Рекурсия

Базово програмиране
Лекция 12
Милена Карова
кат. КНТ ТУ-Варна

Понятие за рекурсия

- Търсене в масив може да се организира като се раздели масива на 2 и после пак на 2 и т.н.
- Техниката, която може да се приложи се нарича рекурсия
- Една функция е рекурсивна ако вика сама себе си, т. е. в тялото на функцията има обръщение към функцията
- Правилното декомпозиране на алгоритъма е предпоставка за правилното използване на рекурсията

Прост пример за рекурсия

- Рекурсивна void функция ще напише числото 1984 вертикално

- Функцията би изглеждала така:

```
if (n < 10)
{
    cout << n << endl;
}
else //n is two or more digits long:
{
    write_vertical(the number n with the last digit removed);
    cout << the last digit of n << endl;
}
```

Recursive subtask

```
n / 10 //the number n with the last digit removed
n % 10 //the last digit of n
```

Код на програмата

```
//Program to demonstrate the recursive function write_vertical.
#include <iostream>
using namespace std;

void write_vertical(int n);
//Precondition: n >= 0.
//Postcondition: The number n is written to the screen vertically
//with each digit on a separate line.

int main( )
{
    cout << "write_vertical(3):" << endl;
    write_vertical(3);

    cout << "write_vertical(12):" << endl;
    write_vertical(12);

    cout << "write_vertical(123):" << endl;
    write_vertical(123);

    return 0;
}

//uses iostream:
void write_vertical(int n)
{
    if (n < 10)
    {
        cout << n << endl;
    }
    else //n is two or more digits long:
    {
        write_vertical(n / 10);
        cout << (n % 10) << endl;
    }
}
```

Sample Dialogue

```
write_vertical(3):
3
write_vertical(12):
1
2
write_vertical(123):
1
2
3
```

Ред на изпълнение на рекурсивните извиквания

```
if (123 < 10)
{
    cout << 123 << endl;
}
else //n is two or more digits long:
{
    write_vertical(123 / 10);
    cout << (123 % 10) << endl;
}
```

Computation will stop here until the recursive call returns.

```
write_vertical(n / 10);
        ↓
write_vertical(123 / 10);
        ↓
write_vertical(12);
```

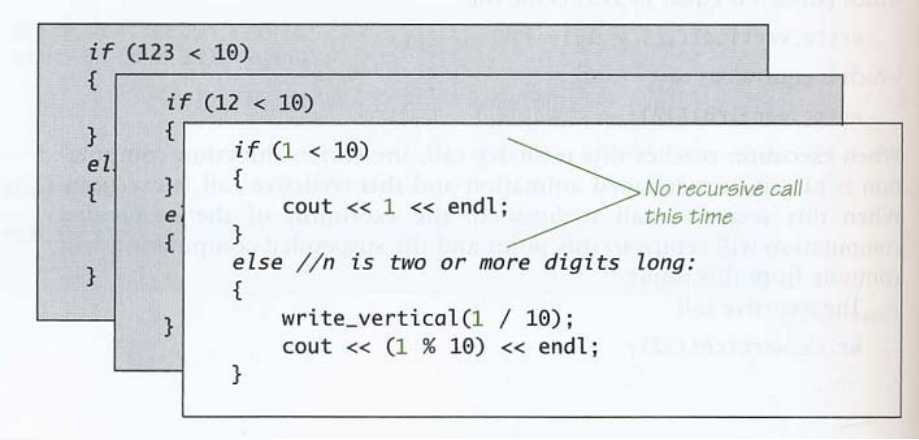
```
if (123 < 10)
{
    cou
}
else //
{
    wri
    cou
}
```

```
if (12 < 10)
{
    cout << 12 << endl;
}
else //n is two or more digits long:
{
    write_vertical(12 / 10);
    cout << (12 % 10) << endl;
}
```

Computation will stop here until the recursive call returns.

Две висящи изчисления

Ред на изпълнение на рекурсивните извиквания

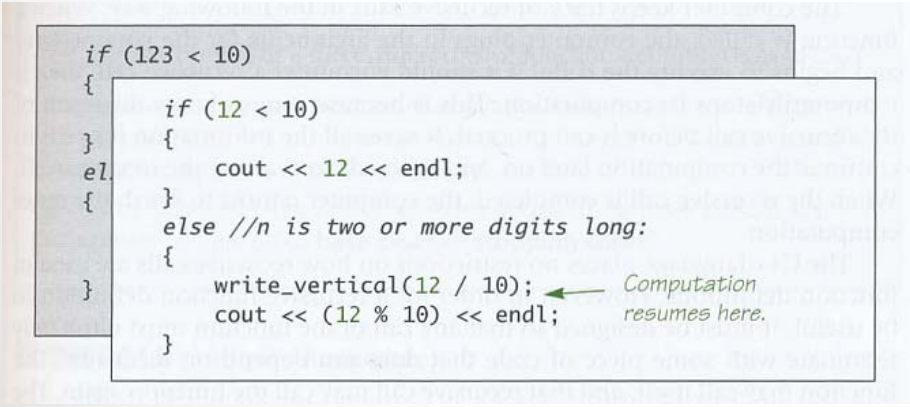



```
write_vertical(12 / 10);
```

↓

```
write_vertical(1);
```

Извежда се 1



Извежда се 2

```
if (123 < 10)
{
    cout << 123 << endl;
}
else //n is two or more digits long:
{
    write_vertical(123 / 10);
    cout << (123 % 10) << endl;   ← Computation resumes here.
}
```

Извежда се 3

Как работи рекурсията

- Компютърът запазва поредицата от рекурсивни извиквания
- При извикване на функцията, компютърът предава аргументите към параметрите и започва да изпълнява кода на функцията
- При срещане на рекурсивно извикване на функция, временно спира изпълнението на кода
- За да продължи функцията, тя трябва да знае резултата от рекурсивното извикване
- Тя запомня всичко, което е пресметнала до момента и започва изпълнението на рекурсивното извикване
- След приключването му, тя се връща в точката в която е прекъснала и продължава изчислението.

Как приключва рекурсивната функция

- Няма ограничения за броя извиквания на рекурсивната функция
- За да бъде използваема, всяка рекурсивна функция трябва най-накрая да завършва с част от код, който не зависи от рекурсията
- Всяка рекурсивна функция има:
  - базов случай (base case), при който тя изпълнява своята задача чрез една или повече по-малки версии на задачата.
  - един или повече случаи, при които не се използват рекурсивни извиквания (stopping case)
  - най-често конструкция if – else определя кой от случаите ще се изпълни
- Всяко извикване на функцията трябва евентуално да води до stopping case в противен случай функцията никога няма да завърши. Получава се безкрайна верига от рекурсивни извиквания

Осигуряване на stopping case

- Най- сигурният начин да се достигне stopping case е функцията да осигури позитивна числова стойност, която прогресивно намалява при всяко рекурсивно повикване и функцията спира при „малка“ стойност (в примера рекурсивното повикване продължава до стойност 10)
- При стойност 10 рекурсивните повиквания спират и изчислителният процес се връща назад до оригиналното извикване и спира

Стек

- За да се следи рекурсията, компютърните системи използват структура от паметта, наречена стек.
- Метод на LIFO(last in/ first out)
- За да се разгледа информацията на стека, тя трябва да се намира на върха му (top) (достъпна е само информацията на върха)
- Постъпването на информация е push, извеждането на информация е pop
- За да се разгледа 3 елемент на стека, трябва да се извадят предходните 2.
- LIFO memory structure
- Stack overflow съществува лимит на размера на стека и при многобройни рекурсивни извиквания се получава грешка - препълване на стека

Рекурсия - итерация

- Рекурсията не винаги е необходима
- Всяка рекурсивна функция може да се пренапише без рекурсия
- Итеративната версия на функция с използване на цикли
- Рекурсивната версия като код е по-просто написана и по кратка
- Като времетраене е по-дълга от итеративния вариант и ползва повече памет
- Рекурсивната функция прави живота на програмиста по-лек и генерира лесно разбираем код

Итерационен вариант на write_vertical

```
//Uses iostream:
void write_vertical(int n)
{
    int tens_in_n = 1;
    int left_end_piece = n;
    while (left_end_piece > 9)
    {
        left_end_piece = left_end_piece/10;
        tens_in_n = tens_in_n*10;
    }
    //tens_in_n is a power of ten that has the same number
    //of digits as n. For example, if n is 2345, then
    //tens_in_n is 1000.

    for (int power_of_10 = tens_in_n;
            power_of_10 > 0; power_of_10 = power_of_10/10)
    {
        cout << (n/power_of_10) << endl;
        n = n % power_of_10;
    }
}
```

Рекурсивни функции, връщащи стойност

- double x=pow(2.0,3.0)
- Имаме нужда от функция, работеща с целочислени стойности int y=power(2,3)
- Дефиниция на функцията power
- x^n is equal to $x^{n-1} * x$ ⟶ на C++ ⟶ `power(x, n - 1)*x`
- Ако n е 0, power(x,n) връща 1 ($x^0=1$)
- Какво става, ако извикването на функцията е power(2,0)

Рекурсивна функция power

```
//Program to demonstrate the recursive function power.
#include <iostream>
#include <cstdlib>
using namespace std;
int power(int x, int n);
//Precondition: n >= 0.
//Returns x to the power n.
int main( )
{
    for (int n = 0; n < 4; n++)
        cout << "3 to the power " << n
             << " is " << power(3, n) << endl;
    return 0;
}
//uses iostream and cstdlib:
int power(int x, int n)
{
    if (n < 0)
    {
        cout << "Illegal argument to power.\n";
        exit(1);
    }
    if (n > 0)
        return ( power(x, n - 1)*x );
    else // n == 0
        return (1);
}
```

```
3 to the power 0 is 1
3 to the power 1 is 3
3 to the power 2 is 9
3 to the power 3 is 27
```

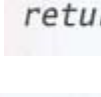
При power(2,0) няма рекурсия и се връща стойност 1

При power(2,1) има рекурсивно извикване на функцията

```
return ( power(x, n - 1)*x );
```



```
return ( power(2, 0)*2 );
```

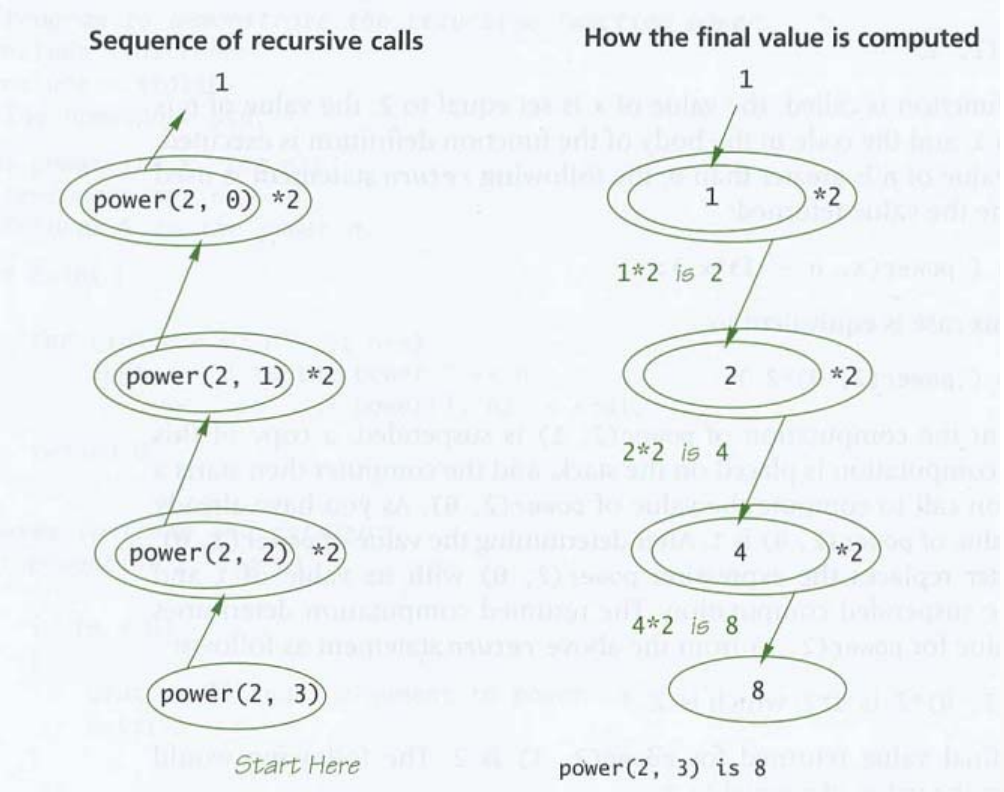


```
power(2, 0)*2 is 1*2, which is 2.
```

Рекурсивна функция power

```
cout << power(2, 3);
```

```
power(2, 3) is power(2, 2)*2
  power(2, 2) is power(2, 1)*2
    power(2, 1) is power(2, 0)*2
      power(2, 0) is 1 (stopping case)
```



Изчисляване на факториел

Рекурсивна версия:

```
main()
{
  int i = 3;                // 1
  cout << f(i) << endl;     // 2
}

int f(int a1)
{
  if (a1 <= 1)              // 3
    return 1;               // 4
  else                      // 5
    return a1 * f(a1 - 1);  // 6
}
```

Итерационна версия:

```
int fact(int n)
{
  int i;
  int prod = 1;

  for (i = 1; i <= n; i++)
    prod *= i;

  return prod;
}
```

Изчисляване на сума на списък от числа

```
#include <iostream>
#include <cstdlib>
using namespace std;
const int N=10;
int mas[N];
int have_sum(int chislo)
  { if (chislo>0) return have_sum(chislo-1)+mas[chislo];
      else return(mas[0]); //дъно на рекурсията
  }
int main()
 {int c; cout<<“Начални стойности на масива:“;
for(c=0;c<N;c++)
{ mas[c]=rand()%50;
  cout<<mas[c]<<“ “;}
cout<<endl<<“Сума:“;
cout<<have_sum(N)<<endl;
system(“pause”);
return 0;
}
```