CODE ACADEMY

ОБУЧЕНИЕ ПО ПРОГРАМИРАНЕ

Функции - 1

Функции в С

Има две категории функции:

- **Predefined functions -** предоставени от стандартните библиотеки на С – stdio.h, math.h, и др.

- **User-defined functions -** функции, които се създават от програмистите за специални задачи.

Например

1) функциите scanf( ) printf( ) са стандартни предефинирани функции

2) В оператора **fAnswer = cube(fInput);** се извиква потребителската функция cube().

Когато се изпълни този оператор, програмата скача към дефинираната функция cube(). След като изпълнението на функцията cube() завърши, управлението се връща към програмния код, откъдето е било направено извикването, например main(). Стойността, която е върнала функцията се присвоява на променливата **fAnswer** и се използва за бъдещи изчисления.

Функции в С

Функцията има следните характеристики:

- **Име, което е уникално**

- **Изпълнява определена задача**

Задачата описва операцията, която трябва да се изпълни, като част от цялата работата на програмата.

- **Независимос**т

Функцията може да изпълнява своята задача без смущения от други части на програмата или намеса в други части на програмата.

- **Може да получава стойности при извикването си.**

Извикващата програма може да предава стойностите към функцията за изпълнение, директно или индиректно – чрез референция.

- **Може да върне стойност към извикващата я програма**.

Извиканата функция може да предаде нещо обратно на извикващата програма

CODE ACADEMY

Функции в C

Програмата не изпълнява операторите във функцията, докато функцията не бъде извикана.

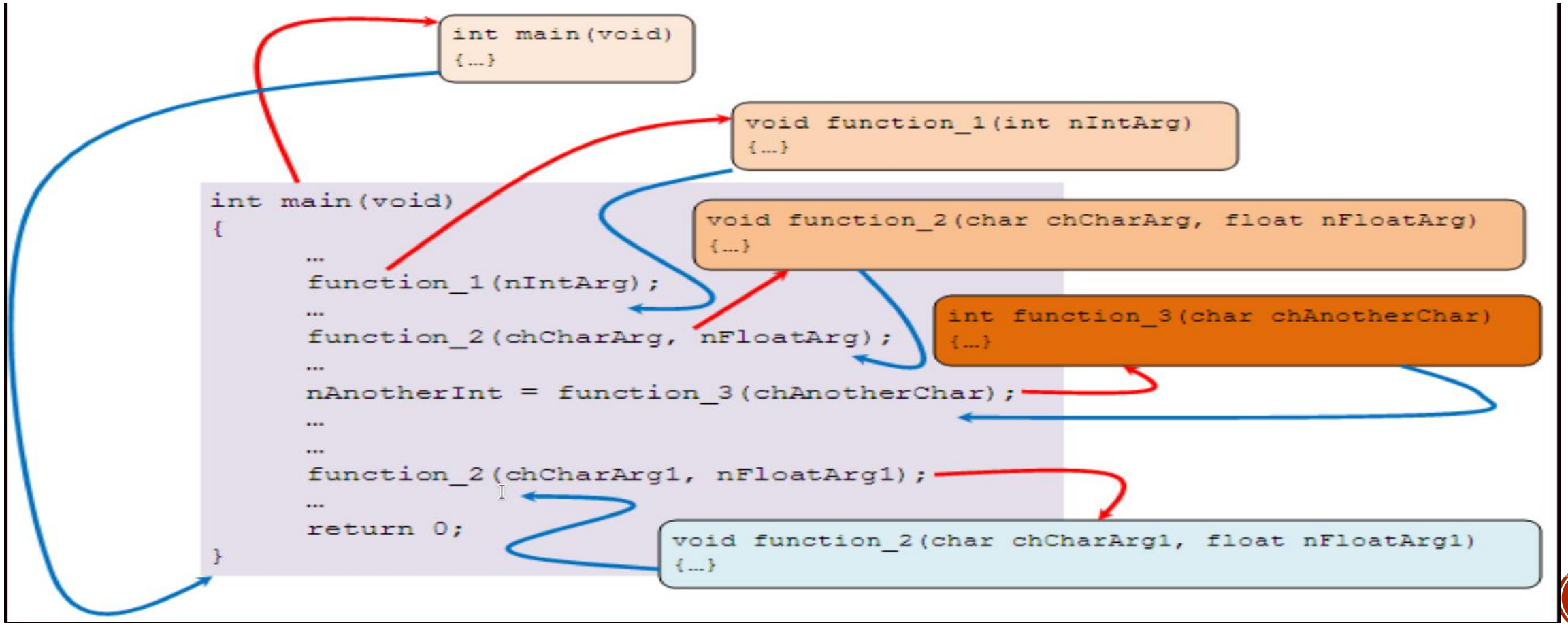
Когато функцията се извика, програмата може да изпрати информация до нея във формата на един или няколко аргумента, въпреки, че това не е задължитерно.

Аргументите са данни в програмата, които са необходими на функцията за да изпълни своята работа.

Когато функцията завърши изпълнението, програмата се връща на същото място, къде е била извикана функцията.

Функции в С

Следната фигура илюстрира извикването на функция, както и активирането на записването в стек-паметта – създаване и унищожаване.



CODE ACADEMY

Функции в С

Функциите могат да се извикат много пъти
– толкова, колкото е необходимо

Функциите могат да се извикат в произволен ред, при условие, че :

- са били заявени – като прототип

```
int readNumber( );
```

- и са определени – дефинирани

```
int readNumber( )
        {
                int n;
                scanf("%d", &n);

                return n;
        }
```

CODE ACADEMY

Въпроси?

CODE ACADEMY

Пример

```
#include<stdio.h>
/* prototype*/
        float readNumber();
        void printN();
        void printNumber(double y);

        int main()
        {
         float x=readNumber();
        printf("%f\n",x);
        printN(67);
        printNumber(67);
        }
 /* definition */
        float readNumber()
        {
        float n;
        scanf("%f",&n);
        return n;
        }
 /* definition */
        void printN(int num)
        {
        printf("%d\n", num);
        }
  /* definition */
        void printNumber(double y)
        {
        printf("%fl", y);
        }
```

CODE ACADEMY

Дефиниране на функция

Всяка функция има тяло, което съдържа код, който ще се изпълни при извикване на функцията

```
int cube(int fCubeSide) {

        // local scope  - local to this function

    int fCubeVolume;      // calculate volume

    fCubeVolume = fCubeSide * fCubeSide * fCubeSide;

        // return the result to caller

    return fCubeVolume;
}
```

CODE ACADEMY

Дефиниране на функция

Първият ред от дефиницията на функцията се нарича заглавие на функцията.
То трябва да бъде идентично с прототипа на функцията, но без ;

В заглавието на функцията задължително се посочват аргументите като променливи, въпреки, че в прототипа те не са задължителни.

Тялото на функцията съдържа оператори, които функцията ще изпълни. Тези оператори са заградени с фигурни скоби { }

Ако функцията не връща нищо, то типа на върната стойност на функцията ще бъде void.

Ако функцията връща някаква стойност, последният оператор в тялото на функцията трябва да е return.

Типа на стойността, която връща функцията трябва да съвпада типа на върнатата стойност на функцията от заглавието.

CODE ACADEMY

Заглавие на функция

Първият ред от дефиницията на функцията е заглавието на функцията и има три компонента:

1. **Тип на върната стойност на функцията –** определя какъв тип данни ще върне функцията към програмния код, който я извиква. Това може да бъде всеки тип данни в C - char, float, int, long, double, pointers и др. Ако не се определи тип на върната стойност, типът е void.

   int calculate_yield
   float mark(…)
   void calculate_interest(…)

2. **Име на функцията** - Може да бъде име на идентификатор, според правилата на езика C. Това име трябва
   - да отговаря на предназначението на функцията
   - да бъде уникално.

Заглавие на функция

3. **Списък с параметри –** много функции използват аргументи - стойности, предавани на функцията, когато е извикана. **Функцията трябва да знае типа на данните на всеки аргумент**. Типът на аргументите се предоставя в заглавието на функцията чрез списък от параметри. Списъкът от параметри действа като placeholder.

   За всеки аргумент, който се предава на функцията, **списъкът с параметри трябва да съдържа един запис, който определя типа и името**.

   void **myfunction**(int x, float y, char z)

   void **yourfunction**(float myfloat, char mychar)

   int **ourfunction**(long size)

   Някои функции не приемат аргументи и за тях списъкът с параметри ще бъде празен или може да се запише void.

CODE ACADEMY

Списък с параметри

Параметрите са вътре в заглавието на функцията.

Те служат за placeholder за аргумента.

Те не се изменят по време на изпълнение на функцията

Аргументът е действителната стойност, предадена на функцията от извикващата програма.

Всеки път, когато се извиква функция, тя може да се извика с различни аргументи.

**При всяко извикване на функцията трябва
да се предават един и същ брой и тип аргументи, като стойностите на аргументите могат да бъдат различни.**

Списък с параметри -2

Нека функцията е float halfOf (float p)

При първото извикване на функцията : z=halfOf(x); стойността на p става стойносста на x.

При второто извикване на функцията : z=halfOf(y); стойността на p става стойносста на y.

Всеки път, когато се извика функцията се предават различни аргументи на параметрите и.

Стойностите на x и y се предават като се копират в параметъра p на функцията halfOf().

След това half_of() връща половината на тази стойност.

Тяло на функцията

Оградено е в къдрави скоби { } , непосредствено след заглавието на функцията.

- **Тук се извършва реална работа на кода на функцията.**

- **Когато функцията е извикана,**
  изпълнението започва в началото на тялото на функцията и
  завършва, когато е изпълнен оператор за връщане или при достигане на }.

- **В тялото на функция може да бъде направена декларацията на променлива.**

- **Локални променливи са променливи, които се дефинират и използват само в тази функция.**

- **Локалните променливи са променливите, са различни от другите променливи със същото име (ако има такива), декларирани другаде в програмата извън функцията.**

- **Декларирането, инициализирането и използването на локалните променливи става както това на всяка друга променлива.**

- **Променливи, които са дефинирани извън всяка функция се наричат глобални променливи.**

Разположение на функциите в програмата

Параметрите на функциите се разглеждат като декларации на променливи

Обикновено, функционалните прототипи се поставят преди main() функцията.

Ако има две потребителски функции и едната от тях използва другата, то функцията, която се използва вътре в другата функция, трябва да се дефинира преди това.

Възможно е и директно да декларираме и едновременно с това да дефинираме функция.

```
#include …
/* function prototype */
int funct1(int);

int main() {
/* function call */

 int y = funct1(3); …
 }
 /* Function definition */
 int funct1(int x){  …  }
```

```
#include …
/* declare and define */
 int funct1(int x) { … }

int main() {
 /* function call */

 int y = funct1(3); …
 }
```

CODE ACADEMY

Използването на променливи във функциите

Три правила управляват използването на променливи във функциите:

1. **За да използваме променлива във функция, трябва да я декларираме в заглавието на функцията като параметър или във тялото на функцията като локална променлива**

1. **За да може функция да получи стойност от извикващата програма, стойността трябва да бъде предава като аргумент (действителната стойност)**

2. **За да може извикващата програма да получи стойност от дадена функция, стойността трябва да бъде изрично върната в извиканата функция с return .**

CODE ACADEMY

Задачи

1. Да се напише програма с 4 функции, предназначени за събиране, изваждане, умножение и деление на 2 реални числа.
Програмата да чете стойности на 2 реални числа и да пресметне сумата, разликата, произведението и частното на тези две числа.

2. Да се напише функция, връщаща максималното от две числа.
Програмата ще чете 3 числа и извежда максималното от тези 3 числа, като използва написаната функция. Задачата да се направи по два начина — с оператор if и с тернарен оператор.

3. Функция, намираща сумата на целите числата в даден интервал [a,b], като a и b са параметри на функцията.

```
long sum(int a, int b)
```

Програмата чете 3 интервала и отпечатва в кой интервал сумата е нечетно числа.

Задачи

4. Да се напише програма, която използва функция, обръщаща до 20 цифрено цяло число. Водещите нули не се извеждат.

Ако числото 123 е параметър —> връща 321
120 —> 21

Програмата отпечатва резултата от изпълнението на функцията.

long reverse(long n) 19

5. Функция, която отпечатва едно цяло десетично число в двоична бройна система — чрез 0 и 1.

void printBinary(int n)

CODE ACADEMY

Задачи

6. Да се напише програма, която използва функция, проверяваща дали едно число е цяло.

bool isInteger(float n)

7. Функция, проверяваща дали едно число е просто. Програма, която от 3 въведени числа, извежда само простите от тях.

20

8. Функция, пресмятаща n!=1*2*3*...*n

long factoriel(int n)

Програма, която за въведено едноцифрено число k, пресмята и отпечатва

1!+2!+...+k!

CODE ACADEMY

Задачи

9. Функция, проверяваща дали едно число е точен квадрат. Програма, която чете число и отпечатва резултата от извикването на функцията.

10. Функция, която връща най-големия точен квадрат в даден интервал (a, b). Програма, която чете a и b и отпечатва резултата от извикването на функцията.

11. Функция, която проверява, дали един символ е латинска буква. Програма, която чете символи от клавиатурата, докато бъде въведена цифрата 0 и извежда броят на въведените букви.

CODE ACADEMY

Рекурсивни функции

Рекурсията е техника, която позволява да разделим проблем на един или повече подпроблеми, които са подобни по форма на оригиналния проблем.

Рекурсията е техника за програмиране, използваща функция или алгоритъм, който се извиква един или повече пъти, докато не бъде изпълнено определено условие.

Рекурсия: Функция, която извиква себе си.

Решаването на проблем с помощта на рекурсия зависи от решаването по-малки (по-прости) появи на същия проблем, докато проблемът е достатъчно прост, за да можете да се решите директно

Рекурсията е мощен заместител на итерацията (циклите) и е подходяща за решаване на определени видове проблеми.

При рекурсия се достига до елегантен, опростен и кратък код.

CODE ACADEMY

Рекурсивни функции

Всеки рекурсивен алгоритъм включва поне 2 случая:

1) основен случай - просто събитие, на което може да се отговори директно

2) рекурсивен случай -по-сложна поява на проблема, на който не може да се отговори директно, но вместо това може да се опише чрез по-малки случаи на същия проблем

Трите правила за рекурсията са:

- **Всеки (валиден) вход трябва да има случай (рекурсивен или базов)**
- **Трябва да има основен случай, който не прави рекурсивни извиквания**
- **Рекурсивният случай трябва да направи проблема по-опростен и да се придвижим към основния случай**

Задачи

12. Функция, връщаща n-тото число на Фибоначи.

A(1)=1, A(2) =1, A(3)=2 A(n)=A(n-1)+A(n-2)

Решение итеративно, с цикъл, без използване на рекурсия

```
long fibonachi(int n){
                        if(n==1 || n==2) return 1;
        else{
                ……
        }
…..
return…
}

int main()
{
int n;
 scanf(“%d”, &n);
printf(  fibonachi(n) );
}
```

Задачи

12. Решение като се използва на рекурсия

```
#include<stdio.h>

long fib(int n)
{
   if(n==1 || n==2)
      return 1;
   else{
      return fib(n-1)+ fib(n-2);
   }
}

int main(){
            int num;
            scanf("%d",&num);
            printf("%ld", fib(num));
}

//num=2  -->fib(2)   return 1;
//num=3  -->fib(3)  -->fib(2)+fib(1) return  1+1 =2
//num=4  -->fib(4)  -->fib(3)+fib(2) return  fib(2)+fib(1) +1 =(1+1)+1
```

Задачи

13. Рекурсивна функция, връщаща n!=1*2*3*4*5*...(n-1)*n=(n-1)!*n

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

26

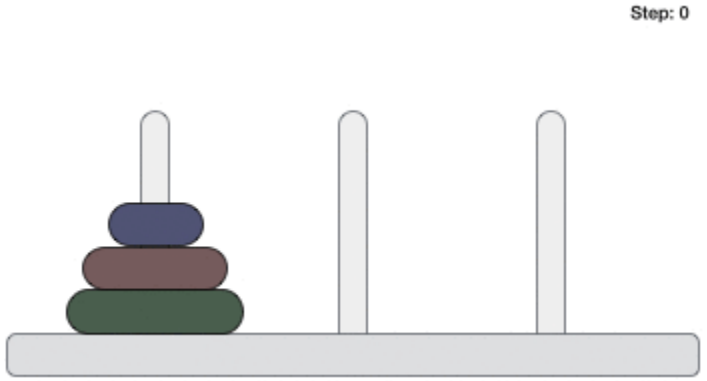
```
int main()
{
        int n;
        scanf(“%d”, &n);
        printf(  fact(n) );
}
```

CODE ACADEMY

Задачи

14. Като се използва рекурсивна функция, да се реши задачата за Ханойските кули.
Дадени са n броя диска. Числото n се въвежда от клавиатурата.
Задачата е да се преместят всички дискове от колче A на колче C, като колче B се използва за помощно.

При преместването се спазва правилата:
1) мести се само по един диск
2) при преместването се поставя по-малко върху по-голямо



CODE ACADEMY

14. *Решение на задачата за Ханойските кули*

```
void hanoi(int n, char start, char end, char help )
{
    if(n==1)
        printf("Move A --> C");
    if(n==2)
    {
        printf("Move A --> B\n");
        printf("Move A --> C\n");
        printf("Move B --> C\n");
    }
     else{
        hanoi(n-1, start, help, end);
        printf("Move A --> C\n");
        hanoi(n-1, help, end, start);
     }
}

int main()
{
    int n;
    printf("Enter number=");
    scanf("%d", &n);
    hanoi(n,'A','C','B');
}
```

CODE ACADEMY

15. Като се използва рекурсивна функция, да се напише програма за бързо повдигане на степен за едно цяло положително число n.

```
#include<stdio.h>

long stepen(int n, int k)
{
    if(k==1) return n;
    if(k%2==0)
      return stepen(n,k/2)*stepen(n,k/2);
    else
        return n*stepen(n,k-1);
}

int main()
{
 int n,k;
 scanf("%d%d",&n,&k);

 printf("%d", stepen(n,k));

}
```

CODE ACADEMY

16. Да се напише рекурсивна функция за намиране на максимален елемент в масив от цели числа. Да се напише програма, която извиква тази функция за да намери най-голямото число в даден масив.

```
#include<stdio.h>

int  max(int *a, int n){
     if(n==1) return a[0];
       else{
            int m=a[n-1];
            int maxn= max(a, n-1);

            if( m< maxn)
               m=maxn;
            return m;
     }
}

int main()
{
    int a[5]={1,3,5,6,8};
    printf("%d", max(a,5)) ;
}
```

CODE ACADEMY

Задачи

17. **Функция, която намира сумата от елементите на масив и програма, намираща средното аритметично от елементите на масив.**

18. Функция, която отпечатва стойностите на масив.

CODE ACADEMY

Задачи за самостоятелна работа

19. Да се напише рекурсивна функция
int numPrint(int n),
С помощта на която се отпечатват всички естествени числа до 100.

20. Напишете програма на С за броене на цифрите на дадено число с помощта на рекурсивна функция.

21. Да се напише функция, която намира и връща лицето на правоъгълник.

22. Да се напише функция, която намира и отпечатва лицето на кръг.

23. Да се напише функция с име isTriangle, която по дадени три цели числа определя дали съществува триъгълник със страни тези числа.

24. Функция, която по зададени граници на интервал [a,b] , намира и извежда сумата от четните числа от този интервал.