Лекция 1 (04.09.2018)

1.1 Рекурсия

> Чтобы понять рекурсию, надо понять рекурсию.

Рекурсивный алгоритм для вычисления вызывает себя.

Виды рекурсии:

1. Прямая рекурсия - функция A вызывает сама себя.
2. Косвенная рекурсия - A вызывает B, A вызывает B. В одной функции обязательно должна быть хотя бы одна нерекурсивная ветка

1.2 Процесс рекурсивного вычисления

Этапы процесса рекурсивного вычисления:

1. Погружение в рекурсию.
2. Всплывание из рекурсии.

1.3 Суперкосинус

$$supercos(x, n) = \underbrace{cos...cos}_{n}(x);$$

```
1  double supercos(double x, int n) {
2      if (n == 0)
3          return x;
4      return cos(supercos(x, n-1));
5  }
```

1.4 Алгоритм Евклида

Находит наибольший общий делитель чисел a и b.

```
int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}
```

1.5 Функция Аккермана

Работает на множестве натуральных чисел.

$$A(m,n) = \begin{cases} n+1, & m = 0 \\ A(m-1,1), & m > 0, n = 0 \\ A(m-1, A(m,n-1)), & m > 0, n > 0 \end{cases}$$

1.6 Разбор строки по грамматике

Нотационные формы Бэкуса-Наура

- <цифра> ::= 0|1|...|9
- <буква> ::= a|...|z|A|...|Z|_
- <идентификатор> ::= <буква> | <идентификатор><буква> | <идентификатор><цифра>
- <оператор> ::= <выражение> | <if> | <while> | ...
- <while> ::= while(<выражение>) <оператор>
- <формула> ::= <цифра> | (<формула> <знак> <формула>)
- <знак> ::= +|-|*

Задачи:

1. Вычислить значение корректного выражения
2. Проверить корректность выражения.

Решаем первую задачу:

```
int form() {
    char c, z;
    int x, y;
    cin >> c;
    if (c >= '0' && c <= '9') return c - '0';
    x = form();
    cin >> z;
    y = form();
    cin >> c;
    if (z == '+') return x + y;
    if (z == '-') return x - y;
    if (z == '*') return x * y;
}
```

Общие принципы вычисления выражений:

1. Для каждого понятия есть своя функция.
2. Если понятие нетривиальное - вызываем его функцию.
3. Тривиальные понятия анализируем явно.

Лекция 2 (11.09.2018)

2.1 Ханойские башни

```
void abc(int n, int a, int b, int c) {
    if (n > 0) {
        abc(n-1, a, c, b);
        cout << a << " --> " << b << endl;
        abc(n-1, c, b, a);
    }
}
```

Общая идея: при $n = 1$ решение тривиально, а если мы умеем решать задачу для $n - 1$ дисков, то перенесём $n - 1$ дисков на третий стержень. Последний оставшийся диск перенесём на второй стержень. После этого перенесём все диски с третьего стержня на второй.

2.2 Быстрое возведение в степень

Неправильный код

```
int pow(int x, int n) {
    if (n == 0) return 1;
    if (n % 2 == 0) {
        return pow(x, n / 2) * pow(x, n / 2);
    }
    return pow(x, n-1) * x;
}
```

Недостатки кода: x нужно сделать глобальным, чтобы не передавать его в рекурсии, т.к. он не изменяется. Также такой код совершает $O(N)$ итераций.

Правильный код

```
int x;
int pow(int n) {
    if (n == 0) return 1;
    if (n % 2 == 0) {
        int m = pow(n / 2);
        return m*m;
    }
    return x * pow(n - 1);
}
```

Лекция 3 (18.09.18)

3.1 Перебор N-значных чисел в M-ичной системе счисления

```
// заполнить iую- позицию
void gen(i) {
    if (i > N) {
        // печать или анализ варианта
    }
    else {
        for (int j = 0; j < M - 1; ++j) {
            r[i] = j;
            gein(i + 1);
        }
    }
}
```

3.2 Перебор всех перестановок

Массив *used*[] хранит метки использованных чисел в текущей перестановке

```
void gen(i) {
    if (i > N)
        // печать
    else {
        for (int j = 1; j <= N; ++j) {
            if (!used[j]) {
                used[j] = true;
                r[i] = j;
                gen(i + 1);
                used[j] = false;
            }
        }
    }
}
```

3.3 Перебор всех сочетаний

Перебираем сочетания из N элементов по K в порядке возрастания

```
void gen(i) {
    if (i > K) {
        // печать
    }
    else {
        for (int j = r[i-1]+1; j <= N - (k-i); ++j) {
            r[i] = j;
            gen(i + 1);
        }
    }
}
```

3.4 Заливка объекта

```
// dx = {1, 1, 1, 0, 0, -1, -1, -1}
// dy = {1, 0, -1, 1, -1, 1, 0, -1}
void fill(int x, int y, int c) {
    if (r[x][y] == 1) {
        r[x][y] = c;
        for (int j = 0; j < 8; ++j) {
            fill(x + dx[j], y + dy[j], c);
        }
    }
}
```