ЛЕКЦИЯ 13

Основные этапы разработки программного обеспечения. Стили и правила программирования

Жизненный цикл программного обеспечения

Программирование - это процесс создания (разработки) программы на том или ином языке программирования.

Жизненным циклом программного обеспечения (ПО) называют период разработки и эксплуатации программного обеспечения: от момента появления идеи создания некоторого программного обеспечения до момента завершения его поддержки разработчиком.

Каскадная модель разработки ПО

Реальный процесс разработки ПО: каскадно-возвратная модель

По данной тематике существует большое число теоретических разработок, учебников и монографий, принятых международных стандартов, практических рекомендаций.

Этапы разработки программного обеспечения

Традиционные ***методы проектирования программных средств*** предполагают наличие следующих основных этапов разработки программы:

1. Спецификация (определение, формулирование требований к программе)

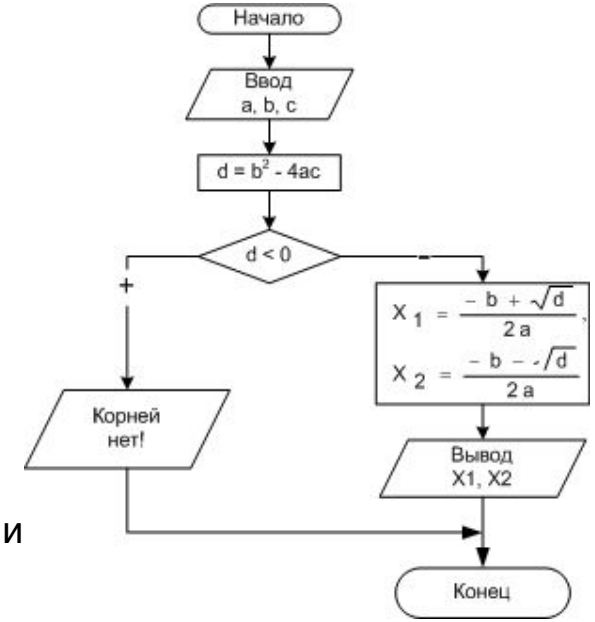
На данном этапе определяются условия задачи, подробно описывается исходная информация и формулируются требования к результату. Определяются входные, выходные и промежуточные данные, описывается поведение программы в особых случаях.

2. Разработка алгоритма решения задачи

Алгоритм - это описание точного, пошагового выполнения действий решающих поставленную задачу должным образом.

Свойства алгоритма:

- однозначность (единственность толкования правил и порядка выполнения действий)
- массовость (возможность применения алгоритма для решения класса задач, правильная работа при меняющихся в заданных пределах значениях исходных данных)
- результативность (выполнение алгоритма должно приводить к получени определенного результата)



Если задача может быть решена несколькими способами (возможны различные варианты алгоритма решения), то на основе некоторых критериев выбирается наиболее подходящее решение.
Для сложных задач целесообразно применять «метод декомпозиции» (разбиение на совокупность более простых взаимосвязанных задач).
Результатом данного этапа является подробное *текстовое описание алгоритма* или его *блок-схема*.

Этапы разработки программного обеспечения

3. Реализация алгоритма (кодирование на языке программирования)

Данный этап предполагает запись разработанного алгоритма в виде программы на выбранном языке программирования. Результатом этапа является исходный код программы.

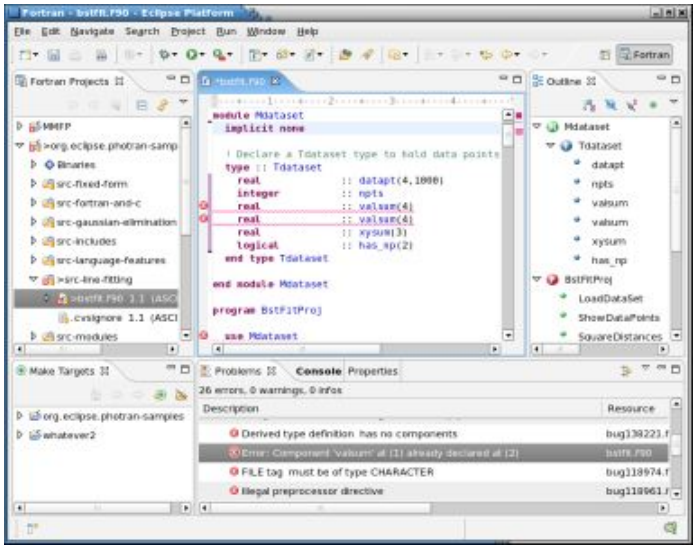
Интегрированные среды разработки включают в себя текстовый редактор, компилятор (интерпретатор), средства автоматизации сборки, отладчик. Примеры (визуальные среды): Eclipse, Microsoft Visual Studio, NetBeans, Qt Creator.

4. Отладка программы

Отладка программы - это процесс поиска, локализации и устранения ошибок в программе.
Ошибки в программе традиционно разделяют на три группы: синтаксические (ошибки в тексте), времени выполнения (выявляются на этапе выполнения) и алгоритмические. Этап отладки можно считать законченным, если программа правильно работает на нескольких наборах входных данных.

Программы-отладчики: Microsoft Visual Studio, GNU Debugger, DBX, WinDbg, TotalView.

```
#include <stdio.h>
#include <windows.h>
#include <conio.h>
#include <math.h>
/*Программа: Вычисление корней квадратного уравнения*/
main()
{
float a, b, c, d, x1, x2;
char str[50];
//Очистить экран
textbackground(4);
textcolor(15);
clrscr();
//ВВод коэффициентов a, b, c
CharToOem("Введите коэффициенты a, b, c\n", str);
printf(str);
scanf("%f %f %f", &a, &b, &c);
//Вычисление дискриминанта d
d=b*b - 4*a*c;
if (d<0) {
CharToOem("Действительных решений нет", str);
printf(str);
}
else {
x1=(-b - sqrt(d))/(2*a);
x2=(-b + sqrt(d))/(2*a);
// Вывод абсолютного значения числа x
CharToOem("\n x1=%f x2=%f", str);
printf(str, x1, x2);
};
```



Этапы разработки программного обеспечения

5. Тестирование программы

На этапе тестирования следует проверить, как ведет себя программа на как можно большем количестве входных наборов данных, в том числе и на заведомо неверных (учет ситуаций, для которых программа в принципе не предназначена). При этом целесообразно использовать некое «эталонное решение» задачи (например, полученное другим методом), которое позволяет получить заведомо верные результаты вычисления.

6. Документирование, поддержка и обновление программы

Документирование - создание текстовых и графических материалов по использованию программы в помощь пользователям и разработчикам (общее описание возможностей программы, техники использования, типовые примеры и т.д.).
Некоторые ситуации и ошибки проявляются только в процессе длительного использования программы с множеством входных данных. В этих случаях может потребоваться обновление кода программы.

Стили и правила программирования

Стиль программирования - набор приемов или методов программирования, которые используют разработчики с целью получения правильных, эффективных, удобных для использования и легко читаемых программ.

При разработке программ могут быть использованы различные стили программирования (структурное, сентенциальное, функциональное, автоматное, событийное и т.д.), наиболее соответствующие архитектуре компьютера и конкретным классам задач.

Структурное программирование - методология и технология разработки программного обеспечения, основанная на следующих принципах:

- «нисходящее» программирование (разработка начинается с определения целей решения проблемы, после чего идет последовательная детализация, расщепление на подзадачи, заканчивающаяся программой);
- модульность (разбиение программы на независимые модули).

При этом логика алгоритма и программы должны использовать три основные структуры: последовательное выполнение, ветвление и повторение.

Стили и правила программирования

При создании программного обеспечения следует помнить, что программа предназначена для:

- пользователя (надежность, удобство использования)
- разработчика (удобство внедрения в более крупные проекты, возможность доработки и переноса на другие платформы)

Правила хорошего стиля программирования - результат «явного» или «неявного» соглашения между разработчиками.

В соответствии с общепринятыми с правилами хорошего стиля программный код должен поддерживать:

- очевидную структурную логику программы и соответствовать реализуемому алгоритму;
- несущие прозрачную смысловую нагрузку имена переменных, процедур и функций;
- аккуратное форматирование (использование вертикальных и горизонтальных отступов);
- развернутые комментарии;
- отсутствие сложных для понимания приемов и нетрадиционных языковых конструкций (использование общепринятых приемов программирования).

Стили и правила программирования

Некоторые замечания о стиле программирования:

- важно придерживаться стандартов языка программирования для обеспечения возможности портирования программы (переноса на другую платформу);
- при разработке «больших программ» целесообразно использовать модульный подход (выделение содержательных функциональных частей кода в отдельные файлы);
- неоднократно повторяющиеся в программе функциональные конструкции следует выносить в отдельные процедуры и функции;
- каждую инструкцию (оператор) программы следует записывать в отдельной строке;
- операторы начала и конца циклов (структурных блоков) следует писать друг под другом, а операторы внутри циклов - сдвигать правее по отношению к ним;
- при написании сложных операторов желательно использовать пробелы;
- комментарии необходимо добавлять перед отдельными структурными блоками программы (занимают всю строку), а в специальных случаях - правее конкретных операторов, избегая при этом избыточности.

Стили и правила программирования

Некоторые замечания о стиле программирования (продолжение):

- следует, по-возможности, избегать нецелевого или избыточного выделения памяти (повторно использовать память для служебных переменных);
- все переменные в программе желательно объявлять явно с указанием их типа, давая им, по-возможности, содержательные имена, соответствующие их функции;
- следует соблюдать аккуратность при кодировании операций с переменными разных типов;
- для предотвращения потенциальных ошибок переменные следует инициализировать;
- использовать константы для не меняющих свое значение величин, участвующих во многих вычислениях в программе.

```
!*******************************************************************
!   pi3f90.f - compute pi by integrating f(x) = 4/(1 + x**2)
!*************************************************************************
program main

 double precision  PI25DT
 parameter         (PI25DT = 3.141592653589793238462643d0)
 double precision pi, h, sum, x, f, a
 integer n, i

!  function to integrate
 f(a) = 4.d0 / (1.d0 + a*a)

!  read the number of intervals
       write(*,*) 'Enter the number of intervals: (0 quits)'
       read(*,*) n

!  calculate the interval size
       h = 1.0d0/n
       sum  = 0.0d0

!  calculate the integral
       do i = 1, n
          x = h * (dble(i) - 0.5d0)
          sum = sum + f(x)
       enddo

       pi = h * sum

!  output the results

       write(*, *) 'pi is approximately: ', pi, ' Error is: ', abs(pi - PI25DT)

stop

end
```

Стили и правила программирования

Некоторые замечания о тестировании и отладке программ:

- весьма полезно и эффективно на этапе отладки и тестирования использовать специальные программы-отладчики (debuggers);
- для компиляции программ целесообразно использовать специальные утилиты и скрипты (make);
- ощутимый выигрыш в скорости работы программы может дать правильная установка опций компилятора (оптимизация), а также использование эффективных компиляторов;
- полезно выполнять проверку входных данных программы для минимизации потенциальных ошибок времени исполнения.

Вопросы использования готовых библиотек подпрограмм

При написании программных кодов в области вычислительной математики и математического моделирования целесообразно использовать свободно распространяемые библиотеки подпрограмм, обладающие высокой надежностью и вычислительной эффективностью.

Наиболее известной коллекцией готовых программ, распространяемых в виде исходных и бинарных кодов, является проект «Netlib Repository» (*http://www.netlib.org).*

Примеры библиотек:

- ✔ LAPACK - Linear Algebra PACKage (библиотека подпрограмм, реализующих алгоритмы линейной алгебры)
- ✔ BLAS - Basic Linear Algebra Subprograms (библиотека подпрограмм, реализующих операции с векторами и матрицами)
- ✔ The ScaLAPACK Project (параллелизация кодов вычислительной математики)
- ✔ FN (библиотека подпрограмм, эффективно реализующих типовые математические функции - тригонометрические, логарифмические, Бесселя и др.)

Подпрограммы из этих библиотек, написанные преимущественно на языке Fortran и реализующие те или иные типовые численные алгоритмы, можно вызывать из собственных программ, придерживаясь специальных правил, отраженных в сопровождающей документации и примерах.