Езици за програмиране

Елисавета Гурова
Десислава Петрова

Курс “Компютърни системи и технологии”

Какво ще научите …

- Какво е език за програмиране
- Езици от ниско ниво и асемблерни езици
- Езици от високо ниво
- Недостатъци на по-рано появилите се езици
- Популярни езици за програмиране
- Фази на жизнения цикъл при разработка на програми (ЖЦ)

Езици за програмиране

- **Дефиниция:** изкуствен език, използван за създаване на програми, които управляват поведението на машина, обикновено компютър.
  - Всеки език притежава синтаксис и семантика, описващи съответно неговата структура и значение.
  - Жизнен цикъл са последователните фази при разработка на програма
- **Предназначение:** използват се за писане на програми, които карат компютъра да извършва определени изчисления или алгоритми, както и за управление на външни устройства като принтери, роботи и др.
- **Целева група:** използват се за комуникация посредством инструкции между човека и компютъра, както и за управление на дадено устройство от друго.

Синтаксис

- **Текстови програмни езици:** използват думи, цифри и пунктуационни знаци
- **Графични програмни езици:** използват специални взаимовръзки между символи
- Синтаксисът на програмния език описва допустимата комбинация от символи, формиращи синтактично правилна програма.
- Лексиката на програмния език се представя посредством регулярни изрази
- Примерна граматика на Lisp:

```
expression ::= atom | list
atom  ::= number | symbol
number  ::= [+-]?['0'-'9']+
symbol  ::= ['A'-'Z''a'-'z'].*
list  ::= '(' expression* ')‘
```

- Не всички синтактично правилни програми са семантично коректни:

```
complex *p = NULL;
complex abs_p = sqrt (p->real * p->real + p->im * p->im);
```

*Тъй като p е NULL указател, операциите p->real и p->im нямат смисъл!

Семантика и типова система

- Семантиката дефинира правилата за конструиране на семантично правилни конструкции.
  - Най-важните от тези правила се определят от типовата система.
  - Типът определя значението и предназначението на стойност или множество от стойности.
- Типизирани и нетипизирани езици
  - Типизиран език: спецификацията на всяка операция дефинира типа данни, върху които е приложима.
  - Нетипизиран език: позволяват изпълнение на операция върху произволни данни, които обикновено са последователности от битове с различна дължина (асемблерни езици).
  - Еднотипов език: притежават един единствен тип, най-често символни низове, които служат за представяне на числови и символни данни (скриптови езици и езици за форматиране).

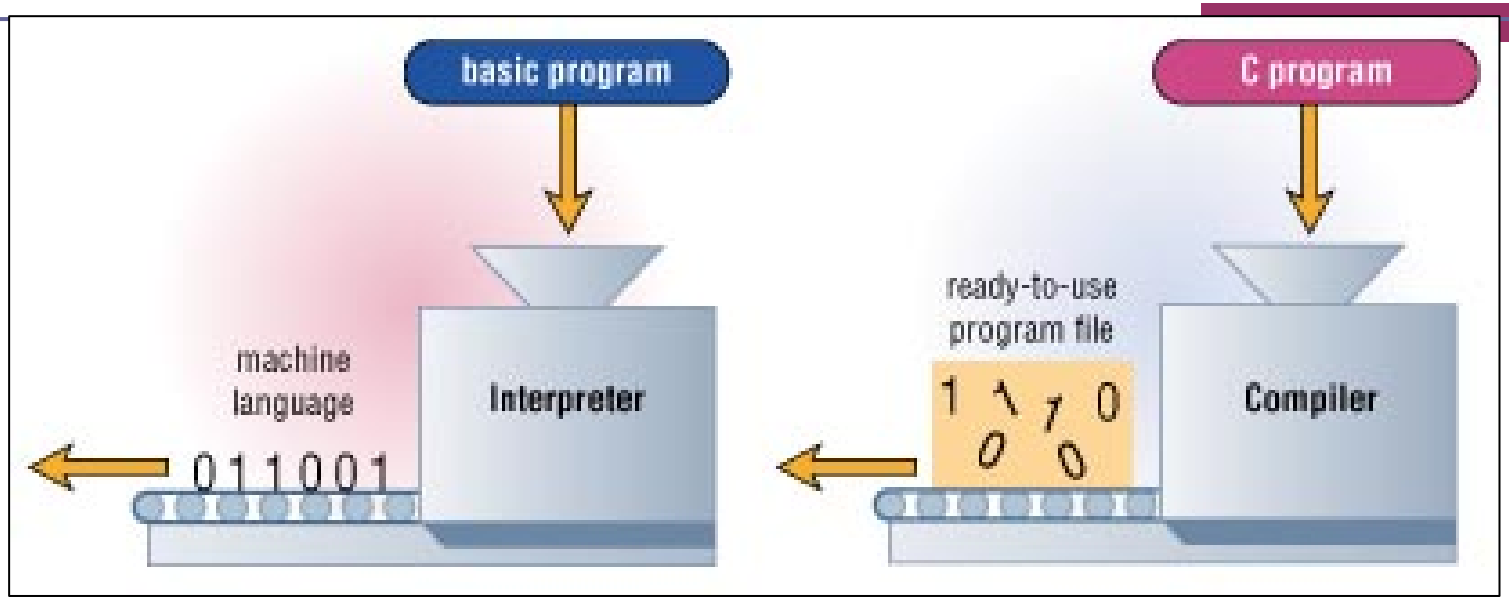
Спецификация

- Осигурява дефиниция, която потребителите на езика използват, за да определят дали поведението на програмата е коректно, разглеждайки кода й.
- Форми на спецификация
  - Явно дефиниране на синтаксиса (най-често чрез формална граматика) и семантиката (посредством естествен или формален език) на езика.
  - Описание на поведението на транслатора на езика на естествен или формален език, от което се определят синтаксиса и семантиката на езика.
  - Моделна имплементация, най-често написана на езика, който се специфицира.

Имплементация

- Осигурява начина на изпълнението на дадена програма върху една или повече конфигурации на хардуер и софтуер.
- Подходи за имплементация
  - Компилация: преобразуване на програма, написана на програмен език в друг компютърен език с цел обработката и от друга програма, например линкер.
  - Интерпретация: изпълнение на инструкции, написани на програмен език; интерпретаторът може да:
    - Изпълнява директно програмен код;
    - Транслира програмен код в ефективен междинен код и незабавно го изпълнява (Perl, MATLAB);
    - Явно изпълнява съхранен прекомпилиран код, създаден от компилатор, който е част от интерпретиращата система (USCD Pascal, Java).

Компилатори и Интерпретатори



- **Компилаторът** е програма, която превежда изходния код в код, готов за изпълнение в последствие
- **Интерпретаторът** превежда по един ред от изходния код и го изпълнява веднага

Езици за програмиране

- Програмните езици са класифицирани по нива или поколения (генерации)
- Езиците от ниско ниво са най-старите
- Петте поколения програмни езици са:
  - Машинни езици
  - Асемблерни езици
  - Процедурни езици
  - Проблемно-ориентирани езици
  - Естествени езици

Първо поколение езици за програмиране

- **Машинен език**:
  - Двоични цифри (0 и 1)
  - Най-ранните езици за програмиране
  - Единствените, които компютрите разбират без транслация (превод)
  - Машинно зависими
    - Всяка фамилия от процесори имат собствен машинен език

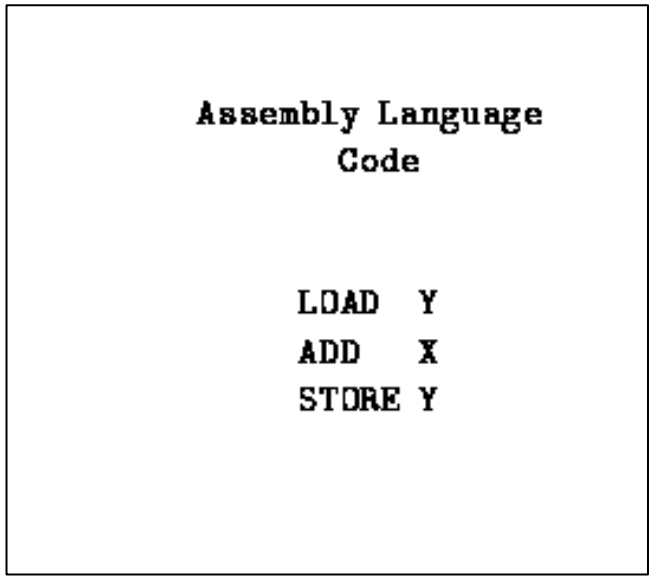
```
Program Fragment:          Y = Y + X

Machine Language Code
(Binary Code)

Opcode          Address
1100 0000       0010 0000 0000 0000
1011 0000       0001 0000 0000 0000
1001 0000       0010 0000 0000 0000


Memory Cell Definitions:

     Addr.      Name       Cell Contents

     1000       X          32
     2000       Y          16
```

Първо поколение езици за програмиране

- Машинни езици за програмиране
  - Не се нуждаят от транслатори;
  - Кодът директно се изпълнява от процесора;
  - Изпълнението е бързо и ефективно;
  - Трудни са за откриване на грешки и програмиране;
  - Приложения:
    - Изходът при “native” компилаторите е машинен език.
    - Виртуални машини: осигуряват мост между машинния език и байт кода.
    - Някои компютърни вируси използват машинен код.
    - През 90-те инжектирането на код се е използвало за разширяване на езици като C и BASIC.

Второ поколение езици за програмиране

- **Асемблерен език**:
  - Близки до машинните езици
  - Език от ниско ниво
  - Използва съкращения за програмните инструкции
    - Съкращенията се наричат мнемонични кодове
  - Програмата се пише в текстов файл и се превежда в машинен език чрез асемблер



Второ поколение езици за програмиране

- Асемблерни езици за програмиране
  - Кодът може да бъде написан или четен от програмист;
  - За да бъде стартирана на компютър, програмата трябва да се преобразува до машинен код (асемблирана);
  - Езикът е специфичен за определена фамилия процесори и среда;
  - Програмистът трябва да е запознат с микропроцесорната архитектура – регистри и инструкции.

Трето поколение езици за програмиране

- Процедурни езици
- Езици за структурно програмиране
- Модулни езици за програмиране

Трето поколение езици за програмиране

- **Процедурни езици**:
  - Езици от високо ниво
  - Създаване на програми на високо ниво на абстракция с цел по-лесно разбиране на програмите от човека
  - По-лесни за четене, писане и поддръжка от машинните и асемблерните езици
  - Използват понятия като именувана променлива, абстрактен тип данни и синтаксис на алгебричен израз.
  - Машинно независими са – програмите са преносими на различни платформи
  - Използват компилатор или интерпретатор за да преведе кода
- Fortran и COBOL са трето поколение езици

Трето поколение езици за програмиране

- Голямата софтуерна криза:
  - Операторите GOTO довеждат до програми, които е трудно да бъдат проследени
  - Криза през 1960-те
    - Програмите не са били готови в срок
    - Надхвърлян е бюджета за разработката им
    - Съдържали са твърде много грешки
    - Клиентите са били неудовлетворени

Трето поколение езици за програмиране

- **Езици за структурно програмиране**:
  - Подмножество на процедурните езици за програмиране, появили се през 60-те години на миналия век
  - Известни са със забраната за използване на оператора GOTO
  - Програмата е изградена от подсекции
  - Предложено е за пръв път от двама математици Corrado Bohm и Guiseppe Jacopini, според които всяка компютърна програма може да се напише посредством три структури:
    - Решение – определен набор от изрази се изпълняват в зависимост от състоянието на програмата
    - Последователност
    - Цикъл – определен израз се изпълнява до достигане на определено състояние на програмата или когато дадена операция е изпълнена за всеки елемент от съответната колекция
  - Най-известната методология е разработена от Dijkstra: програмата се разделя на секции, всяка от които има един вход и един изход
  - Създадени за подобряване на разработката на софтуер
  - Включват Algol и Pascal
  - Структури за контрол
    - IF-THEN-ELSE

Трето поколение езици за програмиране

- **Модулни езици за програмиране**:
  - Създадени поради проблеми в езиците за структурно програмиране
  - Налагат се през 70-те години на миналия век
  - Програмата се разделя на модули със специфична функционалност
  - Модулите се организират йерархично чрез нива на абстракция
  - Данните на всеки модул са скрити за останалите модули
  - Изисква специален вход, за да произведе специален изход

Четвърто поколение езици за програмиране

- Типовете езици от четвърто поколение включват:
  - Генератори на отчети – от описание на данните и отчета се генерира или директно отчет, или програма за генериране на отчет (BuildProfessional, LINC, Metafont, NATURAL, Oracle Reports и др.);
  - Генератори на форми – управляват взаимодействието с потребителя или генерират програма за това (Progress 4GL AppBuilder, OpenROAD, eDeveloper, Omnis Studio, и др.);
  - Програмни среди от четвърто поколение – генерират цялостни системи от спецификации на екрани, отчети и логика на обработка;
  - Управление на данни – осигуряват команди за манипулация на данни, селектиране и документация за изготвяне на статистически анализи и отчети (PL/SQL, SAS, XBASE++, MATLAB и др.).
- Те са непроцедурни
  - Не изискват програмистите да използват процедури, за да получат резултати

Пето поколение езици за програмиране

- Основават са на решаването на проблеми, използвайки ограниченията, дефинирани за програмата, а не алгоритъм, написан от програмиста
- Повечето логически и декларативни езици за програмиране се отнасят към петото поколение езици
- Използват се най-вече в системите с изкуствен интелект (Prolog, OPS5 и Mercury)
- Базирани за на езика Lisp (ICAD)

Обектно-ориентирано програмиране

- **Обектно-ориентираното програмиране (ООП)**:
  - Многократно използване на компоненти
    - Възможност да бъдат създадени програмни модули, които изпълняват специфична задача
  - Елиминира разликата между програми и данни
  - Използва обекти, които съдържат данни и процедури
- **Обектите** са единици информация, които съдържат както данни, така и методи, които боравят и обработват данните
- **Класовете от обекти**:
  - Йерархия или категории от обекти
  - Обектите на върха на категорията - с по-широк обхват отколкото тези в подкласовете
- **Наследяване –** способността на един обект да предава негови характеристики на неговите подкласове

Обектно-ориентирано програмиране

- Първият обектно-ориентиран език е Simula (1967 г.), използван за създаване на симулационни програми, в които данните се предявят посредством обекти.
- Типичен пример за обектно-ориентиран език е Smalltalk (1972-1980 г.)
- Към обектно-ориентираните езици се отнасят:
  - "Чисти" програмни езици – всичко при тях се представя посредством обект, от примитивните типове до прототипите, блоковете и модулите
  - Езици, считани най-вече за обектно-ориентирани, но с процедурни елементи (C++, Java, Python)
  - Езици, които от историческа гледна точка са процедурни, но са обогатени с обектно-ориентирани характеристики (Fortran 2003, Perl)
  - Езици, които притежават повечето от обектно-ориентираните характеристики, но по свой оригинален начин (Oberon-1 и Oberon-2)
  - Езици, поддържащи абстрактен тип данни, но не всички характеристики на обектно-ориентираното програмиране (обектно-базирани езици – Modula-2, Pliant, CLU)

COBOL (Common Business-Oriented Language)

- **COBOL:**
  - Най-ранен (1959) език от високо ниво
  - Най-широко използван бизнес език
- Проектиран за бизнес, финансови и административни системи за компании и правителства
- COBOL 2002 стандарта притежава обектно-ориентирани характеристики – UNICODE, генериране на XML, конвенции за повикване от и на други езици като например C, поддръжка за изпълнение от среди като Microsoft .NET. и Java
- Притежава над 400 ключови думи
- Първоначалната COBOL спецификация поддържа самомодифициращ се код:

  ALTER X TO PROCEED TO Y
- Поддържа “промяна на място” и съкратени условни изрази:

  ADD YEARS TO AGE
  age = age + years
  IF SALARY > 9000 OR SUPERVISOR-SALARY OR = PREV-SALARY
  IF SALARY > 9000
  OR SALARY > SUPERVISOR-SALARY
  OR SALARY = PREV-SALARY
- Позволява дължината на идентификаторите да бъде до 30 символа

Fortran
(Formula Translator)

- **Fortran:**
  - Появява се през 1950
  - Замислен за научни, математически и инженерни приложения
- Разработен от IBM през 50-те години за научни и инженерни приложения.
- Успешна версия е Fortran 77, в която са добавена обработка на символни данни, подобрения за структурно програмиране и входно/изходни операции.
- Fortran 90/95 поддържа работа с масиви, обектно-базирано и модулно-базирано програмиране, рекурсивно програмиране, SELECT . . . CASE конструкция.
- Fortran 2003 поддържа обектно-ориентирано програмиране, процедурни указатели, асинхронен трансфер и др.
- Fortran 2008 поддържа паралелна обработка и двоичен тип данни.
- Съвременните програми на Fortran са напълно преносими.

Ada

- **Ada:**
  - На името на Августа Ада Байрон, известна като “първият програмист” (разработвала програма за проектирана от Чарлс Бабидж машина)
- Структурен, обектно-ориентиран език от високо ниво със статична проверка на типовете, базиран на Pascal
- Езикът е проектиран за вградени системи и системи, работещи в реално време (авиационна радиоелектроника, термоядрени оръжия и космически кораби
- Поддържа строго типизиране, модулност, проверка на грешките в реално време, прихващане на изключения, динамично заделяне на памет
- Синтаксисът е прост и разбираем – блоковете с код се разделят с думи (declare, begin, end), за математическите операции се използват математически символи, условните блокове завършват с end if

Beginner's **A**ll-**P**urpose **S**ymbolic **I**nstruction **C**ode (BASIC)

- Създаден през 1965 г. от Джон Кемени като средство за обучение.
- Принципи, заложени при проектирането на езика:
  - Лесен за използване
  - Език с общо предназначение
  - Възможност за добавяне на допълнителни възможности при запазване на простотата му
  - Интерактивност
  - Яснота на съобщенията за грешки
  - Липса на необходимост от разбиране на компютърния хардуер и операционната система
  - Бързодействие при малки приложения
- Базиран на Fortran II и Algol 60
- Съществуват множество версии на езика:
  - Неструктуриран BASIC: поддържа прости типове данни, цикли и работа с масиви (MSX BASIC и GW-BASIC)
  - Структуриран BASIC: добавят се възможности за структурно и процедурно програмиране като се изключва номерирането на редовете (QuickBASIC и PowerBASIC)
  - BASIC с обектно-ориентирани възможности: добавят се възможности за обектно-ориентирано и управлявано от събития програмиране, повечето вградени функции и процедури са реализирани като методи на обекти (Visual Basic –интегриран е във визуална среда за разработка и StarOffice Basic)

Visual Basic (VB)

- **Visual Basic:**
  - Широко използван в програмни пакети
  - Използва събитийно-управлявано програмиране
  - Позволява на програмиста да създаде приложение с използване на графичен потребителски интерфейс

Pascal

- Императивен и процедурен програмен език, разработен от Никлаус Вирт
- Базиран на езика Algol и носи името на френския математик Блез Паскал
- Създаден е за обучение по структурно програмиране.
- Позволява създаване на структурни типове данни, изграждане на динамични и рекурсивни даннови структури като списъци, дървета и графи, графично програмиране
- Строго типизиране език
- Относително лесен за научаване
- Версии на езика
  - Turbo Pascal – използва се в първите три версии на Borland Delphi;
  - Object Pascal – добавя възможност за обектно-ориентирано програмиране;
  - Super Pascal – добавя нечислени етикети, оператор return и имена на типове;
  - Morfik Pascal – диалект на Object Pascal, използва се за разработка на уеб приложения.

C

- Разработен през 70-те години на миналия век в лабораториите Бел (Bell Labs) за използване под Unix ОС
- Междуплатформен, блоково структуриран, процедурен и императивен програмен език
- Комбинира език от високо ниво с асемблерен език
- Характеристики:
  - Не се допуска влагане на функции (позволено е в C99);
  - Осигурява достъп до паметта на ниско ниво (статично, автоматично и динамично заделяне на памет)
  - Често се използва за системно програмиране
  - Поддържа използването на указатели (достъп до масиви от символи, свързани списъци, дървета, динамично заделяне на памет, функции)
  - Заменяемост на масиви и указатели
  - Използват се библиотеки като средство за разширение на езика
  - Труден за научаване и разбиране на кода

C++

- Разработен през 1979 г. в лабораториите Бел от Бярне Строуструп като подобрение на езика C
- Проектиран е с цел поддръжка на различни програмни стилове – процедурно програмиране, абстракция на данните, обектно-ориентирано програмиране
- Характеристики:
  - Използване на класове (поддържа обектно-ориентирано програмиране)
  - Виртуални функции
  - Предефиниране на оператори (предоставя повече от 30 оператора, като почти всички могат да се предефинират)
  - Множествено наследяване
  - Използване на шаблони (поддържат се шаблони както на функции, така и на класове)
  - Прихващане на изключения

Smalltalk

- “Чист” обектно-ориентиран език, създаден през 70-те години на миналия век от Ксерокс (примитивните типове също са обекти, класовете също са обекти, т.е. инстанции на метакласове)
- Обектите, създадени с езика могат:
  - Да реферират към друг обект
  - Да получават съобщения от себе си и други обекти
  - Да изпращат съобщения на себе си и други обекти
- Притежава само 6 резервирани думи (true, false, nil, self, super и thisContext)
- Smalltalk програмите се компилират до байт код, който се интерпретира от виртуална машина или се преобразува в машинен код
- Повечето Smalltalk системи не разделят данните на приложението от кода, като вътрешното им състояние се запазва в image файл, който след това може да се зареди от виртуалната машина и да върне системата в предишното състояние

Java

- Езикът е разработен от Sun Microsystems през 1995 г.
- Архитектурата на Java притежава 4 компонента:
  - Програмен език
  - Клас файл (байт код)
  - Java API (Java Runtime System)
  - Виртуална машина
- Предимства:
  - Преносимост
  - Обектно-ориентиран
  - Платформено-независим
  - Java е създаден да може да бъде изпълняван на всяка компютърна платформа
- **Java Virtual Machine** позволява между-платформено използване
- **Java аплетите** са малки програми, които могат да бъдат изтеглени
- Слабости:
  - Рискове при изтегляне на аплети
  - Скоростта на изпълнение на програмите

Уеб-базирани езици

- **Хипертекстови езици:**
  - **Hypertext markup language** (**HTML**) множество от атрибути на текст и обекти в Уеб страница
  - **Extensible markup language** (**XML**) се използва за споделяне на данни и обекти в Уеб
- **Скриптови езици:**
  - **VBScript и JavaScript** се използва за малки програми (скриптове), които са “вградени” в Уеб страниците

Жизнен цикъл при разработката на програми (ЖЦ)

- ЖЦ в въведен през 1970-те във връзка с проблемите при създаване на програми
- Предлага организиран план за разделяне на задачата за разработка на програма на управляеми части
- Шест фази на ЖЦ:
  1. Дефиниране на проблема
  2. Дизайн на програма
  3. Програмиране
  4. Тестване и откриване на грешки
  5. Формализиране на решението
  6. Внедряване и поддръжка на програмата

Фаза 1: Дефиниране на проблема

- Анализират се клиентските изисквания и се изготвя спецификация.
- Основни стъпки:
  - Дефиниране на целта на програмния продукт и неговите потребители;
  - Специфициране на изходните данни
  - Специфициране на входните данни
  - Дефиниране на обработката на данните
  - Оценка на възможностите за имплементация
  - Документиране.
- Трудности:
  - Клиентът изисква неподходящ продукт
  - Клиентът няма софтуерни или хардуерни познания
  - Спецификацията може да бъде двусмислена и непълна
- Спецификацията дефинира:
  - Входните данни
  - Обработката им
  - Изхода
  - Графичния интерфейс

Фаза 2: Проектиране на програмата

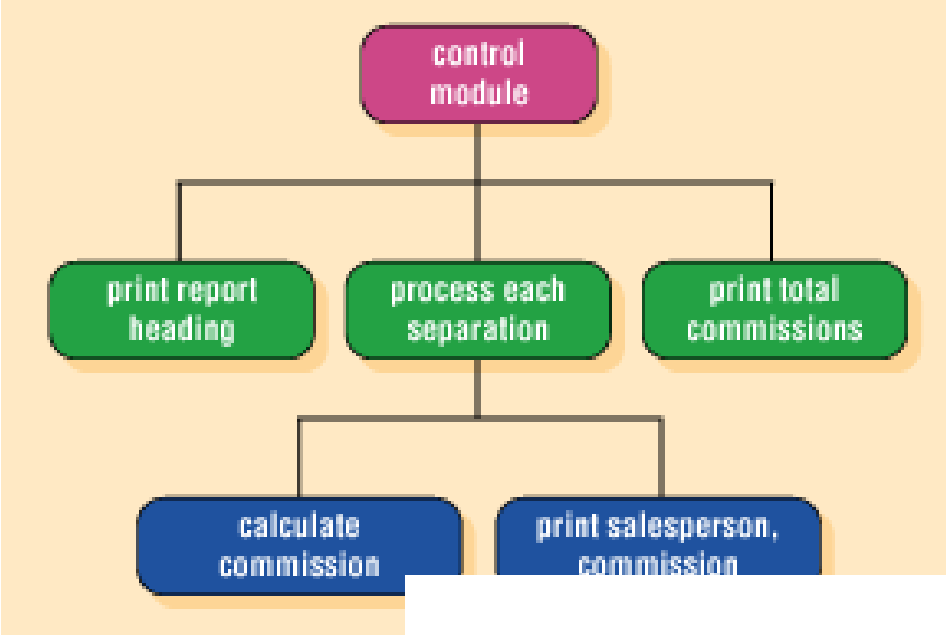
- Програмистите създават дизайна на програмата
  - **При проектирането отгоре надолу** фокусът пада върху главната цел (основна/главна процедура), тогава програмата се разбива на управляеми части (подпрограми/процедури/модули)
  - Контролни структури се използват, за да се види всяка подпрограма дали ще изпълни нейната работа
- Разработката на алгоритъм е описание стъпка по стъпка как ще се достигне до решение
- Средства за проектиране:
  - Структурни диаграми – структурата на програмата
  - Блок-схема – логиката на програмата
  - Псевдокод – алтернатива на блок-схемата

Структурен дизайн

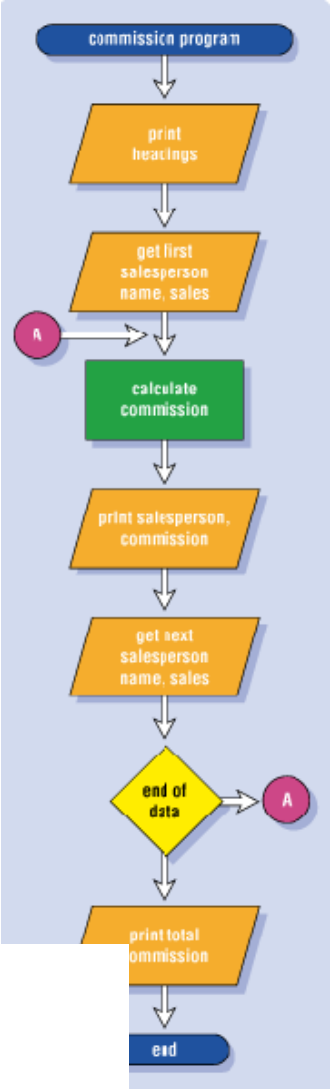
- **Контролните структури** са логически конструкции, които специфицират как ще бъдат изпълнявани инструкциите в програмата
- Три типа конструкции:
  - **Последователна** – Инструкциите се изпълняват в реда, в които се появяват
  - **За избор** – Изпълнението на различни части от програмата зависи от това дали дадено условие е изпълнено; IF…THEN…ELSE
  - **За повторение** – Програмата изпълнява многократно едни и същи инструкции; DO-WHILE и DO-UN

Структурна диаграма и блок-схема

Структурна-диаграма



Блок-схема



Фаза 3: Програмиране

- Основни стъпки:
  - Имплементация на програмните модули с подходящи софтуерни средства според спецификацията;
  - Интегриране на модулите съгласно архитектурния дизайн.
- Трудности:
  - Грешки при взаимодействието на модулите;
  - Редът на интегриране влияе върху качеството на продукта.

Фаза 4: Тестване, откриване и отстраняване на грешки

- Програмният продукт се тества с цел откриване и отстраняване на синтактични и логически грешки:
  - Проверка за съответствие на имплементацията с изискванията и проектираното решение;
  - Тестване на индивидуалната функционалност на всеки модул;
  - Тестване на интерфейса между модулите;
  - Тестване на целия продукт за правилно функциониране.

Фаза 5: Формализиране на решението

- Документацията се създава за бъдещо използване
- Подготвя се документация с имената на променливите и дефинициите, описание на необходимите файлове и изхода на програмата
- Разработва се ръководство за потребителя, за да се обясни как работи програмата
- Документиране:
  - Документация за потребителя
  - Документация за оператора
  - Документация за програмиста

Фаза 6: Внедряване и поддръжка

- Продуктът се доставя на потребителя.
- При реалната работа на продукта е възможно да възникнат грешки
- В случай на възникнали грешки и необходимост от нова функционалност се извършва модификация на продукта
- При всяка модификация продуктът отново се тества за грешки

Обобщение

- Езиците за програмиране са “изкуствени” езици с речник и набор от правила
- Машинните езици са най-ниското ниво езици за програмиране
- Асемблерните езици използват символи за програмните инструкции
- Третото поколение езици (от високо ниво) изискват програмистите да определят процедурите, които ще се следват
- Обектно-ориентираните езици комбинират процедури и данни

Обобщение (2)

- Шестте фази на ЖЦ са:
  - Дефиниране на проблема
  - Дизайн на програма
  - Програмиране
  - Тестване и откриване на грешки
  - Формализиране на решението
  - Внедряване и поддръжка на програмата
- Програмирането отгоре надолу прави по-лесно откриването на грешки и поддръжката
- Откриването и отстраняването на грешки включва синтактични и логически грешки

Въпроси?

За контакти:

elis@fmi.uni-sofia.bg