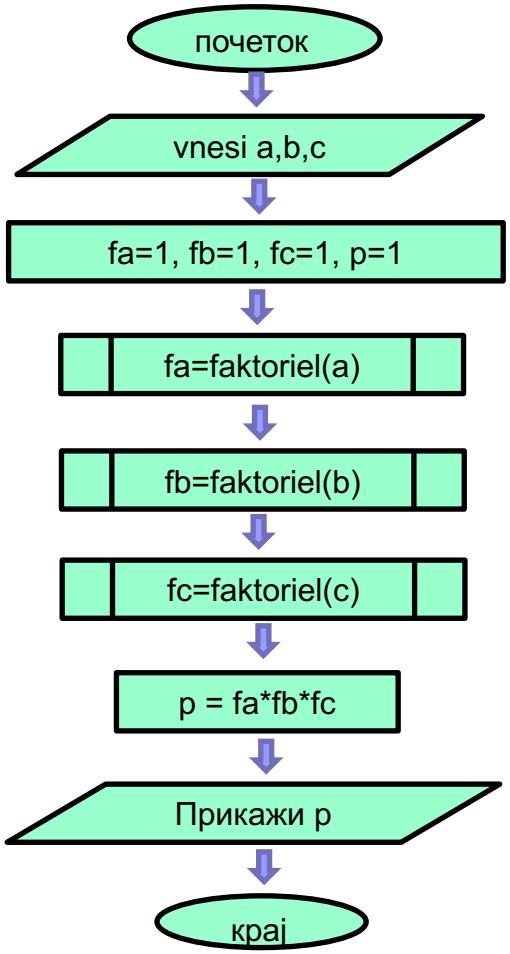

fb=1, i = 1
Дали i<= b
НЕ
ДА
fb = fb * i
i = i + 1
Прикажи fb

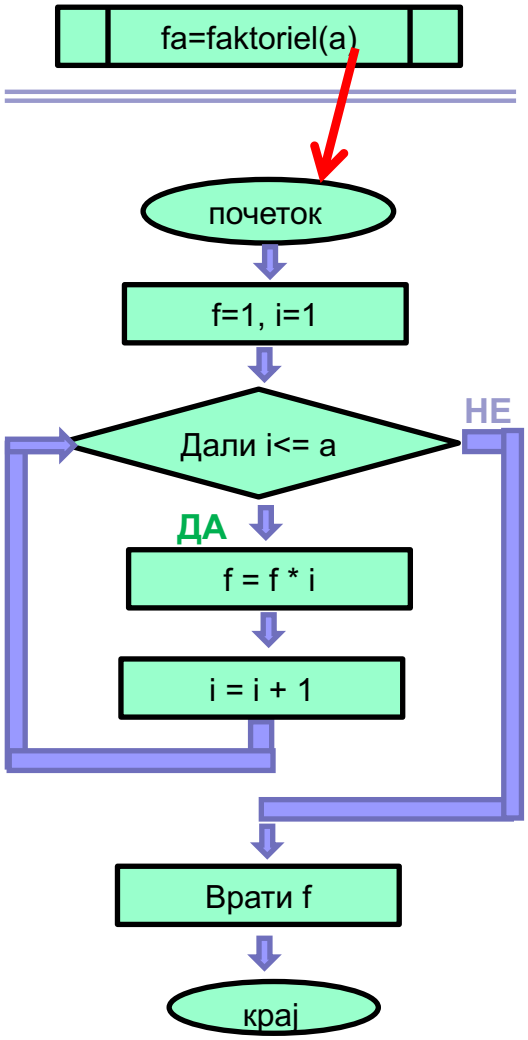

fc=1, i = 1
Дали i<= c
НЕ
ДА
fc = fc * i
i = i + 1
Прикажи fc
p = fa*fb*fc
Прикажи p
крај

ПРОГРАМА

```c
#include <stdio.h>
int main() {
    int a,b,c,fa=1,fb=1,fc=1,i,p=1;
    printf ("Vnesete gi za koi tri broevi
da se presmetuva faktoriel (a, b i c):\n");
    scanf( "%d %d %d",&a,&b,&c);
    i=1;
    while (i<=a)  {
         fa=fa*i; //fa*=i;
         i++;
    }
    printf ("Faktoriel za a = %d\n", fa);

    i=1;
    while (i<=b) {
         fb=fb*i; //fb*=i;
         i++;
    }
    printf ("Faktoriel za b = %d\n", fb);
    i=1;
    while (i<=c) {
         fc=fc*i; //fc*=i;
         i++;
    }
    printf ("Faktoriel za c = %d\n", fc);

    p=fa*fb*fc;

    printf ("Proizvod fa*fb*fc = %d\n", p);

    return 0;
}
```

ИЛИ


почеток
vnesi a,b,c
fa=1, fb=1, fc=1, p=1
fa=faktoriel(a)
fb=faktoriel(b)
fc=faktoriel(c)
p = fa*fb*fc
Прикажи p
крај


fa=faktoriel(a)
почеток
f=1, i=1
Дали i<= a
НЕ
ДА
f = f * i
i = i + 1
Врати f
крај

МОДУЛАРНО ПРОГРАМИРАЊЕ

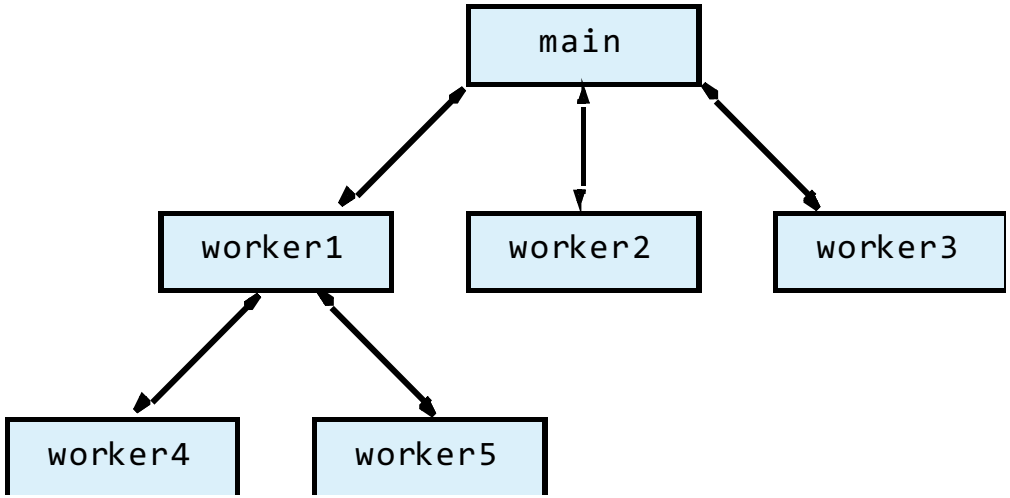
- Поголем проблем да се подели на помали делови
  - ‘модули’, или
  - ‘потпрограми’, или
  - ‘процедури’, или
  - ***функции***
- Зошто?
  - Поедноставно управување со комплексни програми
    - Помали блокови код
    - Полесно за читање
  - Повторна употреба на код
  - Независно развивање на код
  - Обезбедува ниво на ‘апстракција’
  - Се одбегнува повторување на ист код на различни места

```
a = sqrt(9.0);
```

ФУНКЦИИ

- ‘Градбени блокови’ на една C програма
  - Користевте предефинирани функции:
    - main()
    - printf(), scanf(), pow(), sqr(), sqrt()
  - Кориснички-дефинирани функции
    - Ваш код
    - Во комбинација со предефинирани функции

Аналогија ШЕФ - РАБОТНИЦИ



- Шеф задава задача
  - Работник добива информации, работи, враќа резултат
- Криење информации
  - Шефот не ги знае деталите

ФУНКЦИИ – Математички поглед

$$f(x) \overset{\Delta}{=} x^2 + 2x + 3$$

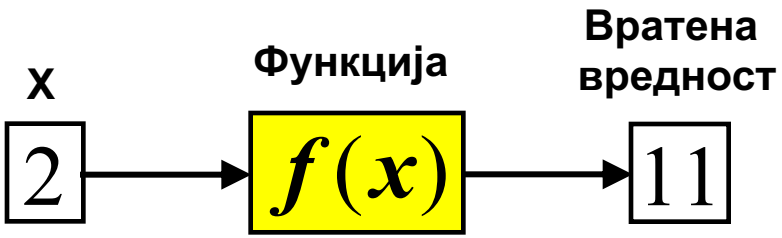
Дефиниција

Што е f(2)?

Повик

$$f(2) \Rightarrow (2)^2 + 2(2) + 3 \Rightarrow 4 + 4 + 3 \Rightarrow 11$$

$$\therefore f(2) \text{ е } 11$$



Предефинирани библиотеки - математичка библиотека -

- Библиотека математички функции
  - `#include <math.h>`
- Сите математички функции враќаат вредност од тип `double`

Function	Description	Example
`sqrt( x )`	square root of x	`sqrt( 900.0 ) is 30.0` `sqrt( 9.0 ) is 3.0`
`exp( x )`	exponential function e^x	`exp( 1.0 ) is 2.718282` `exp( 2.0 ) is 7.389056`
`log( x )`	natural logarithm of x (base e)	`log( 2.718282 ) is 1.0` `log( 7.389056 ) is 2.0`

`log10( x )`	logarithm of x (base 10)	`log10( 1.0 ) is 0.0`
		`log10( 10.0 ) is 1.0`
		`log10( 100.0 ) is 2.0`

`fabs( x )`	absolute value of x	`fabs( 5.0 ) is 5.0`
		`fabs( 0.0 ) is 0.0`
		`fabs( -5.0 ) is 5.0`

`ceil( x )`	rounds x to the smallest integer not less than x	`ceil( 9.2 ) is 10.0` `ceil( -9.8 ) is -9.0`
`floor( x )`	rounds x to the largest integer not greater than x	`floor( 9.2 ) is 9.0` `floor( -9.8 ) is -10.0`
`pow( x, y )`	x raised to power y (x^y)	`pow( 2, 7 ) is 128.0` `pow( 9, .5 ) is 3.0`
`fmod( x, y )`	remainder of x/y as a floating point number	`fmod( 13.657, 2.333 ) is 1.992`
`sin( x )`	trigonometric sine of x (x in radians)	`sin( 0.0 ) is 0.0`
`cos( x )`	trigonometric cosine of x (x in radians)	`cos( 0.0 ) is 1.0`
`tan( x )`	trigonometric tangent of x (x in radians)	`tan( 0.0 ) is 0.0`

КОРИСТЕЊЕ ФУНКЦИИ

- `printf( "%.2f", sqrt( 900.0 ) );`
  - `sin(a);`
  - `printf("ova e primer br.%d", broj);`
  - `scanf("%d",&broj);`
- Користење = повик на функции
  - `ImeFunkcija( argument );`
  - Се дава име на функција и аргументи (податоци)
  - Повеќе аргументи одделени со запирка
  - Аргументи
    - Константи
    - Променливи
    - Изрази
  - Функција извршува операции
  - Функција враќа резултат

ДЕФИНИРАЊЕ на ФУНКЦИИ

- Заглавје функција
  - Тип на вратена вредност (ако има)
  - Име на функција
    - Име на прог. елем.
  - Аргументи (или листа на параметри)
    - tip и ime
- Наредби
  - Декларација променливи
    - Локални за функцијата
  - Операции
  - Вратена вредност (ако има)

```
tip ime_funkcija (tip arg1, tip arg2, … )
{
      naredbi;
}
```

Функција што пресметува *производ* на два броја

```
double product(double x, double y)
{
   double result;

   result = x * y;
   return result;
}
```

АРГУМЕНТИ

- **tip arg1, tip arg2, …**
- Листа (одделени со запирки)

- Како декларација на локални променливи
  - Може да се употребат само во телото на функцијата
- Може да не содржи аргументи: ()
- теориски бројот на аргументи не е ограничен
  - во пракса е ограничен

Препорака: одбегнувајте користење на голем број аргументи

- во K&R стилот заглавието на функцијата се дефинира

```
tipVratenaVrednost  imeFunkcija(Pr1, Pr2, ... )
  tip Pr1;
  tip Pr2; ...
{ telo na funkcijata  }
```

Вредност што ја враќа функцијата

- тип на вредноста
  - `void` – функцијата не враќа вредност
  - ако не е дефиниран типот, се смета дека е `int`

- Пример:

```
int praviNesto(){...}
```

е исто со

```
praviNesto(){...}
```

Препорака: За да се одбегне забуна, секогаш декларирајте го типот.

Вредност што ја враќа функцијата (2)

- Ако функцијата не враќа вредност => се излегува со:
  - `return;`

  или со
  - `}`

```
void print_answer(int answer) {
        if (answer < 0) {
                printf("Answer corrupt\n");
                return;
        }
        printf("The answer is %d\n", answer);
}
```

- Ако функцијата враќа вредност => се излегува со:
  - `return` *expression;*
  - **Типот на вредност** мора да соодветствува на дефиницијата

```
float add_numbers( float n1, float n2 ) {
        return n1+n2;           /* ispravno, zbirot e od
                                   tipot float */
}
```

Вредност што ја враќа функцијата (3)

```
int add_numbers( float n1, float n2 )  {
    return 6.0;              /* ne e vo red */
}
```

Компајлерот нема да јави грешка, ќе си направи **cast**

```
float add_numbers( float n1, float n2 ){
    return 6.0;              /* ispravno, vraka
                             realna vrednost */
}
```

Вредност што ја враќа функцијата (4)

- Можно е една функција да има повеќе наредби за враќање на вредност:
- Се извршува САМО ЕДНА
- Во тој момент завршува функцијата
- Функција враќа САМО ЕДНА вредност

```
int validate_input( char command )    {
   switch( command ) {
         case '+' :case '-' : return 1;
         case '*' :case '/' : return 2;
         default  : return 0;
   }
}
```

Дефинирање и користење функции

```
#include <stdio.h>
float triangle(float width, float height) {
        float area;
        area = width * height / 2.0;
        return (area);
}
int main() {
        float size;
        size = triangle(1.3, 8.3);
        printf("Triangle #1 %f\n", size);
        printf("Triangle #2 %f\n", size = triangle(4.8, 9.8));
        printf("Triangle #3 %f\n", size = triangle(1.2, 2.0));
        return 0;
}
```

Изразот

```
size = triangle(1.3, 8.3);
```

е повик кон функцијата.

- На променливата `width` ќе и биде придружена вредност 1.3
- На `height` 8.3.
- Функцијата ја враќа вредноста 5.4, која се сместува во променливата `size`.

Дефинирање и користење функции

```
#include <stdio.h>
int multbytwo(int x) {
        return x * 2;
}
int main() {
        int i, j;
        i = 3;
        j = multbytwo(i);
        printf("%d\n", j);
        return 0;
}
```

Линијата

```
j = multbytwo(i);
```

е повик кон функцијата.

- На променливата `x` ќе и биде придружена вредност 3.
- Функцијата ја враќа вредноста 6, која се сместува во променливата `j`.

ФУНКЦИСКИ ПРОТОТИПОВИ

- Декларација пред дефиниција
- Потребен само ако употребата на функцијата е пред дефиницијата во програмата
- Формат

tip imeFunkcija(listaParametri);

- Исто со дефиниција
- Завршува со ;
- Валидација на функција
- Не е задолжително да се наведат имињата на аргументите

Прототипот

```
int maximum( int x, int y, int z );
```

- Прима 3 `int` вредности
- Враќа `int`

ФУНКЦИСКИ ПРОТОТИПОВИ

- Примери:

```
float triangle (float width, float height);

int prim(void);

void prim1(void);

void prim(int, float);

float triangle(float, float);

int maximum( int, int, int );
```

- се проследуваат 3 целобројни вредности, враќа целобројна вредност

ФУНКЦИСКИ ПРОТОТИПОВИ

Пример за грешка:

```
square (float x)      {
        float y;
        y = x * x;
        return (y);
}
int main ( ) {
        float a, b;
        printf ("\n Vnesi broj");
        scanf ("%f", &a );
        b = square (a);
        printf ("\n kvadratot na %f e %f", a, b);
        return 0;
}
```

Default return type is ***int***

```
Vnesi broj 2.5
Kvadratot na 2.5 e 6.000000
```

ФУНКЦИСКИ ПРОТОТИПОВИ

Пример за коректна програма:

```
int main ( ) {
        float square (float);
        float a, b;
        printf ("\n Vnesi broj ");
        scanf ("%f", &a);
        b = square (a);
        printf ("\n kvadratot na % f e % f", a, b);
        return 0;
}
float square (float x) {
        float y;
        y = x * x;
        return y;
}
```

```
Vnesi broj 2.5
Kvadratot na 2.5 e 6.2500000
```

ФУНКЦИСКИ ПРОТОТИПОВИ

Пример за коректна програма:

```
/* Programata ilustrira ednostaven
funkciski povik */
#include <stdio.h>
void print_message( void );

void print_message( void ) {
      printf("Ova e funkcija narecena print_message\n");
}

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

```
Ova e funkcija narecena print_message
```

ФУНКЦИСКИ ПРОТОТИПОВИ

```c
/* Programa za presmetuvanje faktoriel  */
#include <stdio.h>
void calc_factorial( int ); /* vo ovoj primer ne e neophodno */
void calc_factorial( int n ) {
        int  i, fact_num = 1;
        for( i = 1; i <= n; ++i )
                fact_num *= i;
        printf("Faktoriel od %d iznesuva %d\n", n, fact_num);
}
int main() {
        int  number = 0;
        printf("Vnesi broj\n");
        scanf("%d", &number );
        calc_factorial( number );
        return 0;
}
```

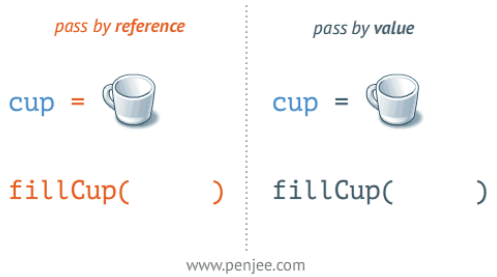
```
Vnesi broj
3
Faktoriel od 3 iznesuva 6
```

ФУНКЦИСКИ ПРОТОТИПОВИ - Пример

```
/* Programa za opredeluvanje na najgolemiot od tri broja */
#include <stdio.h>
int maximum( int, int, int );   /* funkciski prototip */
int main() {
        int  a,b,c;
        printf("Vnesi tri celi broevi: \t");
        scanf("%d%d%d", &a, &b, &c);
        printf("Najgolemiot e: %d \n",maximum(a,b,c));
        return 0;
}
/* Definicija na funkcijata */

int maximum( int x, int y, int z ) {
   int max = x;
   if ( y > max )
       max=y;
   if ( z > max )
       max=z;
   return max;
}
```

```
Vnesi tri celi broevi: 22 85 17
Najgolemiot e: 85
```

Пренос на вредности во функција



- ***Повик по вредност***
  - **Копија** од аргументот се пренсува во функцијата
  - Промени во функција не влијаат на оригиналот
  - Функцијата не го менува аргументот: се избегнуваат несакани промени
- ***Повик по референца***
  - Се праќа **оригиналниот** аргумент
  - Промени во функцијата влијаат на оригиналот
- Засега: **повик по вредност** **!**
- **Вредноста на секој аргумент** при повикувањето на функцијата се сместува **во соодветниот формален аргумент**.

Пренос на вредности во функција (2)

пример:

```
int funkcija (int x, int y) {
        x = 30;
        y = 40;
        return 0;
}
int main ( ) {
        int x = 10, y =20;
        printf ("\n x = %d y = %d", x, y);
        funkcija (x, y);
        printf ("\n x = %d y = %d", x, y);
}
```

x = 10 y =20

x = 10 y =20

Пренос на вредности во функција (2)

пример:

```
int swapv (int x, int y) {
        int t;
        t = x; x = y; y = t;
        printf ("\n x = %d y = %d", x, y);
        return 0;
}
int main ( ) {
        int a = 10, b =20;
        printf ("\n a = %d b = %d", a, b);
        swapv (a, b);
        printf ("\n a = %d b = %d", a, b);
}
```

a = 10 b =20

x = 20 y =10

a = 10 b =20

Области на важење на променливите

- Променливите се разликуваат според типот, областа на важење и начинот на креирање.
- Според областа на важење на дефиницијата:
  - Глобални променливи
  - Локални променливи

Области на важење на променливите (2)

- Локални променливи
  - Постојат само во рамките на функциите/блокот во кои се креирани.
  - Непознати за сите останати функции и главната програма
  - Се уништуваат кога ќе се напушти функцијата во која се креирани.
  - Повторно се креираат при секој следен повик на функцијата.
- Глобални променливи
  - Се дефинираат надвор од секоја функција
  - Може да ги користи секоја функција во програмата
  - Не се креираат повторно при секое следно повикување на која и да е функција

Дефинирање на глобални променливи

Пример 1:

```
#include <stdio.h>
int add_numbers( void );
int value1, value2, value3;
int add_numbers( void ) {
        auto int result;
        result = value1 + value2 + value3;
        return result;
}
int main() {
        auto int result;
        value1 = 10;
        value2 = 20;
        value3 = 30;
        result = add_numbers();
        printf("%d + %d + %d = %d\n", value1, value2, value3,
result);
}
```

10 + 20 + 30 = 60

Дефинирање на глобални променливи (2)

Пример 2:

Областа на важење на глобалните променливи може да се ограничи со внимателно поставување на декларациите на променливите во датотеката.

```
#include <stdio.h>
void no_access( void );
void all_access( void );
int n2;   /*n2 e poznata od ovaa tocka*/
void no_access( void ) {
        n1 = 10;   /* netocno, n1 ne e deklarirana, i kompajlerot
                      pri preveduvanje na programata vo ovaa tockake javi
                      greska*/
        n2 = 5;    /* tocno */
}
int n1;   /* n1 e deklarirana i vazi od
             ovaa tocka vo ostatokot od programata */
void all_access( void ) {
        n1 = 10;    /* tocno */
        n2 = 3;     /* tocno */
}
```

Дефинирање на глобални променливи (3)

```c
#include <stdio.h>
void prikazi();
int x = 20;
int main() {
    printf("%d vo glavnata programa\n", x);
    prikazi();
    return 0;
}
void prikazi() {
    printf("%d vo funkcija\n", x);
}
```

20 vo glavnata programa
20 vo funkcija

```c
#include <stdio.h>
void prikazi();
int x = 20;
int main() {
     int x =10;
     printf("%d vo glavnata programa\n", x);
     prikazi();
}
void prikazi() {
     printf("%d vo funkcija\n", x);
}
```

10 vo glavnata programa
20 vo funkcija

Дефинирање на локални променливи

Локални променливи може да се декларираат и за секој блок наредби и за нив важат истите правила - променливите се креираат при секое повторно влегување во блокот и важат за блокот и сите останати блокови вгнездени во истиот.

```
for(i=0; i<10; i++) {
        float x=0.0;
        ...
}
```

`x` ќе важи само во рамките на `for` блокот и тоа при секое повторување на циклусот: на почетокот ќе се резервира простор за истата и ќе се иницијализира, а на крајот се уништува.

```
int main() {
        int x = 1;
        {
                int x = 2;
                {
                        int x = 3; printf("x=%d ", x);
                }
                printf("x=%d ", x);
        }
        printf("x=%d\n", x);
        return 0;
}
```

x=3 x=2 x=1

Следнава програма е погрешна. Зошто?

```
#include <stdio.h>

void prikazi();
int main() {
        printf("%d vo glavnata programa\n", x);
        prikazi();
}
int x = 20;
void prikazi() {
        printf("%d vo funkcija\n", x);
}
```

Типови променливи според начинот на креирање

- Постојани или привремени променливи
- Постојани
  - се креираат и иницијализираат на почетокот на секоја програма
  - постојат се до завршувањето на програмата
  - најчесто глобални променливи
- Привремени
  - се креираат и иницијализираат при извршувањето на блокот во кој се декларирани
  - се уништуваат при напуштање на блокот во кој се декларирани
  - најчесто локални променливи
- Следните клучни зборови се користат за да се опише кога и каде променливите ќе се креираат и уништат
  - auto
  - static
  - extern
  - register

Auto и static променливи

- `static` променливи
  - се креираат и иницијализираат еднаш, на првиот повик на функцијата
  - При секој следен повик на функцијата не се креираат ниту пак се реиницијализираат статичките променливи.
  - Кога ќе заврши функцијата променливите сé уште постојат, но не може да им се пристапи од другите функции
- `auto` променливи
  - Сосема спротивно
  - Се креираат при повик на функцијата
  - Се уништуваат при напуштање на истата

Auto и static променливи (2)

```
#include <stdio.h>
void demo( void );
void demo( void ) {
      auto int avar = 0;
      static int svar = 0;
      printf("auto = %d, static = %d\n", avar, svar);
      ++avar; ++svar;
}
int main() {
      int i=0;
      while( i < 3 ) { demo(); i++; }
      return 0;
}
```

i=0	auto = 0, static = 0
i=1	auto = 0, static = 1
i=2	auto = 0, static = 2

Локални променливи декларирани како **static**

```
#include <stdio.h>
int main() {
      int counter;      /* loop counter */
      for (counter = 0; counter < 3; ++counter) {
            int temporary = 1;                  /* A temporary variable */
            static int permanent = 1; /* A permanent variable */
            printf("Temporary %d Permanent %d\n", temporary, permanent);
            ++temporary;
            ++permanent;
      }
      return 0;
}
```

```
Temporary 1 Permanent 1
Temporary 1 Permanent 2
Temporary 1 Permanent 3
```

- Привремените променливи се нарекуваат *automatic* променливи бидејќи просторот за истите се резервира автоматски.
- Квалификаторот `auto` може да се користи за да се означат овој вид на променливи иако тоа во практиката многу ретко се случува.

Програмирање и алгоритми

РЕКУРЗИЈА

Рекурзија и рекурзивни функции

- **Функции што се повикуваат самите себе**
- Секоја рекурзивна функција мора да следи две правила:
  - мора да има услов за прекин (дефиниран основен случај)
  - описот мора да го прави проблемот поедноставен

Пример: Пресметување факториел

5! = 5 * 4 * 3 * 2 * 1

Начин на пресметување:

5! = 5 * 4! (поедноставување на проблемот)

4! = 4 * 3! ...

По дефиниција 1! = 0! = 1 (основен случај)

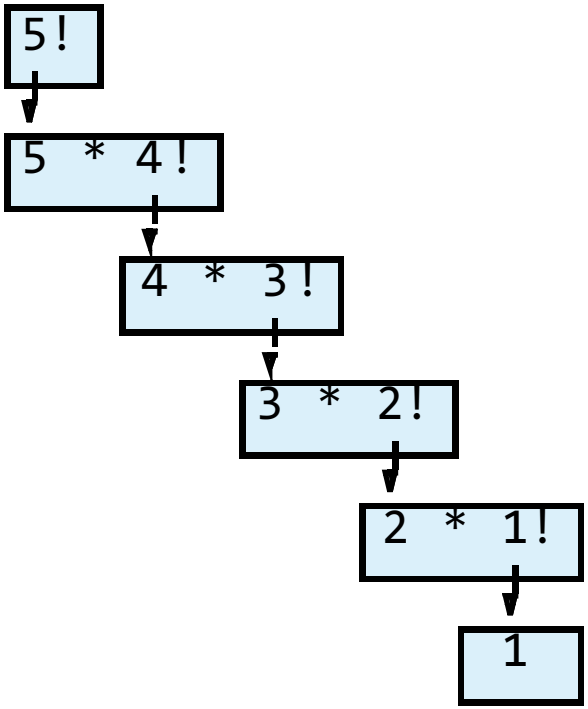
Тогаш важи

2! = 2 * 1! = 2 * 1 = 2;

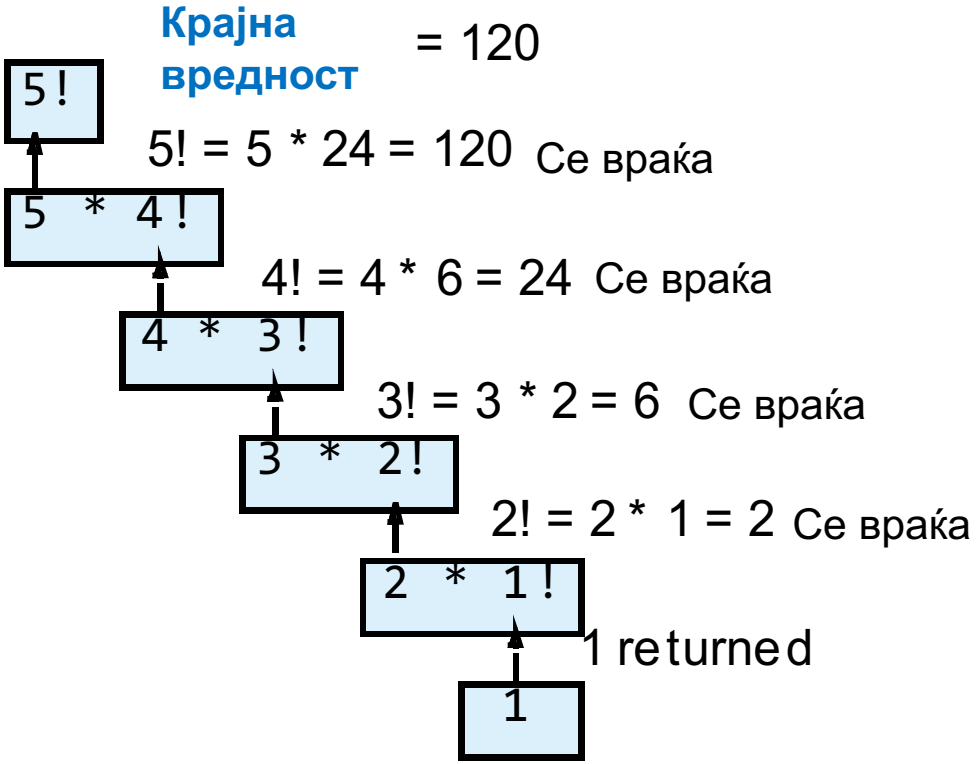
3! = 3 * 2! = 3 * 2 = 6;

```
int fact(int n){
   if(n==0)return 1;
   else
        return n*fact(n-1);
}
```

Рекурзија и рекурзивни функции (2)



а) Секвенца од рекурзивни повици



б) Вредности вратени од секој рекурзивен повик

Факториел

```c
#include <stdio.h>
int fact (int n)
{
    if (n >= 1)
        return n*fact(n-1);
    else
        return 1;
}
int main()
{
    int n;
    scanf("%d", &n);
    printf("Factorial of %d = %d", n, fact(n));
    return 0;
}
```

```c
int fact(int n){
   if(n==0)return 1;
   else
       return n*fact(n-1);
}
```

Фибоначиева низа броеви со рекурзија

fib(0) fib(1) fib(2) fib(3) fib(4) fib(5) fib(6)

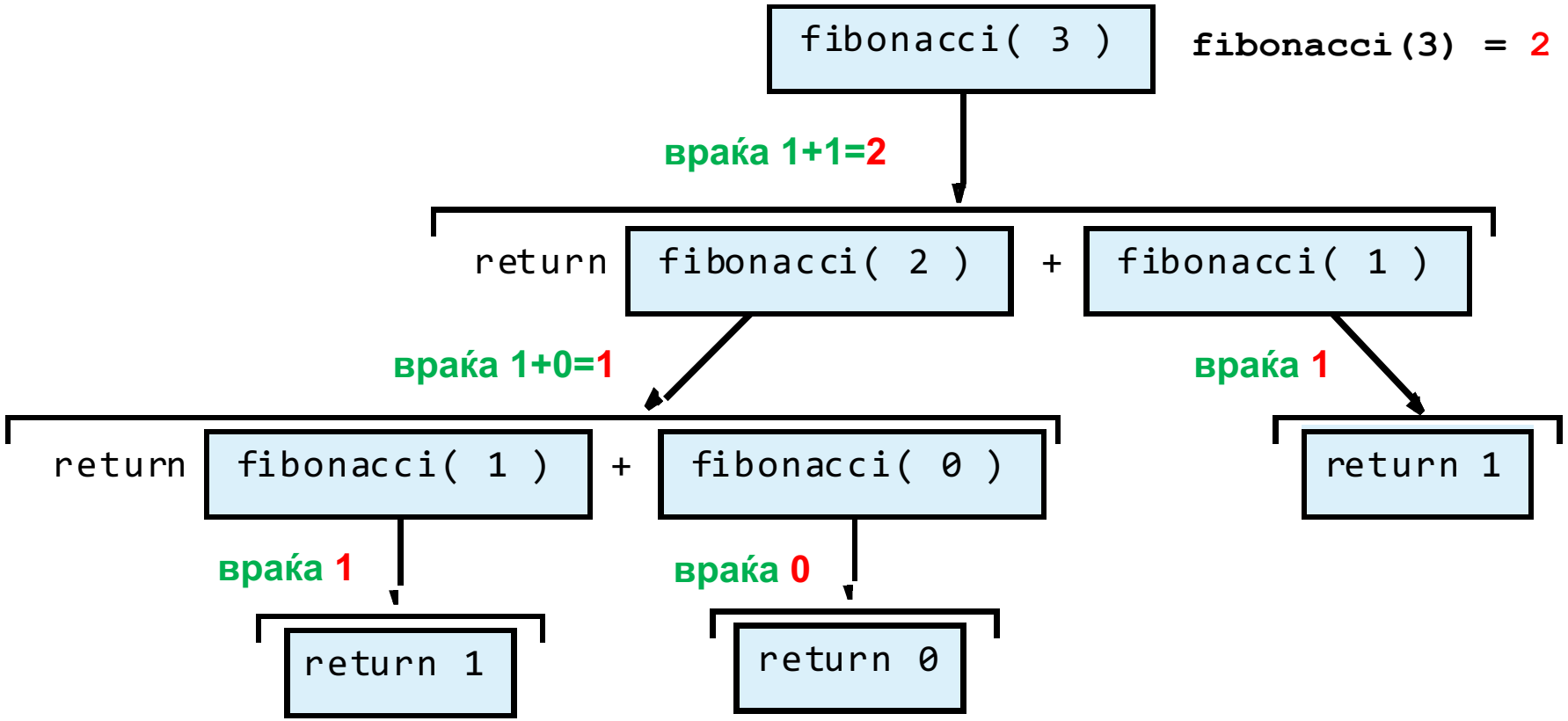
- Фибоначиева низа: 0, 1, 1, 2, 3, 5, 8...

  секој број се добива како збир на претходните два

  fib(n) = fib(n-1) + fib(n-2) – рекурзивна формула

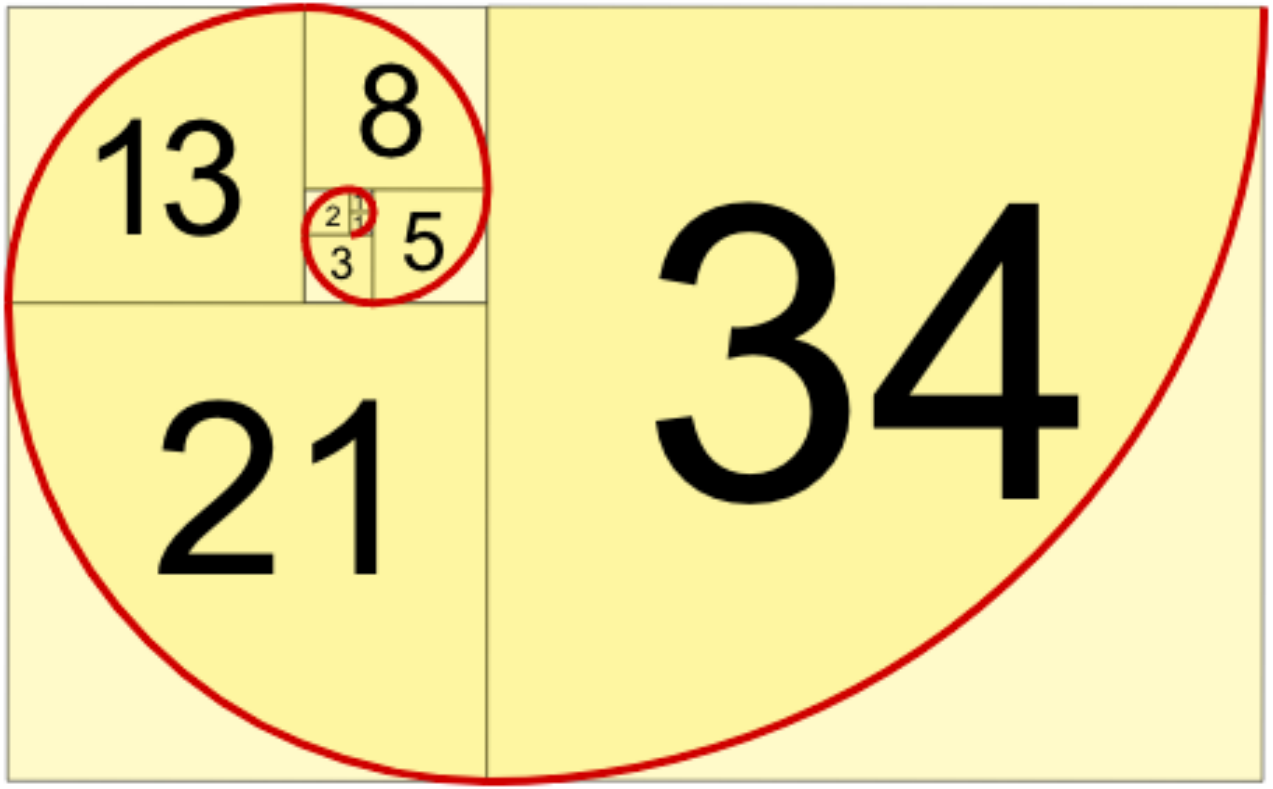
```
long fibonacci(long n) {
    if (n==0 || n==1)  /* osnoven slucaj */
        return n;
    else return fibonacci(n-1) + fibonacci(n-2);
}
```

Начин на извршување на функцијата за пресметка на Фибоначиева низа



Секвенцата на Фибоначи

- Должината на страната на квадратите е број од секвенцата на Фибоначи

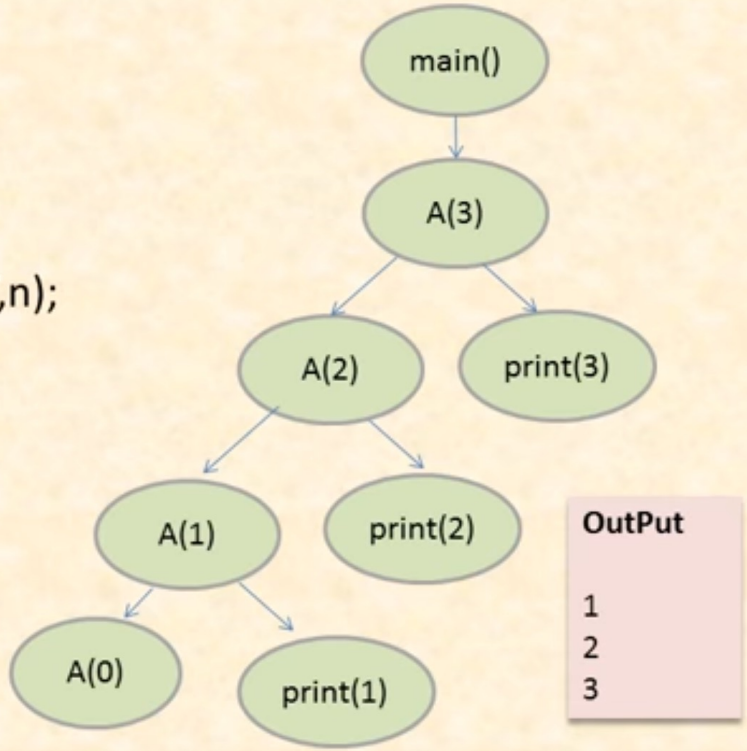


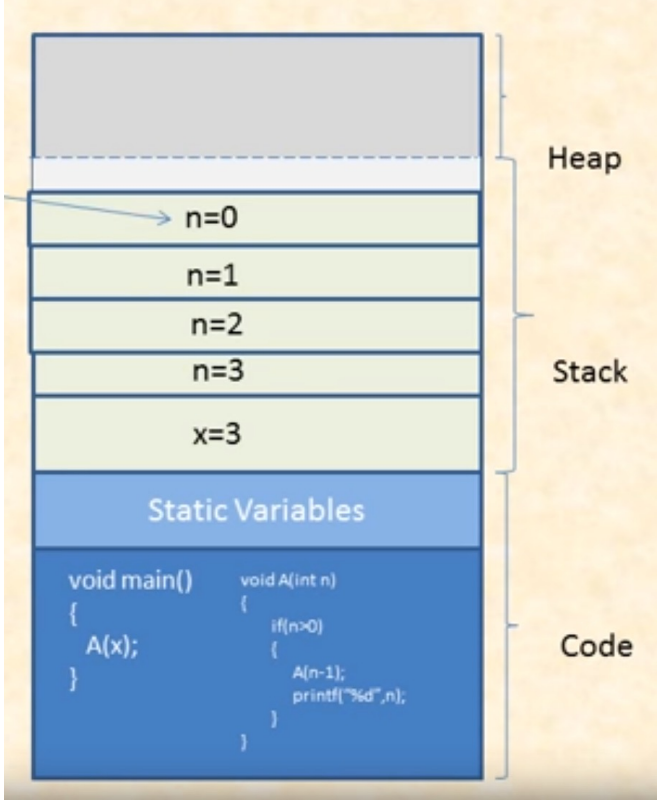
Рекурзија vs итерација

- Повторување
  - Итерација: експлицитни циклуси
  - Рекурзија: функциски повици
- Прекин на повторувањето
  - Итерација: условот за повторување повеќе не важи
  - Рекурзија: се препознава основниот случај
- И во обата случаја можни се појави на бесконечно повторување

Рекурзија и меморија

```
void A(int n)
{
    if(n>0)
    {
        A(n-1);
        printf("%d",n);
    }
}

void main()
{
        int x=3;

        A(x);
}
```





ПРИМЕР 1 (Рекурзија)

- Да се напише рекурзивна функција која како аргумент добива повеќе цифрен број и една буква. Функцијата треба на екран да ја отпечати буквата што се добива кога на внесената буква се додава цифрата од бројот. Најпрво да се почне од најлевата цифра, па потоа од таа пред неа и т.н.

```
#include <stdio.h>
void rek (int x, char c) {
        if (x/10==0)
                printf ("%c", x+c);
        else {
               rek (x/10, c);
               printf ("%c", x%10+c);
        }
}

int main (){
        rek(1234,'A');
        return 0;
}
```

На екран: BCDE

ПРИМЕР 2 (Рекурзија)

- Да се напише рекурзивна функција која ќе добие како аргументи два цели броеви и ќе изврши собирање на цифрите на двата броеви на иста позиција и ќе креира нов број. Доколку сумата на цифрите од двата броеви надминува 9, во новиот број се запишува 1, инаку се запишува 0. Рекурзијата се извршува додека има цифри и во двата броеви.

```
#include <stdio.h>
int rek (int a, int b) {
    if ((a/10==0) || (b/10==0))
        return ((a%10+b%10)>9);
    else
        return rek(a/10,b/10)*10+((a%10+b%10)>9);
}

int main (){
        int izlez;
        izlez=rek (6728,5678);
        printf("%d",izlez);
        return 0;
}
```

На екран: 1101

ПРИМЕР 3 (Рекурзија)

- Да се напише рекурзивна функција која како аргумент добива две букви. Функцијата треба да ги отпечати сите измеѓу букви помеѓу тие две букви почнувајќи од таа што е полево во азбуката кон таа што е подесно во азбуката (абецедата).

```
#include <stdio.h>
void rek (char a, char b) {
        if (a==b) printf ("%c", a);
        else {
                if (a>b) {
                        rek (a-1,b);
                        printf ("%c", a);}
                else {
                        rek (a,b-1);
                        printf ("%c", b);}
        }
}
int main (){
        rek('G','B');
        return 0;
}
```

На екран: BCDEFG

ASCII Code табела

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

КРАЈ