

Основы программной инженерии (ПОИТ)
Технологии разработки программного обеспечения (ИСИТ)

Основные этапы разработки программ

План лекции:

- система программирования, язык программирования;
- алфавит, основные элементы языка программирования;
- символы времени трансляции, символы времени выполнения;
- этапы и цели разработки программы;
- трудоемкость этапов разработки программ.

1. Структура языка программирования



2. Компилятор CL:

исходный код C++ на ASCII, Windows-1251.

Стандарт C++: исходной код основывается на множестве символов ASCII:

Алфавит языка C++

буквы латинского алфавита:

[a...z], [A...Z]; цифры [0...9];

спецсимволы:

_{}[]()#<>.:;%.?*-+~/^&~!=, ' " @ \$

пробельные символы:

пробел, символы табуляции, символы перехода на новую строку.

Дополнительные символы *времени выполнения* определяются **setlocale**.

По умолчанию, локаль: **SetLocale (LC_ALL, "C")** устанавливает стандартный контекст C.

Во время выполнения можно установить кодовую страницу языкового стандарта, используя вызов **setlocale(LC_CTYPE, "rus")**

или

воспользоваться следующими функциями, необходимо включить заголовочный файл `<windows.h>`:

```
#include <windows.h> // windows.h содержит прототипы функций
SetConsoleOutputCP(1251); //установить кодовую таблицу, на поток ввода
SetConsoleCP(1251); //установить кодовую таблицу, на поток вывода
```

Директива **#pragma** позволяет указать целевой языковой стандарт во время компиляции. Это гарантирует, что строки с расширенными символами будут сохраняться в правильном формате.

Алфавит языка программирования служит для построения слов в языке программирования, которые называют лексемами. Примеры лексем:

Лексемы

идентификаторы;

ключевые (зарезервированные) слова;

знаки операций;

константы;

разделители (скобки, знаки операций, точка, запятая, пробельные символы и т.д.).

Границы лексем определяются с помощью других лексем, таких, как разделители или знаки операций.

3. Идентификатор:

Идентификатор – имя компонента (объекта) программы (переменной, функции, метки, типа и пр.), составленное программистом по определенным правилам.

Примеры правил составления идентификаторов в языках программирования:

C/C++	<p>начинаются с буквы или подчеркивания;</p> <p>не совпадают с ключевыми словами C++ или с именами библиотечных функций;</p> <p>могут состоять из любого количества символов, но компилятор гарантирует, что будет считать значащими только 31 первых символов идентификаторов, не имеющих внешней связи;</p> <p>идентификаторы чувствительны к регистру.</p>
Ruby	<p>начинаются с буквы или специального модификатора. имена локальных переменных начинаются со строчной буквы или знака подчеркивания (alpha, _ident);</p> <p>имена глобальных переменных начинаются со знака доллара (\$beta);</p> <p>имена переменных экземпляра (принадлежащих объекту) начинаются со знака «@» (@foobar);</p> <p>имена переменных класса (принадлежащих классу) предваряются двумя знаками «@@» (@@not_const);</p> <p>имена констант начинаются с прописной буквы (K6chip);</p> <p>в именах идентификаторов знак подчеркивания «_» можно использовать наравне со строчными буквами (\$not_const);</p> <p>имена специальных переменных, начинающиеся со знака «\$» (\$beta).</p>
MS Transact-SQL	<p>имена переменных должны начинаться с символа @</p>
Python	<p>используются символы Unicode.</p> <p>начинаются с латинской буквы в любом регистре или символа подчёркивания, могут содержать цифры.</p> <p>не совпадают с ключевыми словами.</p> <p>Имена, начинающиеся с символа подчёркивания, имеют специальное значение.</p>

Структура языка программирования

- ✓ **алфавит языка:** кодировка символов; символы времени трансляции, символы времени выполнения;
- ✓ **идентификаторы:** правила образования идентификаторов; зарезервированные идентификаторы; литералы; ключевые слова;
- ✓ **фундаментальные (встроенные) и пользовательские типы данных:**
 - предопределенные типы данных, массивы фундаментальных типов;
 - типы, которые может создавать пользователь на основе фундаментальных типов (возможно описание их свойств и поведение);
- ✓ **преобразование типов:** явное и неявное (автоматическое).
- ✓ **инициализация памяти:** присвоение значения в момент объявления переменной;
- ✓ **константное выражение:** выражение, которое должно быть вычислено на этапе компиляции;
- ✓ **область видимости переменных:** доступность переменных по их идентификатору в разных частях программы; пространства имен;
- ✓ **выражения**
- ✓ **инструкции языка:** инструкция — это некое элементарное действие, несколько идущих подряд инструкций образуют блок вычислений (последовательность инструкций);
 - присваивания;
 - инструкции объявления;
 - блок вычислений;
 - ветвление;
 - циклы;
 - инструкции перехода;
 - обработка исключений;
- ✓ **программные конструкции** (декомпозиция программного кода): процедуры, функции, методы, ...

4. Этапы и цели разработки программы:

1. Постановка задачи. <ul style="list-style-type: none">• определение функциональных возможностей программы;• подготовка технического задания
2. Выбор метода решения. <ul style="list-style-type: none">• определение исходных и выходных данных, ограничений на них;• выполнение формализованного описания задачи;• построение математической модели, для решения на компьютере.
3. Разработка алгоритма решения задачи. <ul style="list-style-type: none">• выполняется на основе ее математического описания;• полное и точное описание, определяющее вычислительный процесс, ведущий от начальных данных к искомому результату.
4. Написание программы на языке программирования (кодирование) <ul style="list-style-type: none">• запись алгоритма на языке программирования.
5. Ввод программы в компьютер <ul style="list-style-type: none">• подготовка исходного кода программы в виде текстового, который поступает на вход транслятора.
6. Трансляция <ul style="list-style-type: none">• преобразование исходного кода с одного языка программирования в семантически эквивалентный код на другом языке;• получение объектного модуля.
7. Компоновка <ul style="list-style-type: none">• объединение одного или нескольких объектных модулей программы и объектных модулей статических библиотек в исполняемую программу;• связывание вызовов функций и их внутреннего представления (кодов), расположенных в различных модулях;• получение исполняемого (загрузочного) файла.
8. Выполнение <ul style="list-style-type: none">• выполнение исполняемого файла программы на целевой машине.
9. Отладка <ul style="list-style-type: none">• обнаружение, локализация и устранение ошибок.
10. Тестирование <ul style="list-style-type: none">• подготовка тестовых наборов данных для проверки поведения программы на соответствие предъявляемым к ней требованиям.
11. Документирование <ul style="list-style-type: none">• создание пользовательской документации.
12. Эксплуатация <ul style="list-style-type: none">• выполнение в предназначенной для этого среде в соответствии с пользовательской документацией
13. Модификация (Реинжиниринг) <ul style="list-style-type: none">• внесение изменений в целях повышения производительности или адаптации к изменившимся условиям работы или требованиям.
14. Снятие с эксплуатации <ul style="list-style-type: none">• завершение жизненного цикла ПП и изъятие его из эксплуатации.

Трудоемкость этапов

Этапы	Трудозатраты	Ошибки	
		Появление	Выявление
Постановка задачи	10%	40-46%	50%
Математическая формулировка			
Выбор метода решения			
Составление алгоритма	20%	35-38%	
Написание программы на языке программирования	15%		
Ввод программы в компьютер	5%	5-10%	
Выполнение программы			
Тестирование	40%		45%
Отладка			
Документирование	10%		3%
Эксплуатация			
Реинжиниринг			

«Для решения любой сколь угодно простой задачи можно написать программу, которая будет работать сколь угодно медленно». Афоризм.

4. Основные этапы разработки программ

Программа – логически упорядоченная последовательность команд, необходимых для решения определенной задачи.

Программа – алгоритм, записанный на языке программирования.

Текст программы (исходный код) – полное законченное и детальное описание алгоритма на языке программирования.

1) Постановка задачи (ответственность исполнителя).

Постановка задачи	<i>точная формулировка условий задачи с описанием входной и выходной информации, описание поведения программы в особых случаях</i>
Описание входной информации	<i>точное описание всех исходных данных, которые вводятся пользователем</i> <ul style="list-style-type: none">○ синтаксис (формат данных);○ семантика (назначение, тип, допустимые значения, область изменения, ...)
Описание выходной информации	<i>точное описание результатов, формируемых программой</i> <ul style="list-style-type: none">○ синтаксис и семантика выходных данных;○ сообщений об ошибках;○ протокола вычислительного процесса;○ реакции программы на некорректность исходных данных;○ и т.п.
Дополнительные сведения о программе	<i>ограничения:</i> <ul style="list-style-type: none">○ на используемую память;○ длину программы;○ время ее работы; <i>идеи относительно внутреннего проектирования функций (если это необходимо);</i> <i>описание функций преобразования информации, выполняемых программой.</i>

Пример

Цель: ознакомиться с основами кодирования информации;
освоить кодировки ASCII, Windows-1251.

Среда разработки: создать приложение на языке программирования C++ в интегрированной среде разработки Visual Studio.

Задача: по коду символа, введенного с клавиатуры, определить, является этот символ цифрой, буквой латинского либо русского алфавита или другим символом.

Вывести в консоль символ, информацию о принадлежности символа к одной из категорий, его код в соответствующей кодировке ASCII или Windows-1251.

Входная информация: программа принимает один символ из стандартного входного потока.

Выходная информация: выводит в стандартный поток вывода введенный символ, категорию, к которой он принадлежит, и код этого символа с указанием соответствующей кодировки.

2) Формализация задачи.

На этом этапе создается описательная информационная модель, созданная на этапе постановки задачи, выраженная каким-либо формальным языком, например, математическими формулами, адаптированными для решения данной задачи.

3) Разработка алгоритма решения задачи

Алгоритм (лат. algorithmi – от имени Аль-Хорезми, узбекского математика, астронома, IX в.) – совокупность точно заданных правил, с помощью которой можно получить решение задачи за конечное число шагов.

Алгоритм

*точное предписание, определяющее
вычислительный процесс, ведущий от начальных
данных к искомому результату*

Кнут Д.Э. Искусство программирования. Том 1. Основные алгоритмы, 2006г.

Алгоритм

*конечный набор правил, который определяет
последовательность операций для решения конкретного
множества задач и обладает пятью важными чертами:
конечность, определённость, ввод, вывод, эффективность.*

Колмогоров А.Н. Теория информации и теория алгоритмов. Изд. 1987г.

Алгоритм

всякая система вычислений, выполняемых по строго определённым правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи.

Марков А.А. Теория алгоритмов. (1954г.) Изд. 1984г.

Алгоритм

точное предписание, определяющее вычислительный процесс, идущий от варьируемых исходных данных к искомому результату.

ГОСТ 19.004–80

Программа

алгоритм, записанный в форме, воспринимаемой вычислительной машиной.

ГОСТ 19.004–80. Процесс составления программы.

Программирование

это также раздел прикладной математики, разрабатывающий методы использования вычислительных машин для реализации алгоритмов

Свойства алгоритмов

дискретность (возможность разбиения на шаги);
понятность (ориентирован на исполнителя);
определенность (однозначность толкования инструкций);
конечность (возможность получения результата за конечное число шагов);
массовость (применимость к некоторому классу объектов);
эффективность (оптимальность времени и ресурсов, необходимых для реализации алгоритма).

Процесс алгоритмизации

разложение всего вычислительного процесса на отдельные шаги
установление взаимосвязей между отдельными шагами алгоритма и порядка их следования
полное и точное описание содержания каждого шага
проверка правильности составленного алгоритма

<p>Способы описания алгоритмов</p>	<p><i>словесно-формульный (на естественном языке);</i> <i>графический (структурный или блок-схемой);</i> <i>использование псевдокода (специальных алгоритмических языков);</i> <i>с помощью сетей Петри;</i> <i>программный.</i></p>
---	--

Словесно-формульный способ

Пример:

Задача. По коду символа, введенного с клавиатуры, определить, является этот символ цифрой, буквой латинского либо русского алфавита или другим символом. Вывести в консоль информацию, к какой категории символов он принадлежит, и его код в соответствующей кодировке ASCII или Windows- 1251.

Словесно-формульным способом алгоритм решения этой задачи может быть записан в следующем виде:

1. Ввести символ
2. Если код символа попадает в диапазон от 30 в шестнадцатеричной системе счисления (0x30) до 39 в шестнадцатеричной системе счисления (0x39) включительно, то п.3, в противном случае п.5.
3. Вывести «Это цифра», символ цифры, ASCII, код символа в таблице ASCII.
4. Перейти к п.12 (конец).
5. Иначе: если код символа попадает в диапазон от 41 в шестнадцатеричной системе счисления (0x41) до 7A в шестнадцатеричной системе счисления (0x7A) включительно, то п.6, в противном случае п.8.
6. Вывести «Это латинская буква», символ буквы, ASCII, код символа в таблице ASCII.
7. Перейти к п.12 (конец).
8. Иначе: если код символа попадает в диапазон от 0xC0 до 0xFF включительно, то п.9 в противном случае п.11.
9. Вывести «Это русская буква», символ буквы, Windows- 1251, код символа в таблице Windows- 1251.
10. Перейти к п.12 (конец).
11. Вывести «Это не цифра и не буква», символ, код символа в таблице Windows- 1251
12. КОНЕЦ.

Блок-схемы

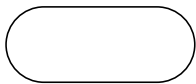


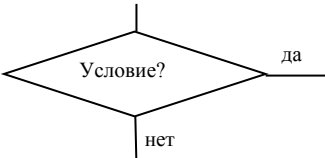

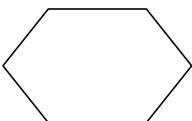


Выработаны соглашения для изображения схем-алгоритмов и закреплены ГОСТ и международными стандартами.

Единая система программной документации (ЕСПД), частью которой является Государственный стандарт — **ГОСТ 19.701-90 «Схемы алгоритмов программ, данных и систем»**.

Рассматриваемый ГОСТ 19.701-90 практически полностью соответствует международному стандарту ISO 5807:1985.

Условные графические изображения, используемые для составления блок-схем, называют символами.

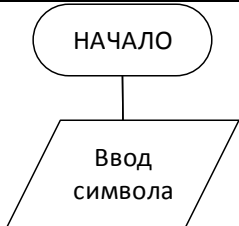
Основные элементы схем алгоритма

 Блок начала-конца алгоритма	 Блок ввода-вывода данных	 Блок вычислений	 Условный блок
 Предопределенный процесс	 Блок подготовки ()	 Комментарий	 Соединитель (ссылка на текущую страницу при разрыве схемы)

Типы процессов

Линейные процессы.

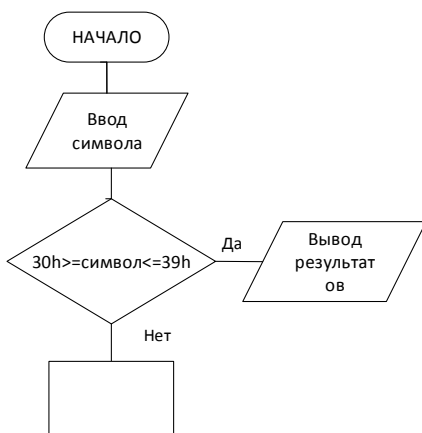
направление вычислений не зависит от значения исходных данных и получаемых в результате решения задачи промежуточных результатов.

1. Ввести символ		<pre>cout << "Введите символ "; cin >> code;</pre>
------------------	---	--

Разветвляющиеся процессы.

вычислительные процессы, в которых в зависимости от значения некоторого признака проводятся вычисления по одному из нескольких возможных направлений, называются ветвящимися (разветвляющимися).

2. Если код символа попадает в диапазон от 3016 до 3916 включительно, то п.3, в противном случае п.5.



```
if (code >= '0' && code <= '9')
{
    printf("Это цифра %c, код ASCII = %X", code, code);
    cout << endl;
}
else
```

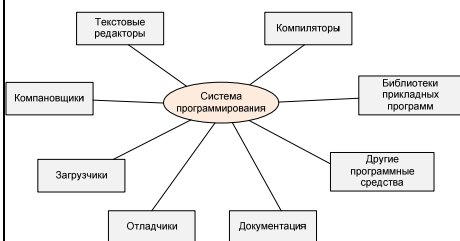
4) Кодирование

Кодирование:

запись разработанного алгоритма в виде программы на выбранном языке программирования.

Результатом этапа является исходный код программы на ЯП.

Система программирования: комплекс программных средств, предназначенных для автоматизации процесса разработки, отладки ПО и подготовки программного кода к выполнению



```
#include <windows.h>
#include <iostream>

int main()
{
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
    using namespace std;
    unsigned char code;
    cout << "Введите символ ";
    cin >> code;
    if (code >= '0' && code <= '9')
    {
        printf("Это цифра %c, код ASCII = %X", code, code);
        cout << endl;
    }
    else if (code >= 'A' && code <= 'z')
    {
        printf("Это латинская буква %c, код ASCII = %X", code, code);
        cout << endl;
    }
    else if (code >= 0xC0 && code <= 0xFF)
    {
        printf("Это русская буква %c, код Windows-1251 = %X", code, code);
        cout << endl;
    }
    else
    {
        printf("Это не цифра и не буква <%c>, код = %X", code, code);
        cout << endl;
    }
    system("pause");
    return 0;
}
```

5) Ввод программы в компьютер

Инструментальные средства программирования:

Текстовый редактор

компонента системы программирования (или IDE) – программа, позволяющая подготовить исходный код программы

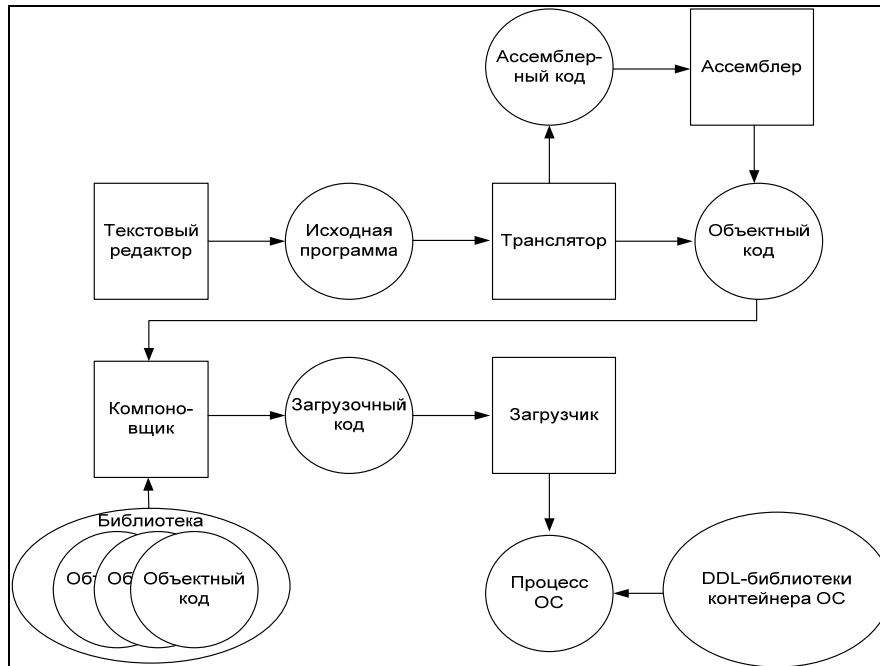
Интегрированная среда разработки (integrated development environment - IDE)

набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла разработки программы

Примеры IDE (визуальные среды):

Eclipse, Microsoft Visual Studio, NetBeans, Qt Creator, ...

Общая схема системы программирования:

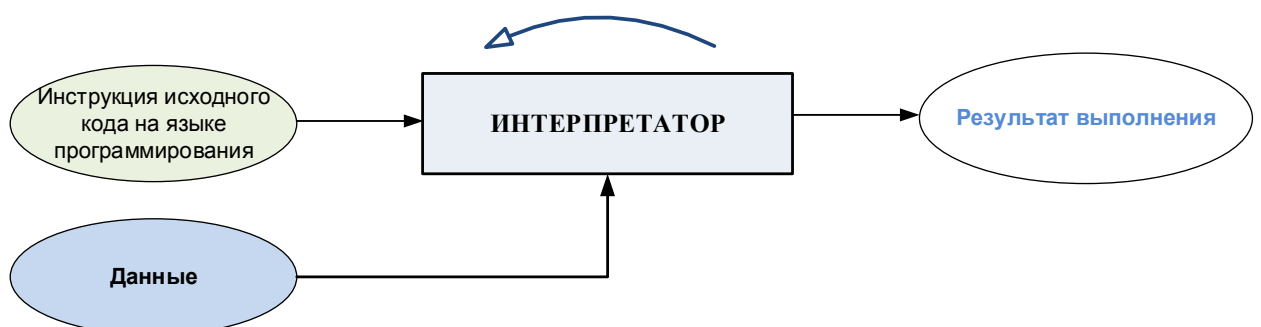


6) Компиляция

Компилятор (транслятор) – программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке; результат – объектный модуль.

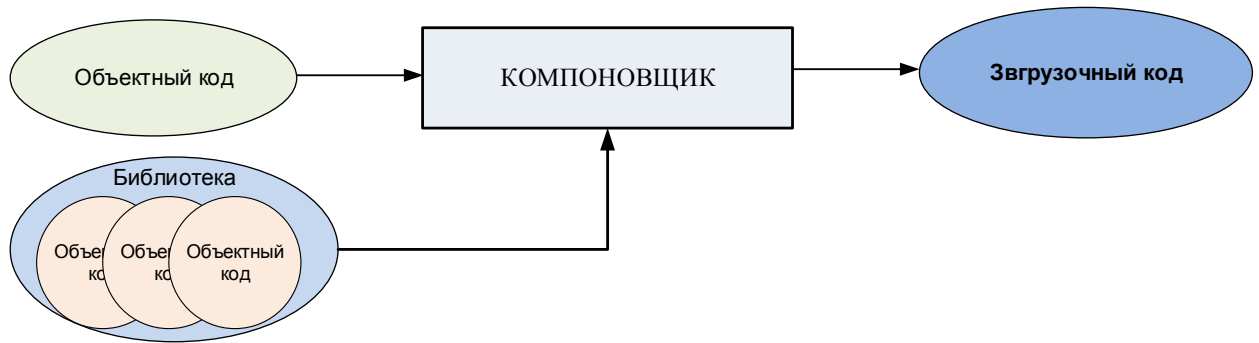


Интерпретатор – разновидность транслятора. Переводит и выполняет программу с языка высокого уровня в машинный код строка за строкой.



7) Компоновка

Компоновщик (linker, редактор связей) – программа, принимающая один или несколько объектных модулей и формирующая на их основе загрузочный модуль.



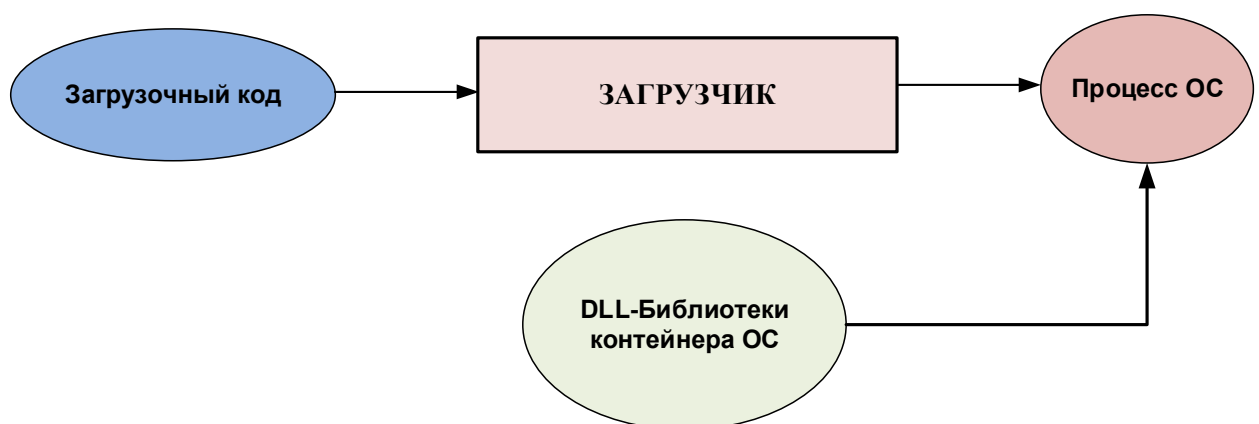
Если программа состоит из нескольких объектных файлов, компоновщик собирает эти файлы в единый исполнимый модуль, вычисляя и подставляя адреса вместо неопределенных внешних имен, в течение **времени компоновки** (статическая компоновка) или во **время исполнения** (динамическая компоновка).

8) Выполнение исполняемого файла программы на целевой машине

Загрузочный код – результат работы компоновщика.

Один файл загрузочного кода – загрузочный модуль.

Загрузчик (loader) – программа, обычно входящая в состав операционной системы, предназначенная для запуска процесса операционной системы на основе загрузочного модуля.



9) Отладка программы

Отладка программы – процесс поиска, локализации и устранения ошибок в программе.

Отладчик (debugger) – компонента системы программирования (или IDE) – программа, позволяющая контролировать ход выполнения программы (приостанавливать, выполнять пошагово), просматривать и изменять области памяти и т.п.

Ошибки в программе можно разделить на три группы:

- синтаксические (ошибки в исходном коде программы);
- времени выполнения (выявляются на этапе выполнения);
- 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.

10) Тестирование программы

Тест – это набор конкретных значений исходных данных, при которых известен ожидаемый результат работы программы.

На этапе тестирования и отладки проверяется, работает ли программа, если работает, то правильно ли. Проверяется отсутствие ошибок в программе.

С этой целью выполняется проверка поведения программы на большом количестве входных наборов данных, в том числе и наборах заведомо неверных данных (учет ситуаций, для которых программа в принципе не предназначена).

Если результаты работы программы соответствуют ожидаемым – значит задача решена, иначе – на одном из этапов допущена ошибка.

На этапе тестирования и отладки требуются как знания по предметной области, так и знание основ программирования. Так как без знаний в предметной области мы не можем знать результирующих данных в тестах, а без знаний в программировании мы не сможем отыскать ошибки и составить наиболее полный набор тестов, учитывающий все частные случаи и исключения.

11) 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.).

12) Эксплуатация

Выполнение операций, выполняемых персоналом, эксплуатирующем систему в предназначенной для этого среде в соответствии с пользовательской документацией.

13) Модификация (Реинжиниринг)

Реинжиниринг – это модификация программного продукта при необходимости исправить ошибки, выявленные в процессе эксплуатации, модернизировать или адаптировать программу к изменившимся требованиям.

Некоторые нештатные ситуации и ошибки проявляются только в процессе длительного использования программы с множеством входных данных. В этих случаях может потребоваться обновление кода программы.

14) Снятие с эксплуатации

Завершение жизненного цикла ПП и изъятие его из эксплуатации