Технология разработки программных продуктов

1. Основные этапы технологического процесса разработки программ

1. Постановка задачи.
2. Построение математической модели.
3. Разработка (выбор и адаптация) алгоритма.
4. Составление программы.
5. Тестирование и отладка.
6. Сдача в эксплуатацию.

1.1. Постановка задачи

Раскрывается организационно-экономическая сущность задачи:

- формулируется цель ее решения
- определяется взаимосвязь с другими задачами
- указывается периодичность ее решения
- раскрывается состав и форма представления входной, промежуточной и выходной информации
- характеризуются формы и методы контроля достоверности информации
- описываются формы взаимодействия пользователя с ЭВМ

Особое внимание уделяется детальному описанию входной, выходной и промежуточной информации.

При этом определяется:

- форма представления отдельных данных
- количество знаков, выделяемых для записи данных, исходя из их максимальной значности
- источник возникновения данных

Для цифровой информации указывается:

- целочисленный или дробный характер данных (для дробных указывается количество 10-х знаков) и допустимый диапазон изменения величин.
- Завершается постановка задачи описанием контрольного примера, демонстрирующего порядок решения задачи традиционным способом.
- **Основное требование к контрольному примеру** - это отражение всего многообразия возможных форм существования исходных данных.

1.2. Построение математической модели объекта.

Производится анализ и исследование задачи.

Структура этапа:

1. Анализ существующих аналогов задачи.
2. Анализ технических и программных средств.
3. Разработка математической модели.
4. Разработка структур данных.

- **Математическая модель** - это система математических соотношений (формул, уравнений, неравенств и т.д.), отражающих существенные свойства объекта или явления.
- *Математическая запись постановки задачи отличается высокой точностью отображения ее сущности, лаконичностью записи, однозначностью понимания, но она может быть выполнена не для всех задач.*

При выборе метода решения предпочтение отдается методу, который:

- Обеспечивает необходимую точность и не обладает свойством вырождения (бесконечного зацикливания).
- Позволяет использовать уже готовые стандартные программы.
- Ориентирован на минимальный объем информации.
- Наиболее быстрое получение результатов.

План написания постановки задачи (ТЗ)

- Наименование задачи.
- Назначение.
- Достигаемая цель.
- Для кого предназначена.
- Технические средства.
- Периодичность использования.
- Входная информация.
- Выходная информация (формируется по запросам).
- Метод проверки правильности (сравнивается с контрольным примером).
- Организация внедрения задачи.
- Разработка контрольного примера (входная информация с конкретными данными, выходная информация).
- Методы защиты.

2. КРИТЕРИИ КАЧЕСТВА ПРОГРАММНОГО ИЗДЕЛИЯ

- Программа является **правильной**, если она работает в соответствии с техническим заданием (ТЗ - документ, которым завершается постановка задачи).
- Программа является **точной**, если выдаваемые ею числовые данные имеют допустимые отклонения от аналогичных результатов, полученных с помощью идеальных математических зависимостей.
- Программа является **совместимой**, если она работает должным образом не только автономно, но и как часть программной системы.

- Программа является **надежной**, если она при всех входных данных обеспечивает полную повторяемость результатов.
- Программа является **универсальной**, если она правильно работает при любых допустимых вариантах исходных данных. В ходе разработки программ предусматриваются специальные средства защиты от ввода неправильных данных, обеспечивающие целостность системы.
- Программа является **защищенной**, если она сохраняет работоспособность при возникновении сбоев (режим реального времени, программа большого времени выполнения).
- Программа является **полезной**, если задача, которую она решает, представляет практическую ценность.

- Программа является **эффективной**, если объем требуемых для ее работы ресурсов ЭВМ не превышает допустимого предела.
- Программа является **проверяемой**, если ее качества могут быть продемонстрированы на практике (проверка правильности и универсальности).
  - Существуют формальные математические методы проверки и неформальные (прогоны программы с остановками в контрольных точках, обсуждение результатов заинтересованными пользователями).
- Программа является **адаптируемой**, если она допускает быструю модификацию с целью приспособления к изменяющимся условиям функционирования.

3. ПРАВИЛА ХОРОШЕГО СТИЛЯ

1. Структурное программирование предполагает использование базовых структур:

- следование
- выбор
- повторение

2. В рамках хорошего стиля нельзя явно задавать количество вводимых значений. Для этого надо использовать некоторый *признак конца ввода* (конца файла).

3. Структурное программирование сокращает потребность в комментариях. Комментарии должны содержать информацию, которую нельзя подчеркнуть в самой программе. Перед сдачей программы комментарии нужно проверить на их соответствие возможно изменившейся программе.

- Вводные комментарии содержат: номер и имя модуля; фамилию автора; дату, номер версии; назначение модуля; перечень основных алгоритмов со ссылками на источники; имена подпрограмм, вызывающих модуль; имена подпрограмм, вызываемых модулем; словарь данных; описание ввода/вывода; описание процесса обработки ошибок выполняемого модуля.

4. Имена данных должны быть мнемоническими.

Мнемоника - искусство запоминания, основанное на законах ассоциаций.

Не следует использовать слова, в которых обычно делаются орфографические ошибки; имена, различающиеся только одной буквой; слова, имеющие более одного очевидного сокращения; ключевые слова языка программирования.

Имена переменных типа **i, j, k** следует давать только управляющим переменным в операторах цикла.

5. Отступы и выравнивания в тексте программы проясняют ее логику и облегчают ее чтение. Начальной позицией называется самая левая колонка, с которой может начинаться предложение.

6. Для повышения наглядности предназначены пробелы и пустые строки, которые разделяют программу на отдельные, логически-завершенные части-параграфы.

7. Сопутствующие комментарии. Поясняют назначение каждого параграфа. Ставятся вначале параграфа.

4. ВЫБОР АЛГОРИТМА

Типы алгоритмов.

1. Если задача может быть решена прямым способом, то говорят, что она имеет **детерминированный алгоритм**.

В таких алгоритмах отсутствует элемент неопределенности, недопустимо применение метода проб и ошибок. К таким задачам относятся математические уравнения, проверка данных, печать отчетов.

2. Если решение задачи выбирается из заранее определенного множества вариантов и не может быть получено прямым методом, то такая задача имеет **недетерминированный алгоритм**.

- Для реализации таких алгоритмов используются методы проб и ошибок, повторов, откатов назад или случайного выбора.
- *К числу подобных задач относятся такие, как нахождение делителей числа, поиск кратчайшего пути, задача о восьми ферзях (найти такой способ расстановки, при котором ни один из ферзей не находился бы под угрозой других).*

3. Предназначен не для поиска ответа на поставленную задачу, а **для моделирования физических систем с помощью ЭВМ**.

5. Трудоемкость, эффективность и сложность алгоритма

- Основным фактором при выборе алгоритма для задач, решаемых с помощью перебора большого числа вариантов, является суммарное время нахождение решения.
- Методы, используемые для сокращения числа вариантов при переборе или позволяющие выбрать наиболее правдоподобные варианты, называют **эвристическими**.
- **Трудоемкость алгоритма** - это число шагов.

- Если трудоемкость ограничена **полиномом**, то алгоритм называется **эффективным**; если **более быстро растущей функцией**, то **не эффективным**.
- Зависимость времени работы программы от объема обрабатываемых данных определяется оценкой сложности алгоритма.
- Время работы алгоритма обработки массивов данных зависит от размеров этих массивов.

- Например, *время работы алгоритма выполняющего чтение и запись данных в ОЗУ* определяется по формуле **an+b**, где **a** - время, необходимое для того, чтобы прочитать или записать один элемент массива; **n** - количество элементов массива; **b** - время для выполнения вспомогательных функций.
- Поскольку эта формула выражает линейную зависимость от **n**, сложность соответствующего алгоритма называют **линейной. O(n).**

- Пример: обменная сортировка списка из n элементов представляет собой следующий процесс: определяется минимальный элемент всего списка и осуществляется его обмен с первым элементом списка; затем определяется наименьший элемент оставшегося списка и производится его обмен со вторым элементом.
- Таким образом число сравнений здесь выражается полиномом второй степени и сложность здесь будет **квадратичная. $O(n^2)$**.

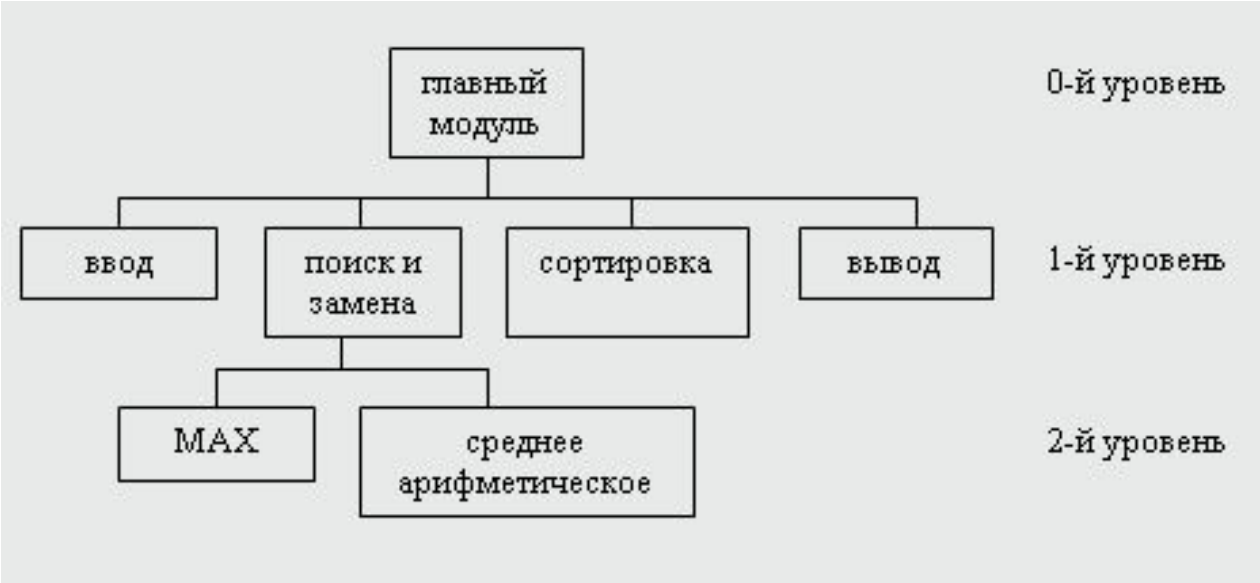
$$(n-1)+(n-2)+(n-3)+ \ldots + 3 + 2 + 1 = \frac{n(n-1)}{2} = \frac{n^2 - n}{2}$$

```
for i:=1 to n-1 do
        for j:=i+1 to n do
                if A[I] >A[j] then
                begin
                        tmp:=A[j];
                        A[j]:=A[i];
                        A[i]:=tmp;
                end;
```

```
for i:=1 to n-1 do
        for j:=1 to n do
        begin
                C[i, j]:=0;
                for k:=1 to n do
                        C[i, j]:= C[i, j]+A[i, k]*B[k, j];
        end;
```

- Алгоритм перемножения двух матриц размером **n*n** имеет число срабатываний внутреннего цикла равное n^3. $O(n^3)$.

- Алгоритм двоичного поиска в таблице с упорядоченными элементами оценивается как **$O(\log_2 n)$**.
- Логарифмическая зависимость сложности от возрастания **n** более приемлема чем линейная; линейная предпочтительнее чем полиномиальная или экспоненциальная.
- Для больших объемов данных нежелательна даже полиномиальная сложность.



- Пример: составить проект. Имеется последовательность чисел (не больше 30). Положительные четные заменить максимальным числом, отрицательные нечетные заменить средним арифметическим отрицательных чисел. Подсчитать количество замен. Упорядочить элементы последовательности в порядке убывания.

6. ИТЕРАЦИЯ И РЕКУРСИЯ

Существуют 2 основные формы повторений: итерация и рекурсия.

- **Итерация** в основном используется для тех видов обработки, которые можно определить выражением "выполнить для всех", а **рекурсия** задается выражением "выполнить тоже, что и в последний раз". Текущее действие выполняется с помощью предыдущего ответа или предыдущих стадий вычисления.
- В действительности итерация и рекурсия взаимозаменяемы.

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

```
function fact (n: integer): longint;
begin
if n=0 then fact:=1
else fact:=n*fact(n-1);
end;
begin
f:=1;
for i:=1 to n do
f:=f*i;
end.
```

- Оба алгоритма имеют линейную сложность, но для рекурсивной процедуры требуются дополнительные расходы памяти и времени, т.к. происходит многократное обращение из подпрограммы к самой себе. Должно создаваться и сохраняться много копий регистров, переменных и точек возврата. Для хранения этой информации используется стековая память, поэтому предпочтительнее итерационная форма.

7. СПОСОБЫ ОПИСАНИЯ АЛГОРИТМОВ

- Словесный.
- Графический.
- Псевдокод.
- Таблицы решений.

- **Словесный**. Действия описываются средствами естественного языка.

Достоинства:

- общедоступный
- позволяет описывать алгоритм с любой степенью детализации

Недостатки:

- отсутствие строгой формализации, т.е. разные люди могут понять по-разному
- низкая наглядность, громоздкое описание

- **Формульно-словесный**: нагляден, лаконичен, но не является строго формальным.
- **Графический**. Представляет собой изображение структуры алгоритма, при котором все этапы обработки данных представлены в виде блоков - определенных геометрических фигур.

Достоинства:

- формализован
- нагляден
- компактен

Недостатки:

- необходимость определенных знаний

- **Таблицы решений**. Применяются для разработки алгоритмов решения многовариантных расчетов с большим количеством проверок условий, определяющих выбор той или иной ветви процесса обработки информации.
- Они позволяют четко описывать саму задачу и необходимые для ее решения действия. Таблицы решений в наглядной форме определяют от каких условий зависит выбор того или иного действия.

Достоинства:

- простота отражения задачи
- компактность записи
- легкость модификации
- хорошее восприятие логики решения
- Недостатки:
- ограниченность применения.