Використання функцій

Визначення власних функцій

Приклад 1

Завдання. Користувач вводить будь-яке число a. На екран будуть виведені значення

1)до введеного числа a додається 5

2) сума двох введених чисел (a+a)

Текст програми

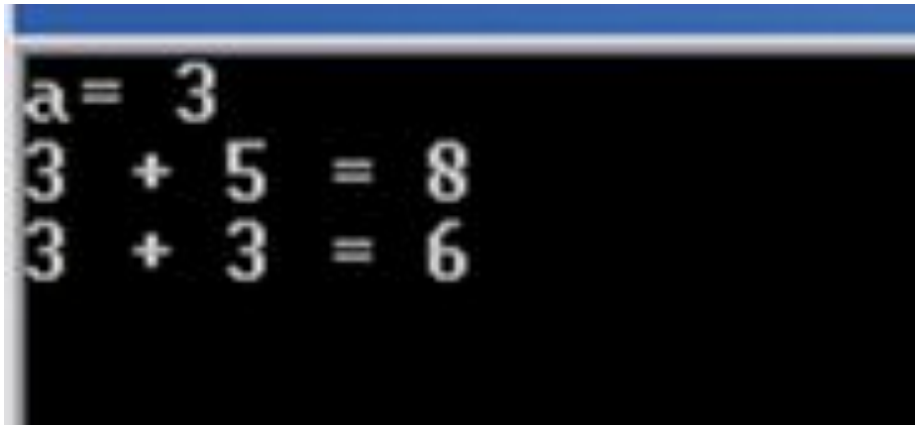
```
#include <stdio.h>
int sum(int x, int y); // прототип функції sum

int main() {
  int a, r;
  printf("a= ");
  scanf("%d",&a);
  r = sum(a, 5);    // вызов функции: x=a, y=5
  printf("%d + 5 = %d\n",a, r);
   r = sum(a, a);    // вызов функции: x=a, y=a
  printf("%d + %d = %d\n",a,a, r);
 }
```

```
int sum(int x, int y) {
  int k;              // об’ява змінної
  k = x + y;          // оператор
  return(k);          //повернення результату
}
```

```
int sum(int x, int y) {
  int k;              // об’ява змінної
  k = x + y;          // оператор
  return k ;      //повернення результату
}
```

```
int sum(int x, int y) {
return (x+y);     //повернення результату
}
```

Результат роботи програми



Будь-яка функція (метод) у своєму тілі може викликати сама себе. *Рекурсія* – це такий спосіб задавання функції, при якому результат повернення з функції для даного значення аргументу визначається на основі результату повернення з тієї ж функції для попереднього (меншого або більшого) значення аргументу.

Якщо функція (метод) викликає сама себе, то такий виклик називається *рекурсивний виклик* функції. При кожному рекурсивному виклику запам'ятовуються попередні значення внутрішніх локальних змінних та переданих у функцію параметрів. Щоб наступний крок рекурсивного виклику відрізнявся від попереднього, значення як мінімум одного з параметрів функції повинно бути змінене. Припинення процесу рекурсивних викликів функції відбувається, коли змінюваний параметр досягнув деякого кінцевого значення, наприклад, оброблено останній елемент в масиві.

Рекурсивне звернення до функції може бути здійснене, якщо алгоритм визначений рекурсивно.

Щоб циклічний процес перетворити в рекурсивний, потрібно вміти визначити (виділити) три важливі моменти:

- умову припинення послідовності рекурсивних викликів функції. Умова припинення вказується в операторі return;
- формулу наступного елементу або ітератору, що використовується в рекурсивному процесі. Ця формула вказується в операторі return;
- перелік параметрів, що передаються в рекурсивну функцію. Один з параметрів обов'язково є ітератор (лічильник), який змінює своє значення. Інші параметри є додатковими, наприклад, посилання на масив, над яким здійснюється обробка.

Приклад 2

Завдання. Обчислити факторіал числа, що введено користувачем

$f = 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n$, де $n >= 1$.

Рекурсивний виклик можна організувати двома способами:

в порядку зростання 1, 2, …, *n*;

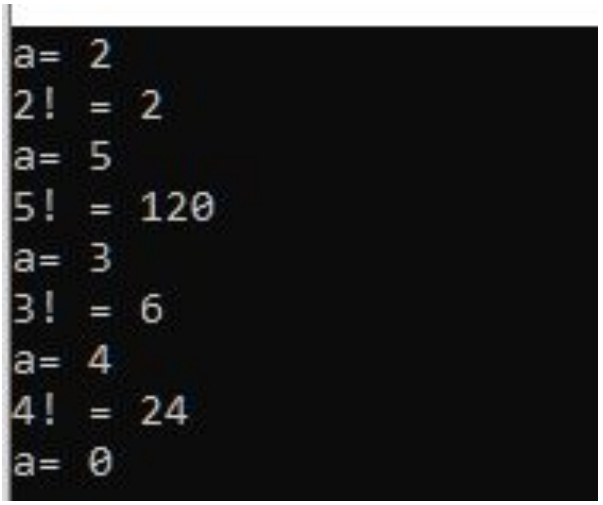
в порядку спадання *n*, *n*-1, …, 2, 1.

Текст програми

```
#include <stdio.h>


int fact1(int num) {
 if(num==1) // умова завершення рекурсивного процесу
 return(1);
 else return(num*fact1(num-1)); // рекурсивний виклик, перехід до попереднього числа
}
```

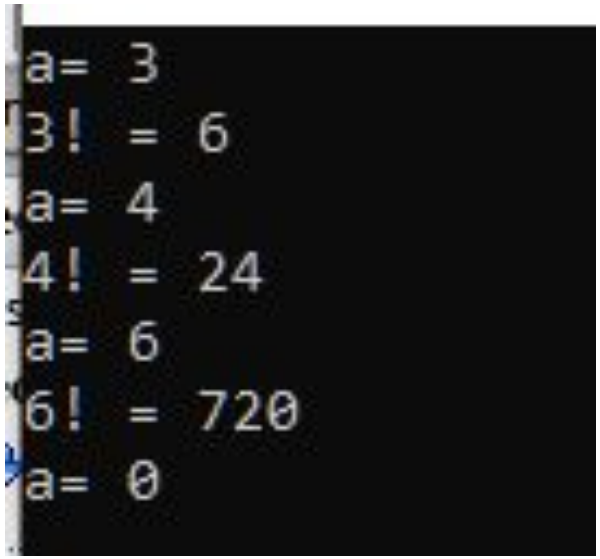
```
int main() {
 int a, r;
 do{
 printf("a= ");
 scanf("%d",&a);
 if(a==0) break;
r=fact1(a);
 printf("%d! = %d\n",a,r);
 }while(1);
return(0);
}
```

Результат роботи програми



```
#include <stdio.h>

int fact2(int num,int i) {
 if(num==i) // умова завершення рекурсивного процесу
 return(num);
 else return(i*fact2(num,i+1)); // рекурсивний виклик, перехід до наступного числа
}
```

```
int main() {
 int a, r;
 do{
 printf("a= ");
 scanf("%d",&a);
 if(a==0) break;
r=fact2(a,1);
printf("%d! = %d\n",a,r);
 }while(1);
return(0);
}
```


a= 3
3! = 6
a= 4
4! = 24
a= 6
6! = 720
a= 0

У функцію fact1() передається 1 параметр – максимально можливе значення num, що бере участь у рекурсивному множенні. Другого параметру тут не потрібно, оскільки межа припинення рекурсивного процесу є відома і рівна 1. У функції fact1() рекурсивний процес завершується коли num=1.

У функцію fact2() передаються 2 параметри. Перший параметр визначає максимально можливе значення num, яке може бути помножене. Другий параметр визначає поточне значення **i** яке приймає участь у множенні.

Приклад 3

Знайти суму перших N членів ряду, заданого рекурентною формулою.

$F_1 = 1,$ для $\forall i > 1$ $F_i = \log(|F_{i-1} * i|) + \sin(i)$

```
void fun3(int N)
{
float s=0,x=1;
int i;
for(i=1;i<=N;i++)
{
   if(i!=1)
   {
       x=log(i*(fabs(x)))+sin(i);
   }
   printf("\n%f",x);
   s=s+x;
}
printf("\n%f",s);
}
```

```
int main()
{ do{
    printf("\nEnter N\n");
    scanf("%d",&N);
    if (N>=1) fun3(N);
    else printf("error");
     }while (N!=0);
     return 0;
}
```

```
Enter N
1

1.000000
1.000000
Enter N
2

1.000000
1.602445
2.602445
Enter N
3

1.000000
1.602445
1.711263
4.313707
Enter N
5

1.000000
1.602445
1.711263
1.166723
0.804713
6.285143
Enter N
0
error
Process returned 0 (0x0)   execution time : 13.694 s
Press any key to continue.
```

```
#include <stdio.h>;
#include <stdlib.h>
#include <math.h>
float s=0;
int N;
void fun2(float x,int i,float s)
{

   if(i<=N)
   { if (i>1)
        x=log(i*(fabs(x)))+sin(i);

   printf("\n%f",x);
   s=s+x;
   fun2(x,i+1,s);}
   else  printf("\n%f",s);
   return x;
}
```

```
int main()
{ do{
    printf("\nEnter N\n");
    scanf("%d",&N);
    if (N>=1) fun2(1,1,0);
    else printf("error");
     }while (N!=0);
return 0;
}
```

```
Enter N
1

1.000000
1.000000
Enter N
2

1.000000
1.602445
2.602445
Enter N
3

1.000000
1.602445
1.711263
4.313707
Enter N
5

1.000000
1.602445
1.711263
1.166723
0.804713
6.285143
Enter N
0
error
Process returned 0 (0x0)   execution time : 19.680 s
Press any key to continue.
```

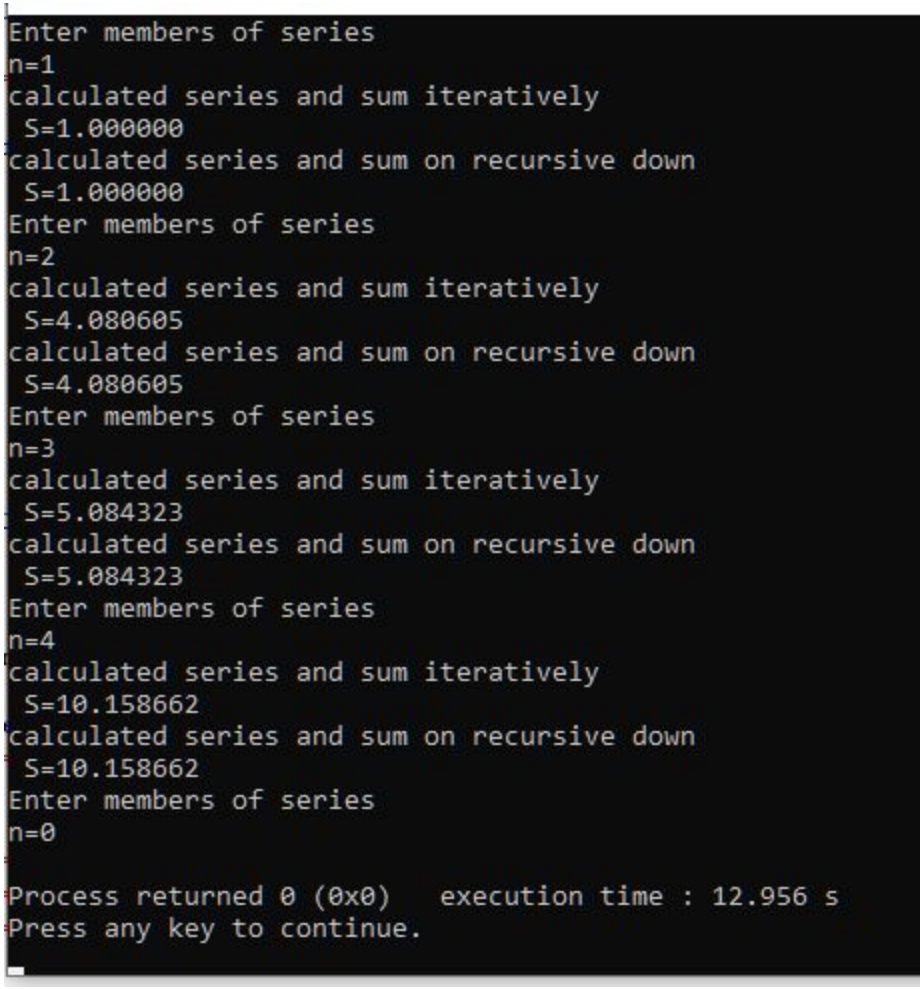
Знайти суму перших N членів ряду, заданого рекурентною формулою.

$F_1 = 1,$ для $\forall i > 1$ $F_i = \log(|F_{i-1} * i|) + \sin(i)$

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
double FF1(int i,int n,double f,double s)
{
   if (n==1) {return f;}
   else {
      if(n>1){
         if(i<n){
            s=s+f;
            f=2*cos(f)+i+1;
            i=i+1;
            return FF1(i,n,f,s);
         }
         else return s+f;
      }
      else return s;
   }
}
```

```
double sumit(int n)
{
   double f=1,s=0;
   int i=1;
   while(i<n)
   {
      s=s+f;
      f=2*cos(f)+i+1;
      i=i+1;
   }
   return s+f;
}
```

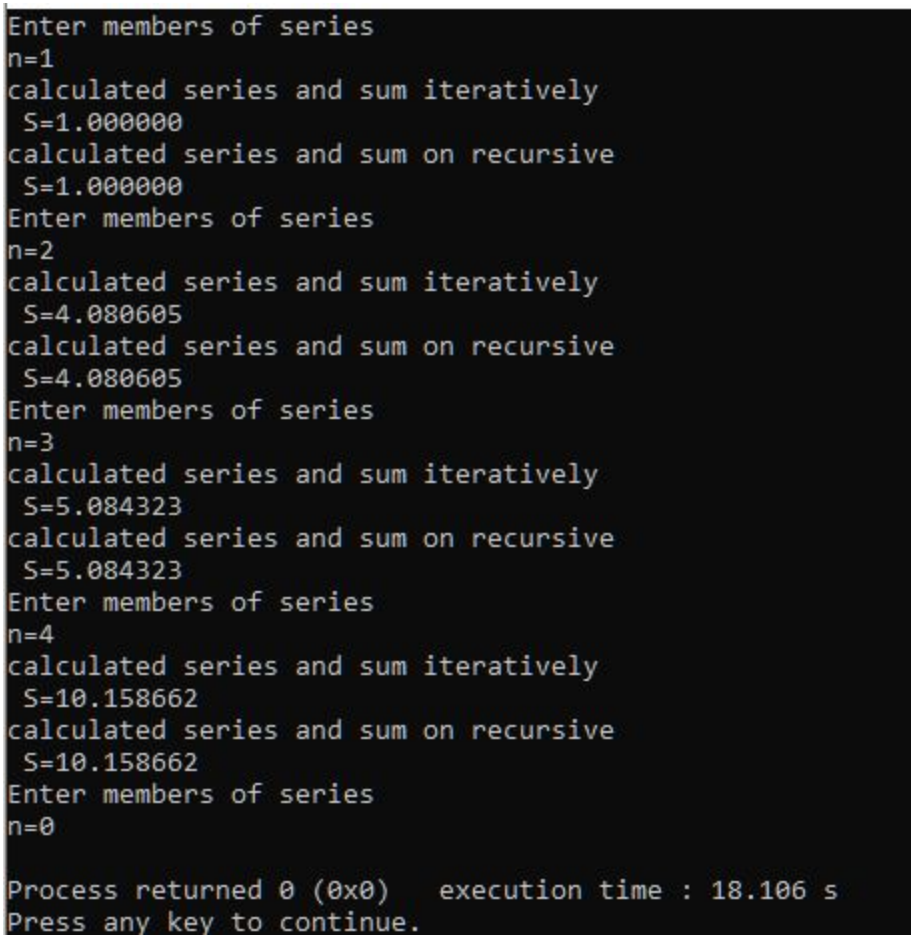
```
int main()
{int n;
  do{
  printf("Enter members of series\nn=");
   scanf("%d",&n);
   if(n==0) break;
   printf("calculated series and sum iteratively \n S=%lf\n",sumit(n));
   printf("calculated series and sum on recursive down\n S=%lf\n",FF1(1,n,1,0));
   }while(n!=0);
   return 0;
}
```


Enter members of series
n=1
calculated series and sum iteratively
S=1.000000
calculated series and sum on recursive down
S=1.000000
Enter members of series
n=2
calculated series and sum iteratively
S=4.080605
calculated series and sum on recursive down
S=4.080605
Enter members of series
n=3
calculated series and sum iteratively
S=5.084323
calculated series and sum on recursive down
S=5.084323
Enter members of series
n=4
calculated series and sum iteratively
S=10.158662
calculated series and sum on recursive down
S=10.158662
Enter members of series
n=0
Process returned 0 (0x0) execution time : 12.956 s
Press any key to continue.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
double FF2(int i,int n,double f)
{
   if (n==1) {return f;}
   else {
      if(n>1){
         if(i<n){
            f=2*cos(f)+i+1;
             i=i+1;
             return f+FF2(i,n,f);
         }
         else return 1;
      }
      else return 0;
   }
}
```

```
double sumit(int n)
{
   double f=1,s=0;
   int i=1;
   while(i<n)
   {
      s=s+f;
      f=2*cos(f)+i+1;
      i=i+1;
   }
   return s+f;
}
```

```
int main()
{int n;
  do{
  printf("Enter members of series\nn=");
   scanf("%d",&n);
   if(n==0) break;
   printf("calculated series and sum iteratively \n S=%lf\n",sumit(n));
   printf("calculated series and sum on recursive\n S=%lf\n",FF2(1,n,1));
   }while(n!=0);
   return 0;
}
```


Enter members of series
n=1
calculated series and sum iteratively
S=1.000000
calculated series and sum on recursive
S=1.000000
Enter members of series
n=2
calculated series and sum iteratively
S=4.080605
calculated series and sum on recursive
S=4.080605
Enter members of series
n=3
calculated series and sum iteratively
S=5.084323
calculated series and sum on recursive
S=5.084323
Enter members of series
n=4
calculated series and sum iteratively
S=10.158662
calculated series and sum on recursive
S=10.158662
Enter members of series
n=0
Process returned 0 (0x0) execution time : 18.106 s
Press any key to continue.

Переваги та недоліки використання рекурсії

Рекурсію часто порівнюють з ітерацією. Організація циклічного процесу з допомогою рекурсії має свої переваги та недоліки.

Можна виділити такі взаємозв‘язані ***переваги рекурсії*:**

природність (натуральність) представлення складних, на перший погляд, алгоритмів;

рекурсивний алгоритм більш читабельний у порівнянні з ітераційним;

для багатьох поширених задач рекурсію легше реалізувати ніж ітерацію. Рекурсія добре підходить для реалізації алгоритмів обходу списків, дерев, аналізаторів виразів, комбінаторних задач тощо.

Недоліки рекурсії полягають у наступному:

порівняно з ітерацією багаторазовий виклик рекурсивної функції потребує більше часу. Це зв ‘язано з тим, що при виклику рекурсивного методу його параметри копіюються в стек. При завершенні виклику рекурсивної функції попередні значення параметрів витягуються зі стеку, що призводить до зайвих операцій. Ітераційний алгоритм для тієї самої задачі працює швидше;

якщо рекурсивна функція містить великі об’єми локальних внутрішніх змінних і велику кількість параметрів, то використання рекурсії не є ефективним. Це зв’язано з тим, що для кожного рекурсивного виклику потрібно робити копії цих змінних та параметрів. При великій кількості рекурсивних викликів це призведе до надмірного використання пам’яті.