Функции, Рекурсия

гл.ас. д-р. Нора Ангелова

Функции

```
const int N = 2;

void printMatrix(int matrix[][N], int n, int m) {
  for(int i=0; i<n; i++)
    for(int j=0; j<m; j++)
      cout << matrix[i][j] << " ";
}

int main() {
  int matrix[3][2] = {{1, 2}, {3, 4}, {5, 6}};
  printMatrix(matrix, 3, 2);

  return 0;
}
```

Функции

```
const int SET_LENGTH = 10;
int randomSet[SET_LENGTH];

int* getRandomSet() {
  for (int i = 0; i < SET_LENGTH; ++i) {
      randomSet[i] = rand();
  }

  return randomSet;
}

int main()
{
  int * temp = getRandomSet();

  for(int i=0; i<SET_LENGTH; i++) {
    cout << temp[i] << " ";
  }

  return 0;
}
```

Функции

```
const int SET_LENGTH = 10;

int* getRandomSet() {
  int randomSet[SET_LENGTH]; //ще се разруши при излизане

  for (int i = 0; i < SET_LENGTH; ++i) {
      randomSet[i] = rand();
  }

  return randomSet;
}

int main()
{
  int * temp = getRandomSet();

  for(int i=0; i<SET_LENGTH; i++) {
    cout << temp[i] << " "; //недефинирани стойности
  }

  return 0;
}
```

Задача

Да се напише функция, която сравнява лексикографски два символни низа.

```
int strcmp1(const char *str1, const char* str2) {
  while(*str1 && *str1 == *str2) {
    ++str1;
    ++str2;
  }
  return *str1 - *str2;
}
```

Задача

Да се напише функция, която намира дължината на най-дългия общ префикс на два символни низа.

```
int maxCommonPreffix(const char *str1, const char* str2) {
  int i = 0;
  while(str1[i] && str2[i] && str1[i] === str2[i])
    i++;
  return i;
}
```

Указател към функция

<тип_на_функция>(*<указател_към_функция>)(<формални_параметри>)
[=<име_на_функция>]$_{опц}$

- <указател_към_функция> - идентификатор.
- <име_на_функция> - идентификатор, означаващ име на функция от тип <тип_на_функция> и параметри - (<формални_параметри>).
- <тип_на_функция>, <формални_параметри> - съответстват на дефинициите за функции

Указател към функция

Пример:

```
int maxCommonPreffix(const char *str1, const char* str2) {
  int i = 0;
  while(str1[i] && str2[i] && str1[i] == str2[i])
    i++;
  return i;
}
...
int (*ptr)(const char*, const char*) = maxCommonPreffix;

char str1[10]="adh123";
char str2[10]="adhdg";
cout << ptr(str1, str2); // 3
```

Задача

Да се напише функция, която намира стойността на

$$\sum_{\substack{i=a,\\ i \to next(i)}}^{b} f(i)$$

Указател към функция

$$\sum_{\substack{i=a,\\ i\to next(i)}}^{b} f(i)$$

```
double sum(double a, double b, double (*f)(double),
  double (*next)(double)) {
  double s = 0;
  for(double i=a; i<b + 1e-14; i=next(i)) {
    s = s + f(i);
  }

  return s;
}
```

Задача

Да се напише функция, която намира стойността на

$$\prod_{\substack{i=a,\\ i \to next(i)}}^{b} f(i)$$

Указател към функция

$$\prod_{\substack{i=a,\\ i\to next(i)}}^{b} f(i)$$

```
double prod(double a, double b, double (*f)(double),
  double (*next)(double)) {
  double s = 1;
  for(double i=a; i<b + 1e-14; i=next(i)) {
    s = s * f(i);
  }

  return s;
}
```

Задача

Да се напише функция `accumulate`

$$f(a)\ op\ f\big(next(a)\big)\ op\ \ldots op\ f(b)$$

Указател към функция

$$f(a)\ op\ f\big(next(a)\big)\ op\ \ldots op\ f(b)$$

```
double accumulate(double (*op)(double, double),
  double null_val, double a, double b,
  double (*f)(double), double (*next)(double)) {
  double s = null_val;
  for(double i=a; i<b + 1e-14; i=next(i)) {
    s = op(s, f(i));
  }

  return s;
}
```

Оператор typedef

```
typedef <тип> <име>;
```

- <име> - идентификатор, определящ името на новия тип;
- <тип> - е дефиниция на тип;

Семантика

Определя <име> за алтернативно име на <тип>.

Оператор typedef

Пример:

```
typedef unsigned int POSITIVE_NUMBERS;

int main() {
  POSITIVE_NUMBERS a = 5;
  return 0;
}
```

Оператор typedef

Пример:

```
double next(double a) {
  return a;
}

typedef double (*TYPE)(double);

int main() {
  TYPE func = next;
  return 0;
}
```

Указател към функция

$$f(a)\ op\ f\big(next(a)\big)\ op\ \ldots op\ f(b)$$

```
typedef double (*type1)(double, double);
typedef double (*type2)(double);

double accumulate(double (*op)(double, double),
  double null_val, double a, double b,
  double (*f)(double), double (*next)(double));



double accumulate(type1 op, double null_val, double a,
  double b, type2 f, type2 next);
```

Функция като оценка

```
typedef double (*f_type)(double);
f_type getOperation(char ch) {
  switch(ch) {
    case 'a': return sin;
    case 'b': return cos;
    case 'c': return exp;
    case 'd': return log;
    default: cout << "Error!";
  }
}


int main() {
  cout << getOperation('a')(0.5);
  return 0;
}
```

РЕКУРСИЯ

Рекурсия

Ако в дефиницията на някаква функция се използва самата функция, дефиницията на функцията се нарича **рекурсивна**.

$$n! = n * (n-1) * (n-2) * \cdots * 1$$

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

Условието при $n = 0$ не съдържа обръщение към функцията факториел и се нарича **гранично**.

Рекурсия

Пример:

$$x^n = \begin{cases} 1, & n = 0 \\ x * x^{n-1}, & n > 0 \\ \dfrac{1}{x^{-n}}, & n < 0 \end{cases}$$

Гранично условието при $n = 0$.

Рекурсия

Пример:

$$1\ 1\ 2\ 3\ 5\ 8\ \ldots$$

$$fib(n) = \begin{cases} 1, & n = 1 \\ 1, & n = 2 \\ fib(n-1) + fib(n-2), & n > 1 \end{cases}$$

Гранично условието при $n = 0, n = 1$.

Рекурсия в C++

- Механизъм на рекурсия в C++ - разрешено е функция да вика в тялото си сама себе си.
- Функция, която се обръща пряко или косвено към себе си, се нарича **рекурсивна**.

- **Пряко рекурсивна функция -** в тялото на функция се извършва извикване на същата функция.

- **Непряко (косвено) рекурсивна** или **взаимно-рекурсивни-** функция A вика функция B, B вика C, а C отново вика A

Рекурсия в C++

- **Дъно на рекурсията** - гранично условие.

Един или повече случаи, които не изискват рекурсивно извикване за намиране нарешение.

Пример:
В редицата на Фибоначи – n=0 && n=1;

Рекурсия в C++

```
void func () {
  cout << "1" << endl;
}

int main() {
  cout << "0" << endl;
  func();
  cout << "2" << endl;

  return 0;
}
```

Рекурсия в C++

```
void func () {
  cout << "1" << endl;
  func(); // Зацикляне
}

int main() {

  cout << "0" << endl;
  func();
  cout << "2" << endl;



  system("pause");
  return 0;
}
```

Рекурсия в C++

```
void func (int n) {
  if (n>10) {
    cout << "11!!!" << endl;
    return;
  }
  cout << n << endl;
  func(++n);
}

int main() {
  cout << "0" << endl;
  func(1);
  cout << "2" << endl;

  return 0;
}
```

```
0
1
2
3
4
5
6
7
8
9
10
11!!!
2
```

Рекурсия в C++

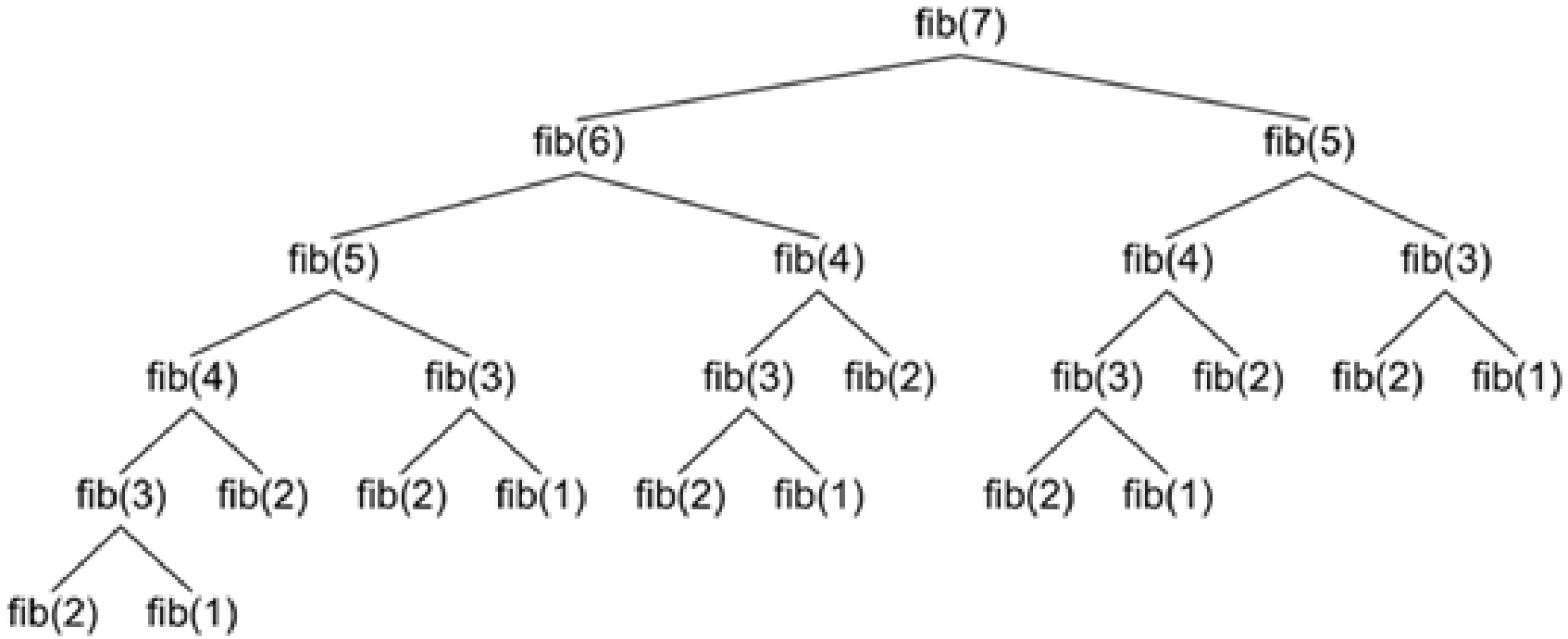
Да се напише рекурсивна функция, която пресмята n-ти член на редицата на Фибоначи.

```
int fib (int n) {
  if (n<=2)
    return 1;

  return fib(n-1) + fib(n-2);
}

int main() {
  cout << fib(5);
  return 0;
}
```

Лоша рекурсия

Редицата на Фибоначи.

- Броят на стъпките за рекурсивно изчисление на fib(100) ~ 1.6 на степен 100
- Броят на стъпките за линейно решение -100.



Memorization

Редицата на Фибоначи.

- Оптимизация – записват се пресметнатите числа в масив. Извършва се рекурсивно извикване ако числото, което пресмятаме, не е било пресметнато до момента.

```
int fibNumbers[10] = {0};

int fib (int n) { // n >= 1
  if (fibNumbers[n] == 0)
    fibNumbers[n] = fib(n-1) + fib(n-2);

  return fibNumbers[n];
}

int main() {
  fibNumbers[1] = 1;
  fibNumbers[2] = 1;
  cout << fib(5);

  return 0;
}
```

Рекурсия

Да се напише рекурсивна функция, която намира минималния елемент на редица от реални числа.

```
double min(int n, double* x) {
  double b;
  if (n == 1) return x[0];
  b = min(n-1, x);
  if (b < x[n-1]) return b;
  return x[n-1];
}
```

Рекурсия

Да се напише рекурсивна функция, която решава задачата за ханойските кули.

- Ако броят на дисковете е 0 – не се прави нищо.
- Да се преместят n-1 диска от стълб A на стълб B (същата задача с размерност n-1).
  Да се премести последният останал диск от стълб A на стълб C (нерекурсивна задача).
- Да се преместят поставените на стълб B n-1 диска на стълб C (същата задача с размерност n-1).

Рекурсия

Да се напише рекурсивна функция, която решава задачата за ханойските кули.

```
void hanoi(int n, char X, char Y, char Z) {
  if(n > 0) {
    hanoi(n-1, X, Z, Y);
    printf("move a disk from %c to %c\n",X,Z);
    hanoi(n-1, Y, X, Z);
  }
}

int main() {
  int n=3;
  hanoi(n, 'A', 'B', 'C');
  printf("the %d disks are successfully moved\n", n);

  return 0;
}
```

```
cout << “Край”;
```