ЛАБОРАТОРНА РОБОТА №13

Тема: Програмування з використанням рекурсивних функцій

Мета: Навчитись викликати функцію з неї самої або через інші функції

Теоретичні відомості

Рекурсія — метод визначення класу чи об'єктів методів попереднім заданням одного чи декількох (звичайно простих) його *базових* випадків чи методів, а потім заданням на їхній основі правила побудови класу, який визначається.

рекурсія значить самоповторюваний шаблон, наприклад, це може бути функція визначена через себе. Інакше кажучи, це функція, що викликає саму себе. Кожна рекурсивна функція має умову завершення; інакше вона буде викликати себе безперестанку, і цю умову можна назвати *базова умова*

Іншими словами, рекурсія — часткове визначення об'єкта через себе, визначення об'єкта з використанням раніше визначених. Рекурсія використовується, коли можна виділити самоподібність задачі.

Визначення у логіці, що використовує рекурсію, називається *індуктивним*.

Рекурсія в програмуванні

У програмуванні рекурсія — виклик функції чи процедури з неї самої (звичайно з іншими значеннями вхідних параметрів), чи безпосередньо через інші функції (наприклад, функція A викликає функцію B, а функція B — функцію A). Кількість вкладених викликів чи функції процедури називається глибиною рекурсії.

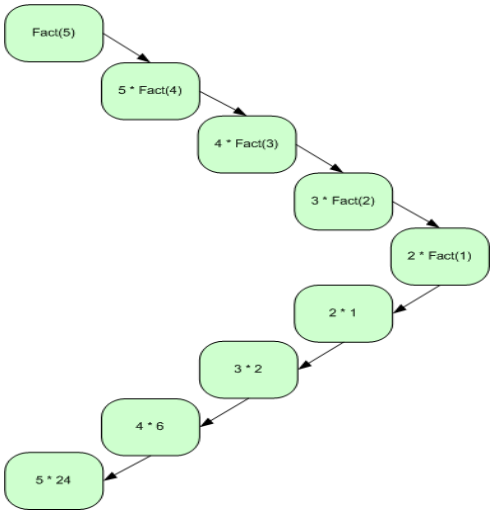
Міць рекурсивного визначення об'єкта в тім, що таке кінцеве визначення здатне описувати нескінченно велике число об'єктів. За допомогою рекурсивної програми ж можливо описати нескінченне обчислення, причому без явних повторень частин програми.

Лінійна рекурсія

Лінійна рекурсія це найпростіший вид рекурсії і можливо найуживаніша рекурсія. У цієї рекурсії, одна функція просто викликає себе доти, доки не досягне умови завершення (також відомої як базова умова); цей процес відомий як **намотування.** Як тільки виконана умова завершення, виконання програми повертається до викликальника; це зветься **розмотуванням.**

Упродовж намотування і розмотування функція може виконувати якісь додаткові корисні задачі; у випадку факторіала вона множить вхідне значення на значення повернуте під час фази розмотування. Цей процес можна зобразити у вигляді такої діаграми (знизу), яка показує обидві фази функції обчислення факторіала із використанням лінійної рекурсії.

Варто уникати надлишкової глибини рекурсії, тому що це може викликати переповнення стека викликів.

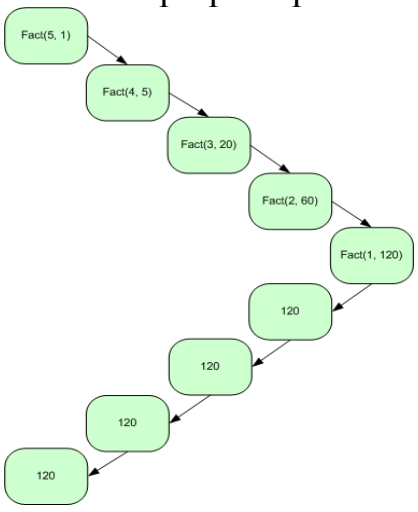


Хвостова рекурсія

Існує спеціальний тип рекурсії, називаний «хвостовою рекурсією». Інтерпретатори і компілятори функціональних мов програмування, що підтримують оптимізацію коду (вихідного і/чи що виконується), виконують хвостову рекурсію в обмеженому обсязі пам'яті за допомогою ітерацій.

Хвостова рекурсія замість рекурсивного виклику функції, процедура скидає свої параметри, встановлюючи ті, які б вона отримала при рекурсивному виклику, і потім виконує знову

Хвостова рекурсія це спеціальна форма лінійної рекурсії, де рекурсивний виклик зазвичай іде останнім у функції. Цей тип рекурсії здебільшого більш ефективний, бо розумні компілятори автоматично перетворять таку рекурсію в цикл задля уникнення вкладених викликів функцій. Через те, що рекурсивний виклик функції зазвичай останній, що робить функція, вона не має потреби ще щось робити під час розмотування; натомість, вона просто повертає значення отримане через рекурсивний виклик. Ось приклад тої самої програми реалізованої як хвостова рекурсія.



Визначити хвостову рекурсію можна математично через наступну формулу; інакше кажучи, коли значення “n” нуль, просто повернути значення “a”; якщо значення “n” більше ніж нуль, викликати рекурсивну функцію з параметрами “n-1” і “n*a”. Також, можна зауважити, що під час фази розмотування кожна рекурсивно викликана функція просто повертає значення “a”.

$$f(n,a) = \begin{cases} a, & n = 0 \\ f(n-1, n*a), & n > 0 \end{cases}$$

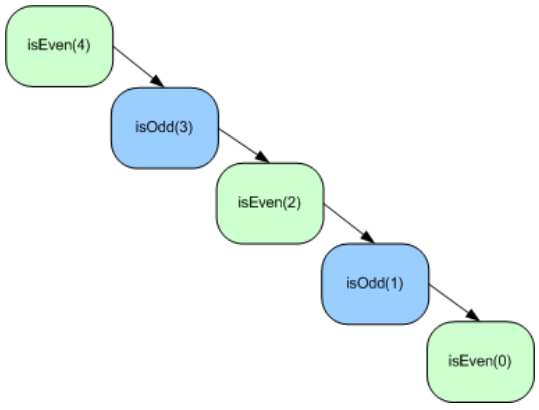
```
int Factorial(int no, int a)
{
  // перевірка на помилковий параметр
  if (no < 0)
   return -1;

  // умова завершення
  if (0 == no || 1 == no)
   return a;

  // хвостовий рекурсивний виклик
  return Factorial(no - 1, no * a);
}
```

Обопільна рекурсія

Обопільна рекурсія також відома як *непряма рекурсія*. В цьому типі рекурсії, дві або більше функції викликають одна одну циклічно. Це єдиний шлях для здійснення рекурсії в мовах, що не дозволяють вам викликати функції рекурсивно. Умова завершення в такій рекурсії може бути в одній або всіх функціях.



Математично, ви можете визначити ці функції як

$$f(n) = \begin{cases} true, & n = 0 \\ isOdd(n-1), & n > 0 \end{cases}$$

$$f(n) = \begin{cases} false, & n = 0 \\ isEven(n-1), & n > 0 \end{cases}$$

Практичне завдання

У середовищі С++ набрати наступні задачі та виконати рекурсивні розрахунки. Парні і непарні варіанти виконують, відповідно, парні та непарні задачі.

Задача 1. Згідно з заданим натуральним числом n обчислити суму n членів гармонічного ряду

$1 + {}^{1}/_{2} + \cdots {}^{1}/_{n}$ у вигляді звичайного дробу.

```
#include <iomanip.h>
void scdr(long* p1, long* p2);
void main()
{ cout<<"Введіть кількість членів ряду:";
int n;
cin>>n;
long s1 =0;
long s2 = 1;
for(int k=i ;k<=n;k++)
{ s1=s1*k+s2;
  s2*=k;
  scdr(&s1, &s2); }
cout<<"Сума "<<n<<" членів ряда: "<<s1<<" / "<<s2<<endl; }
long gcd(long x, long y)
{ long r;
If (x<y)
  {r=y;
  y=x;
  x=r;}
while(y)
{ r=x%y;
  x=y;
  y=r;}
return (x); }
void scdr (long* p1, long* p2)
{ long t;
  t=gcd(*pl, *p2);
  *p1/=t;
  *p2/=t; }
```

Задача 2. Знайти факторіал цілого невідємного числа N який позначається N!

.Для розв'язку задачі рекурсивним алгоритмом необхідно скористуватись рекурентним правилом обчислення факторіалу І умовою виходу з рекурсії:.

N!=(N−1)! ∗ N, якщо N = 0, то N! = 1

//Класичний спосіб

```
#include <iostream>;
#include<cmath>;
#include<fstream>;
using namespace std;
int main()
{
ifstream in;
ofstream out;
in.open("input.txt");
out.open("output.txt");
long long int n, p=1;
cin>>n;
for (int i=1; i<=n; i++)
{ p=p*i; }
 cout<<p;
in.close();
out.close();
return 0;
}
```

//Розв’язок за допомогою рекурсії

```
#include <iostream>;
#include<cmath>;
#include<fstream>;
using namespace std;
// опис рекурсивної функц
long long int fact(int k)
{
if (k==1) return 1;
else
   return k*fact(k-1);
}
int main()
{
ifstream in;
ofstream out;
in.open("input.txt");
out.open("output.txt");
long long int n;
in>>n;
out<<fact(n); // Виклик рекурсії
in.close();
out.close();
return 0;
```

```
}
```

Задача 3. ***По заданому числу N обчислити число Фібоначі F(N)***

Для розв'язку задачі необхідно знати, що послідовністю Фібоначі називається послідовність чисел **$F_0, F_1, ..., F_n, ...,$** де $F_0 = 0, F_1 = 1, F_k = F_{k-1} + F_{k-2}$ $(k > 1)$

//Класичний спосіб . Динамічне програмування.

Псевдокод

```
створити масив F[0...n]
F[0]←0,F[1]←1
для i від 1 до n:
     F[i]← F[i−1] +F[i−2]
повернути F[n]
```

//Розв'язок за допомогою рекурсії .

Псевдокод функції

```
Fib1(n)
Якщо n ≤ 1 :
     повернути n
```

Вернути Fib1(n−1) +Fib1(n−2)

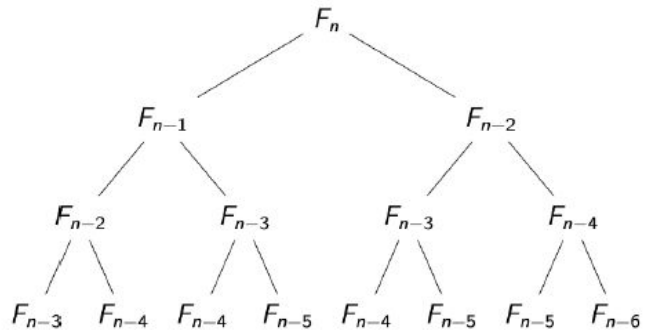


Рис. 1 – Дерево рекурсивних викликів

Задача 4: ***Відсортувати одномірний масив*** *a[n]* ***за спаданням елементів, використовуючи рекурсивну функцію швидкого сортування. Записати код програми в звіт***.

```
/* відсортувати масив по зрастанню методом швидкого сортування */
   #include <iostream.h>
   #include <conio.h>
   template <class T>
   void QuickSortRecur(T* a, int Nend)
   {
   // початковий масив a[ ], a[Nend] – його останній елемент
    int i = 0, j = Nend;
    T rab, c;
    c = a[Nend >>1];
   /* вибір центрального елемента з використанням операції
   побітового зсуву праворуч */
    do
    { while (a[i] < c) i++;
     while (a[j] > c) j--;
     if 0 <=j)
     { rab = a[i];
      a[i++] = a[j];
```

```
    a[j--] = rab; }
 }
 while (i <= j);
/* рекурсивні виклики функції QuickSortRecur(), якщо лишилось, що
сортувати */
 if (j > 0) QuickSortRecur(a, j);
 if (Nend > i) QuickSortRecur(a+i, Nend-i);
}
void main()
{ int a[ ] = (2, 7, 6, 9, 45, 4, 3, 67, 104, 1, 99, 72, 43, 8, 4,
28, 100};
  int n = sizeof(a)/sizeof(int);
  QuickSortRecur(a, n-1);
  cout << "\n Result Quick-sortirovki massiva" << endl;
  for (int i = 0; i<n; i++)
  cout << a[i] <<' ';
  getch ();
}
```

Записати алгоритми та лістинги програми до звіту та зробити висновки.
Реалізувати алгоритм програм на мові Pascal.
Додатково: За допомогою рекурсії перевести число з десяткової системи числення у іншу (2-ійкову, 8-кову, 16-тиркову)

Контрольні запитання

1. Що таке рекурсія?
2. Коли використовується рекурсія?
3. Які можливості надає рекурсивна задача?
4. Який загальний випадок рекурсивної процедури?
5. Яка рекурсія називається «хвостовою»?
6. Яка рекурсія називається «лінійною»?
7. Яка рекурсія називається «обопільною»?
8 Наведіть приклади задач, для розв’язку яких використовують рекурсію (крім тих, що розглядались у лабораторній роботі.

.