

Лекция 2.

ЕП и тяхната реализация

*Лесно ли е
да създадем ЕП?*

Процесът на трансляция

За да бъде *използван*
новосъздаден ЕП от високо ниво



той трябва да се *реализира* чрез
някой от преди това същест-
вуващите и реализирани ЕП



т.е в крайна сметка той трябва
да бъде *реализиран на*
машинния език, който е
материализиран чрез хардуера
на машината, а не чрез софтуер

За да бъде *изпълняван* един ЕП
от високо ниво



той трябва да може да се
превежда (translate) на машинен
език



т.е. написаните от програмиста
програми в текстов вид (*source*
code) трябва да се преведат във
вид, който може да се изпълни
от компютъра (*byte, executable code*)

- **Транслация** (*translation*) – процесът на превеждане на текстовия код на програмата в изпълним (двоичен) код
- **Транслатор** (*translator*) е инструментът, който превежда текстовия код на програмата в изпълним код

Компилатори и интерпретатори

- Видове транслатори (ЕП обикновено се реализира чрез):
 - **Компилятор** (*compiler*) – превежда цялата програма от текстовия код на езика (*source language*) в програма на друг ЕП (обектен/целеви език, *object/target language*)
 - **Интерпретатор** (*interpreter*) – чете текста на програмата от езика и я изпълнява директно твърдение след твърдение
- Сравнение на компилаторите и интерпретаторите:

Показатели	Компилатори	Интерпретатори
Реализация	по-трудно	по-лесно
Време за компилиране на програмата	повече	по-малко
Време за изпълнение на програмата	по-малко	повече
Многократно изпълнение на програмата	по-ефективни при многократно изпълнение	по-бързи при еднократно изпълнение
Вид на разработвания софтуер	за комерсиален софтуер (C++, Java, Delphi)	експериментални разработки, напълно нови ЕП (Lisp, Prolog)

Програмиране

Дейности, които включва процесът на програмиране:

- ✓ *написване* на програмата в текстов вид (**кодиране**)
- ✓ *превеждане* на текстовия код на програмата в изпълним код (**транслация**)
- ✓ *изпълнение* на програмата със съответните входни данни (**изпълнение**)
- ✓ *проверка за грешки* в програмата и тяхното отстраняване (**тестване и настройка**)
 - *разглеждане на резултатите от изпълнението* на програмата, за да се разбере какво е нейното действие
 - *изпълнение* на програмата с различни входни данни, за да се открият грешките (**тестване**)
 - *отстраняване* на грешките (**настройка**)
- ✓ *модифициране* на програмата, за да се адаптира към промените в света и нуждите на потребителите (**поддръжка**)
- ✓ *разширяване и допълване* на функционалните възможности на програмата, за да бъде тя по-полезна и удобна за потребителите (**развитие**)

Програмни грешки

При създаването на една програма могат да се допуснат три вида грешки:

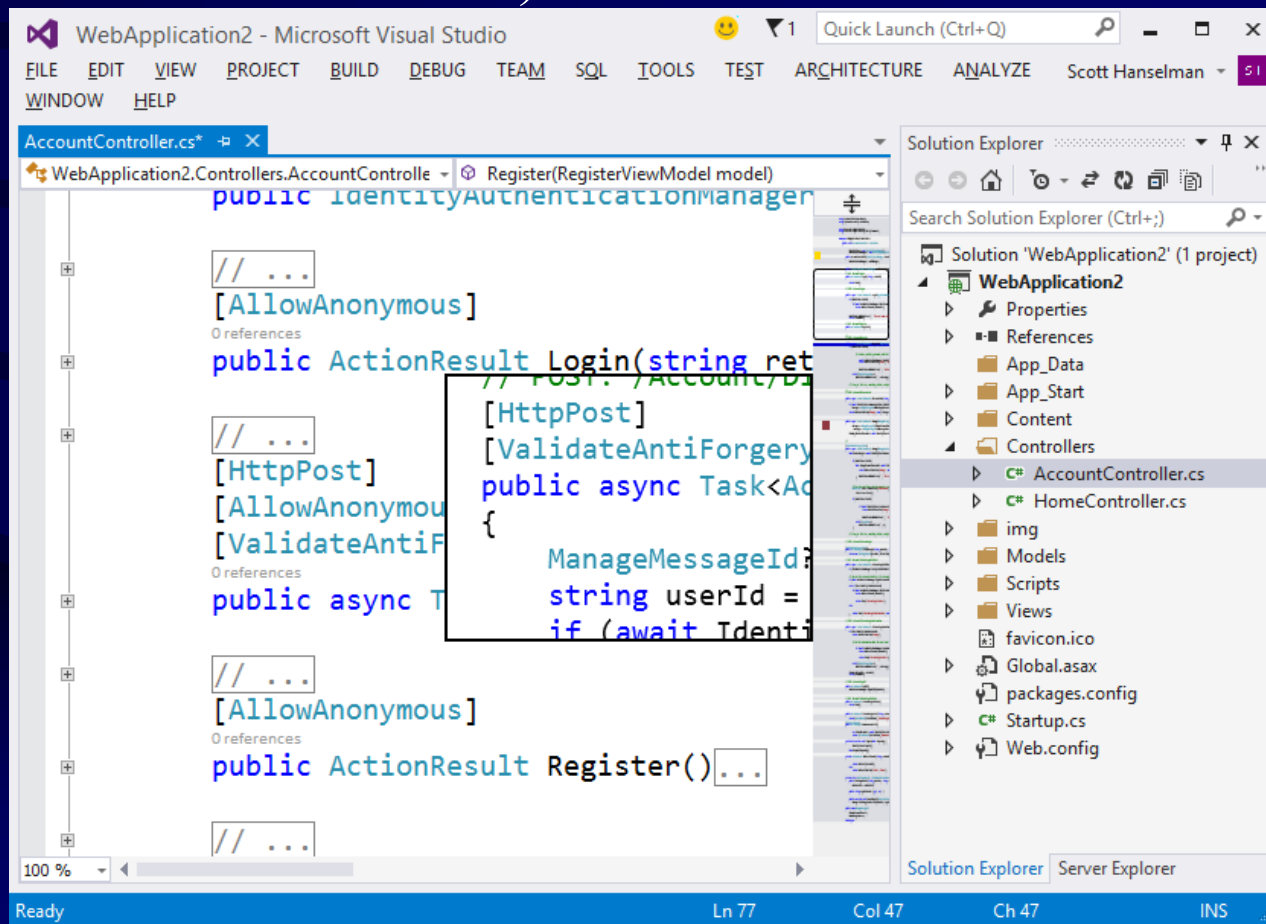
- *Синтактични грешки* – твърдения в програмата, които са грешни от синтактична гледна точка
- *Грешки по време на изпълнението на програмата* – програмни конструкции, които са синтактично правилни, но са безсмислени
- *Логически грешки* – част от програмата, която е синтактично правилна, има смисъл, но е грешна от гледна точка на действията, които програмата трябва да изпълнява

Среди за програмиране

- Среда за програмиране:
Съвкупност от програмни средства и инструменти за разработка на програми на някакъв ЕП
- Предназначение:
 - цялостна поддръжка на програмните дейности
 - улеснява всички етапи от разработването на програмите
 - осигурява създаването на високо-качествени, ефективни и надеждни програми
- Могат да включват:
 - Редактори
 - Дебъгери
 - Компилатори
 - Свързващи редактори и др.

Среди за програмиране: *Програмен редактор (Program editor)*

- Предназначение — написване (кодиране) на текста на програмата и създаване на файловете, от които се състои тя (напр. `Rational.s.cpp` в C++)
- Характеристики (контекстно-зависими):
- улесняват кодирането, като предлагат избор между допустимите възможности при въвеждане на следващата конструкция според контекста, формиран до момента
- подпомагат структурирането на текста и подобряват четливостта на програмата за човека
- предупреждават за синтактични грешки, чрез форматиране на кода на базата на синтаксиса на използвания ЕП



Среди за програмиране: *Транслатори и свързващи редактори*

Транслатори

- Предназначение — превежда програмата, написана с помощта на редактора на език, разбираем за машината и я записва във файл (напр. `Rationals.obj` в средата на C++)
- Характеристики:
 - подпомага откриването и премахването на синтактични грешки
 - оптимизиращи транслатори
 - файлът, създаден от транслатора се нарича *обектен код* — предимно машинен код. Той притежава повечето от изграждащите блокове на програмния код освен системния код, който се намира във вградените библиотеки

Свързващи редактори (Linker)

- Свързва обектния код със съответния библиотечен код, т.е. свързването на изграждащите блокове в едно
- Създава изпълнимият файл (напр. `Rationals.exe` в C++)

Среди за програмиране: Зареждаща програма и дебъгери

Зареждаща програма (Loader)

- Предназначение — зарежда изпълнима програма, създадена от компилатора в паметта на компютъра с цел тя да бъде изпълнена (при нейното стартиране)
- Характеристики:
 - тя е част от операционната система
 - след зареждането на програмата операционната система казва на процесора да започне изпълнението на последователността от инструкции от мястото в паметта, където сега се намира програмата

Дебъгери (Debugger)

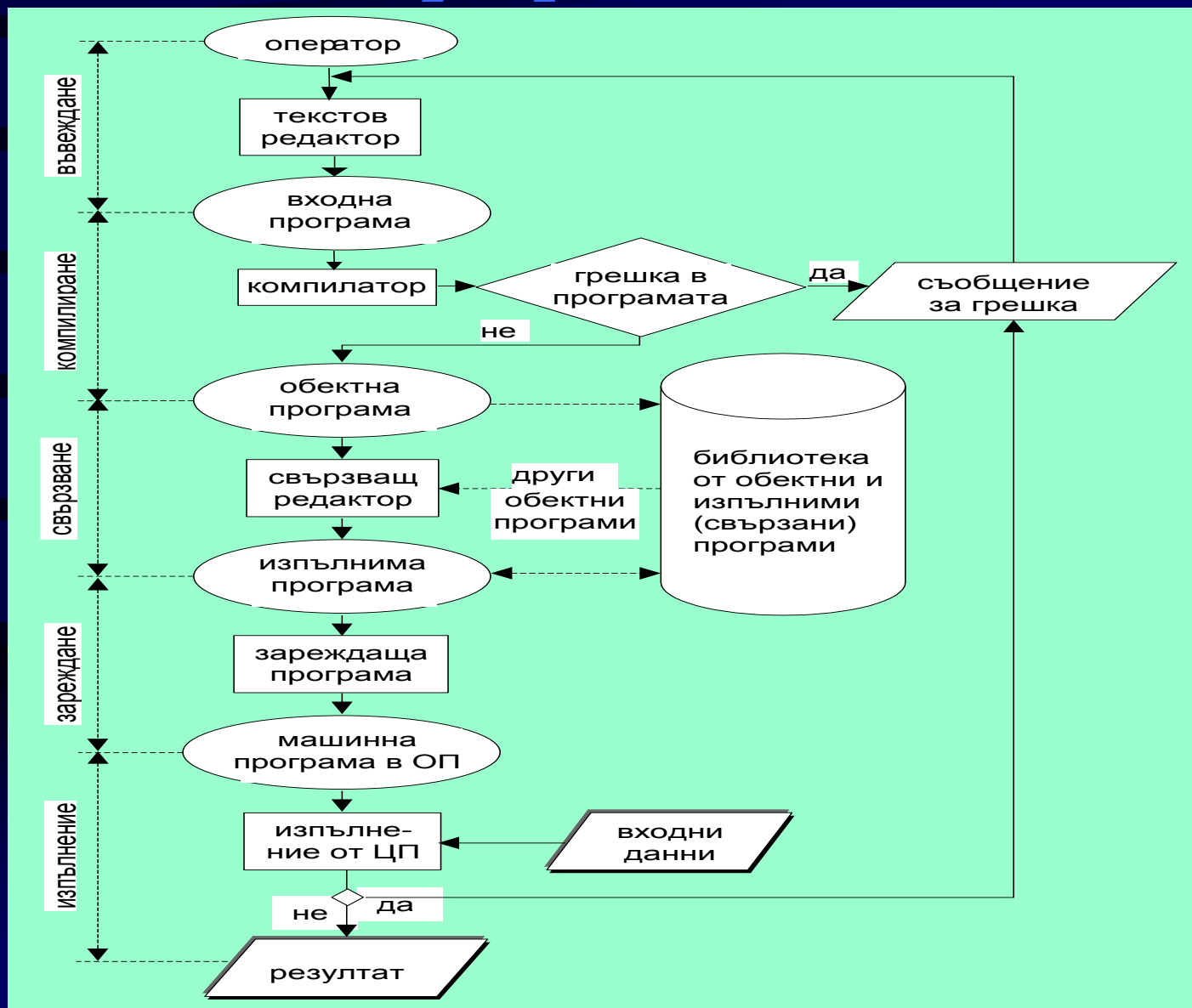
- Предназначение — подпомага откриването и премахването на грешките по време на изпълнението на програмата и логическите грешки

Самият процес се нарича *дебъгване* (*bug** — грешка)

Дебъгърът позволява т.нар. *трасиране* (*Tracing*) на програмата. Трасирането включва следене на състоянието на паметта (напр. стойност на променлива) при изпълнението на програмните инструкции, така че цикълът от операции да може да се проследи и да се открият грешките

* Терминът *bug* е въведен от математичката Грейс Хопър, когато намира заклещена буболечка в изчислителната машина

Среди за програмиране: *Жизнен цикъл на програмата*



Принципи за разработването на ЕП:

Ефективност

- ✓ по отношение на скорост
- ✓ по отношение на обем необходима памет
- ✓ скорост и леснота при писането на програма
- ✓ изразителност
- ✓ ефективност на транслиране
- ✓ леснота на реализиране на езика (написване на транслатор)
- ✓ ефективност при изпълнение на програмите
- ✓ надеждност
- ✓ лесна поддръжка
- ✓ възможност за повторната употреба
- ✓ преносимост

Други принципи за разработването на ЕП

- ✓ **общност** – избягването на частните случаи
- ✓ **ортогоналност** – предсказуемост при обединяването на конструкции
- ✓ **еднородност** – подобните неща изглеждат по подобен начин
- ✓ **простота** – всичко е, колкото е възможно по-просто
- ✓ **точност** – определеност, съществуването на прецизни езикови дефиниции (напр. наръчник за езика, наличие на стандарти – ANSI, ISO)
- ✓ **машинно-независимост** – наличие на предварително дефинирани абстрактни типове данни и решаване на проблема с точността
- ✓ **сигурност** – откриването на грешки (свързано е с надеждността и точността)
- ✓ **разширяемост** – потребителят да може да добавя нови възможности (напр. натоварване и полиморфизъм)

C++ Програмиране

C++

- съчетава принципите на две програмни парадигми – процедурната и обектно-ориентираната
- създаден от Строуструп (Dr Bjarne Stroustrup)
 - през 80-те години
 - в AT&T Bell Labs
- последна версия от 2011 неформално наречена C++11

Основни лексикални елементи на програмите

Лексикалните елементи (тоукън, *token* – знак, нещо значещо, символично) са основни градивни елементи на програмите, които са значещи за компилатора

- **име, идентификатор** (*identifier*) – последователност от символи, което служи за идентификация обикновено на програми (алгоритми), подпрограми (подалгоритми), елементи на данните и др.
- **ключова дума** (*keyword*) – предварително дефинирани имена, които имат специално предназначение при писане на програмите и не могат да бъдат използвани за други цели
- **константа** (*constant*)/**литерал** – стойност, която не се променя по време на изпълнение на програмата, а само се използва в процеса на изчисление
- **операция** (*operator*) – задава намирането на някаква стойност по дадени един, два или три операнда (съответно едно-, дву-, три-аргументна операция)
- **коментар** (*comment*) – текст, който се игнорира от компилатора, но е полезен за програмиста
- **препинателен знак** (*punctuator*) – ! % ^ & * () - + = { } | ~ [] \ ; ' : " < > ? , . / #



Лексикалните елементи на програмите се отделят с **разделители**, като празен интервал, край на ред, коментар и др.


Основни елементи на ЕП от гледна точка на синтаксиса и семантиката

Най-общо ЕП включва средства за описание на два структурни елемента:

- **Деклариране и дефиниране на данните** — служи за определяне на компютърното представяне на входните и изходните данни и междинните резултати, на техния тип и структура, както и свързаните с тях изисквания за памет
- **Деклариране и дефиниране на обработката** — определя правилата за обработка на данните, как евентуално те се групират, както и последователността на тяхното прилагане чрез т.н. **оператори** (команди, *statements*) на ЕП

 В програмите написани на някои ЕП тези части са строго разграничени (Паскал), а в други могат да се преплитат в общата структура (СИ)

 **Декларацията** „казва“ на компилатора, че съответният програмен елемент съществува

 **Дефиницията** определя точно кой код или данни декларираното име идентифицира

 Името трябва да бъде декларирано преди да бъде използвано

 Всеки елемент данни в програмата е или **константа** или **променлива** (**величини**)

Най-проста структура на програма на C++ – пример

- **Пример** (Програма, която извежда на екрана “Програмирането на C++ е лесно”):

```
/*  
    Първа програма на C++  
    за курса по Програмиране  
*/  
#include <iostream.h>  
  
main() //от тук започва изпълнението на програмата  
{  
    cout << "Програмирането на C++ е лесно.\n";  
}
```

- **Изпълнение на програмата:**
Програмирането на C++ е лесно.

Най-проста структура на програма на C++ – резюме върху примера

- изпълнението на програмата започва от `main()`
- ключовите думи се пишат с малки букви
- всеки оператор завършва с `“ ; ”`
- текстовите константи (низове) се заграждат в кавички (`“ . . . ”`)
- в C++ малките и големи букви се различават
- `\n` означава позициониране на курсора в началото на следващия нов ред
- `cout <<` може да се използва за извеждане на текст на екрана
- `{ }` определят началото и края на програмата (блок на програмата)
- `//` указва началото на коментар, който завършва в края на същия ред
- коментари с дължина повече от един ред се ограждат от `/* . . . */`. Те не могат да се влагат

Най-проста структура на програма на C++

```
/* ЗАГЛАВНА СЕКЦИЯ                                */
/* Съдържа име, автор, номер версия                */

/* СЕКЦИЯ ЗА ВКЛЮЧВАНЕ НА ФАЙЛОВЕ                  */
/* Съдържа #include директиви                      */

/* СЕКЦИЯ ЗА ДЕКЛ. И ДЕФ. НА КОНСТАНТИ И ТИПОВЕ   */
/* Съдържа типове и директивата #define            */

/* СЕКЦИЯ ЗА ГЛОБАЛНИ ПРОМЕНЛИВИ                   */
/* Всички глобални променливи се декларират тук    */

/* СЕКЦИЯ ЗА ДЕКЛАРИРАНЕ И ДЕФИНИРАНЕ НА ФУНКЦИИ */
/* Функции дефинирани от потребителя              */

/* main() СЕКЦИЯ (ГЛАВНА ФУНКЦИЯ)                  */

int main()
{
    //...
}
```

 Вж. Лекция 14

Абстракции в езиците за програмиране

- **Необходимост:**

- Компютрите нямаше да са толкова популярни ако всеки път, когато искаме да умножим две числа трябва да слизаме до нивото на машинното представяне
- Програмистите имат нужда да описват програмите по начин, който има смисъл за самите тях

- **Същност:**

Абстракциите в ЕП се използват, за да пресъздадат човешкото разбиране за данните и тяхната обработка, отделяйки го от компютърното разбиране (скривайки компютърното представяне)

Видове абстракции в ЕП

- *Абстракции на данните* – служат за изразяване свойствата на данните и тяхната функционалност:
 - *базови абстракции на данни* – представяне на данните на базата на концепцията за тип на данните. Те се наричат **прости** или примитивни **типове данни**
 - *структурирани абстракции на данни* – абстракции на съвкупности или свързани стойности от данни (напр. C++ структура или запис на Pascal). Те се наричат **структурни типове данни**
- *Абстракции за контрол на управлението* (**управляващи конструкции**) – предназначени са да управляват последователността на изпълнението:
 - *базови абстракции за контрол* – оператори, които комбинират няколко машинни инструкции в по-разбираемо абстрактно твърдение (напр. присвояване =, goto). Те се наричат **прости оператори**
 - *структурирани абстракции за контрол* – оператори, които разделят програмата на групи от инструкции и управляват тяхното изпълнение (напр. case, switch, if-then-else, do-while, subprogram, subroutine, function). Т. нар. **структурни оператори**