ЛЕКЦИЯ 13
СТРУКТУРНИ ОПЕРАТОРИ

- Видове участъци в един алгоритъм
- Същност на структурните оператори
- Съставен оператор
- Оператори за разклонение
- Оператори за повторение

ВИДОВЕ АЛГОРИТМИ

За да определим какви структурни оператори трябва да има в един ЕПВР преди всичко трябва да решим въпроса какви участъци може да съдържа един алгоритъм.

Лесно се вижда, че те са три вида:

- (Линейна) последователност;
- Разклоняване по поне 2 пътя;
- Повторение на даден участък.

 Последна редакция 29 Януари 2012 г.

СЪЩНОСТ НА СТРУКТУРНИТЕ ОПЕРАТОРИ

Съдържат като част от своя запис **други оператори** и **определят реда** за тяхното изпълнение.

Това са операторите, които визират теоретиците **на структурното програмиране**.

Могат да се влагат един в друг (**като структурите от данни**).

ПРОГ_13 3/45

СЪСТАВЕН ОПЕРАТОР

Последователното изпълнение е главна характеристика на програмата като единно цяло.

От това гледище **специален оператор за последователно изпълнение** като, че ли не е необходим.

В много ЕПВР съществува ***синтактично ограничение*** като **част от структурен** оператор да се задава **само един единствен друг оператор**.

ПРОГ_13 4/45

СЪСТАВЕН ОПЕРАТОР (продължение)

За преодоляване на посоченото ограничение в езици, в които има такова, се предоставя **специален оператор за последователно изпълнение** на съставящите го оператори.

Този оператор се нарича ***съставен оператор*** – като **единен оператор**, той спазва ограничението и в същото време осигурява последователно **изпълнение на операторите**, които са **част от него**.

ПРИМЕР: СЪСТАВЕН ОПЕРАТОР

Паскал: `BEGIN` <оператор1> `;` <оператор2> [`;` ...] `END`

Си: {<оператор1> <оператор2> [...]}

ВБ: **няма синтактично ограничение** като част от структурен оператор да се задава един единствен оператор, поради което **не е необходим** съставен **оператор**.

ОПЕРАТОРИ ЗА РАЗКЛОНЕНИЕ

Предназначени са да осигурят приспособимост на програмата към възникващите по време на нейното изпълнение обстоятелства.

При изпълнението им се **изчислява израз** (наречен **условие**) и **в зависимост от получената стойност изпълнението** продължава **по един от зададените алтернативни пътища**.

ПРОГ_13 7/45

ОПЕРАТОРИ ЗА РАЗКЛОНЕНИЕ (прод.)

В зависимост от типа на израза – условие, операторите за разклонение, които често се наричат **оператори за вземане на решение**, са **два вида**:

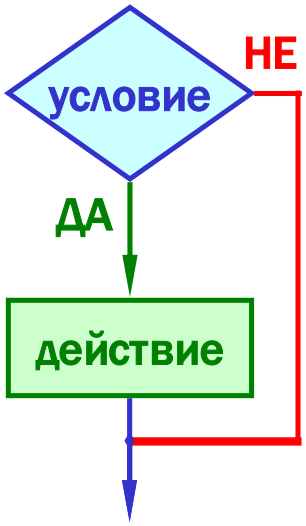
1. ***условен оператор*** с условие от **логически тип** и **две** алтернативи;
2. ***оператор за избор*** с условие от **целочислен** (или друг дискретен) **тип** и **произволен брой** алтернативи.

ПРОГ_13 8/45

УСЛОВЕН ОПЕРАТОР (КРАТКА ФОРМА)

Най-простият вид **разклонение** е **да изпълним или да заобиколим** (**пропуснем** изпълнението на) дадено **действие**.

Напълно ***достатъчни*** са **един израз и едно действие**.



ПРИМЕР: КРАТКА ФОРМА НА УСЛОВЕН ОПЕРАТОР

Паскал: `IF` <условие> `THEN` <оператор>

Си: `if(`<условие>`)` <оператор>

ВБ: **редови** (в един ред) :

`If` <условие> `Then` <оп1> [`:` <оп2> ...]↵

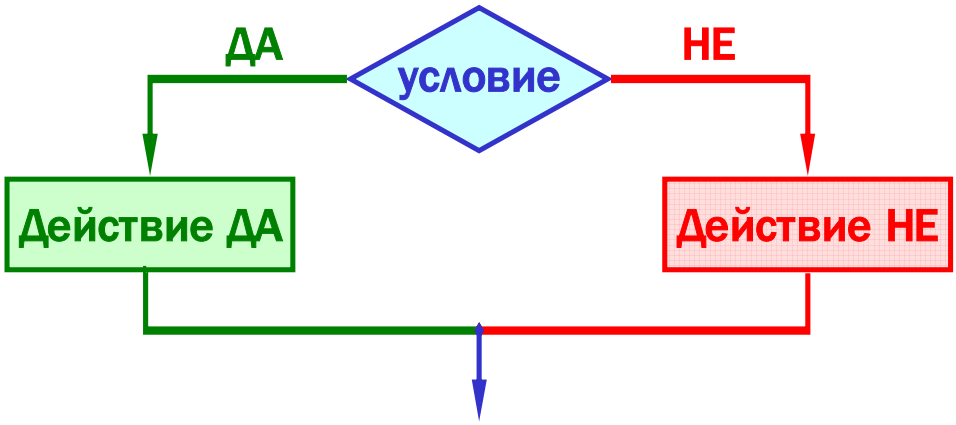
`If` <условие> `Then` ↵ **блоков**

```
   <оператор1>
   ...
End If ↵
```

(**if** – **ако**, **then** – **то**)

УСЛОВЕН ОПЕРАТОР (ПЪЛНА ФОРМА)

По-добро решение е да изпълним само едно от две зададени алтернативни действия.



ПРОГ_13 11/45

ПРИМЕР: ПЪЛНА ФОРМА НА УСЛОВЕН ОПЕРАТОР

Паскал: `IF` <усл.> `THEN` <опер.И> `ELSE` <опер.Л>

Си: `if(`<усл.>`)` <опер. И> `else` <опер. Л>

ВБ: редови (в един ред):

```
If <усл> Then <И1> [:<И2> ...] Else <Л1> ... ↵
If <условие1> Then ↵   и блоков
 <оператори1>
[ElseIf <условие2> Then ↵
 <оператори2>
 . . .  ]
Else ↵
 <оператори иначе>
End If ↵
```

ПРОГ_13 12/45

ДВУСМИСЛИЕТО НА IF

Наличието на две форми на условния оператор **и** правото за **влагане на структурни оператори** води до **двусмисления запис**:

IF <условие$_1$> THEN IF <условие$_2$> THEN <оператор$_1$> ELSE <оператор$_2$>

❶ **записан е** условен **оператор в пълна форма** **IF** условие$_1$ **THEN** оператор_И **ELSE** оператор$_2$, чийто **оператор_И е** условният оператор **в кратка форма** **IF** условие$_2$ **THEN** оператор$_1$;

ДВУСМИСЛИЕТО НА IF (продължение)

Двусмислен запис:

IF <условие$_1$> THEN IF <условие$_2$> THEN <оператор$_1$> ELSE <оператор$_2$>

❷ **записан е** условен **оператор в кратка форма** **IF** условие$_1$ **THEN** оператор, чийто **вложен** оператор **е** условният **оператор в пълна форма** **IF** условие$_2$ **THEN** оператор$_1$ **ELSE** оператор$_2$.

КОРЕКТЕН ЗАПИС

Коректният и недвусмислен запис в двата случая е:

❶ IF <условие$_1$> THEN BEGIN; IF <условие$_2$> THEN <оператор$_1$> END; ELSE <оператор$_2$>

❷ IF <условие1> THEN BEGIN; IF <условие$_2$> THEN <оператор$_1$> ELSE <оператор$_2$> END;

РЕШЕНИЯТА

Създателите на Алгол-60 са били толкова шокирани от факта на двусмислие, че набързо са обявили цитираната конструкция за незаконна, забранявайки след then да се записва условен оператор.

Създателите на по-новите езици са по-либерални като въвеждат правилото, че всяко else се свързва с най-близкия от ляво then (Си, Паскал) – тълкуване ❷.

ЗАБЕЛЕЖКИ

1. Операторът за **безусловен преход go to** и **кратката форма на** оператор **if** са напълно **достатъчни за** запис на **произволен алгоритъм**.
2. **Машинните езици** съдържат преките еквиваленти само на тези два оператора.
3. **Логическият тип е беден** на стойности и може да осигури **само две алтернативи**.
4. **Влагането на** оператори **if** осигурява **произволен брой алтернативи**.

ОПЕРАТОР ЗА ИЗБОР

Влагането на оператори if влошава четенето и разбирането на програмата и **изисква изчисляване на множество изрази**, което не е рационално, а може и да не е желателно.

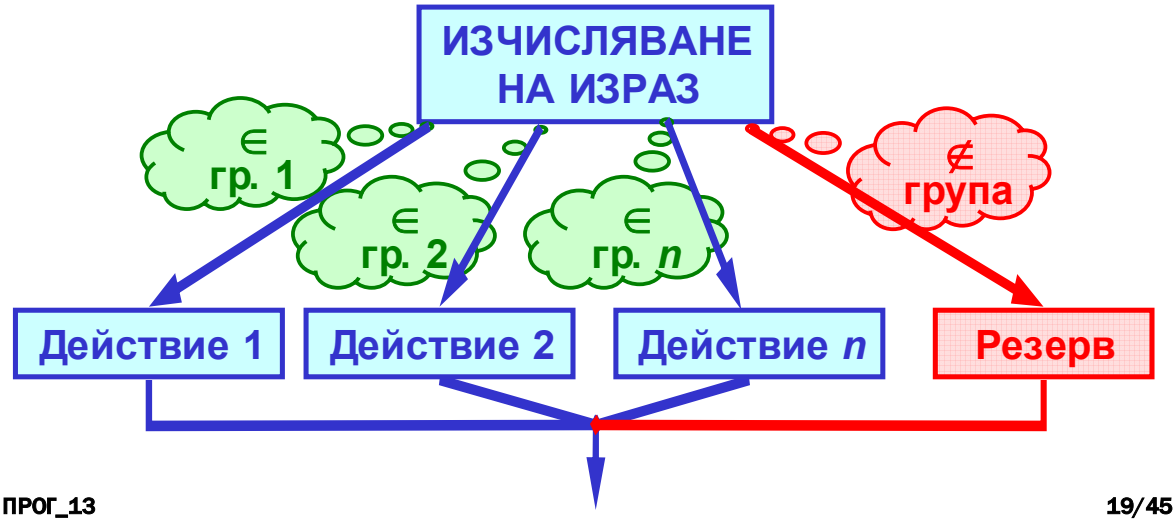
За осигуряване на **множество алтернативи** се търсят решения още в първите езици:

- Изчисляемо go to във **Фортран**;
- Именоващ израз в **Алгол–60** (≈ масив етикети).

Съвременните езици осигуряват специален **оператор за избор**, чрез който **еднократното изчисляване на** целочислен (дискретен) **израз** дава възможност за **продължаване** на изпълнението **по повече от две алтернативи**.

ОПЕРАТОР ЗА ИЗБОР (продължение)

Освен от целочислен тип изразът за избиране на алтернатива може да бъде **от** всеки **друг дискретен тип** (знак, изброим и т. п.).



ПРИМЕР: ОПЕРАТОР ЗА ИЗБОР В ПАСКАЛ

```
CASE <Израз> OF
  <Списък константи 1>  : <Оператор 1>;
  <Списък константи 2>  : <Оператор 2>;
  [  .  .  .
  <Списък константи n>  : <Оператор n>;]
  [ELSE  <Оператор Else>]  само в диалекти
END;
```

<Списък константи i> са разделени със запетаи константи и интервали (н..к) от типа на <Израз>.

ПРИМЕР: ОПЕРАТОР ЗА ИЗБОР В СИ

```
switch (<Израз>) {
 <Константа 1> : <Оператори 1>
 <Константа 2> : <Оператори 2>
 [ . . .
 <Константа n> : <Оператори n>]
 [default: <допълнителни оператори>]
}
```

<Константа i> играе ролята на етикет, т. е. след <Оператори i> се изпълняват <Оператори i +1> ...

За напускане на `switch` се използва `break;`.

ПРИМЕР: ОПЕРАТОР ЗА ИЗБОР ВЪВ ВБ

```
Select Case <Израз> ↵
  Case <Диапазон 1> ↵
    <Оператори 1>
  Case <Диапазон 2> ↵
    <Оператори 2>
[ ... Case <Диапазон n> ↵
    <Оператори n>]
 [Case Else ↵
    <допълнителни оператори>]
End Select ↵
```


ПРИМЕР: ДИАПАЗОНИ ВЪВ ВБ

Диапазоните на ВБ са най-развити. Всеки диапазон се състои от произволен брой разделени със запетаи (,) елементи от следните три вида:

- <произволен израз>, вкл. константа;
- интервал – <израз От> To <израз До>;
- сравнение – Is <знак> <израз>.

ОПЕРАТОРИ ЗА ПОВТОРЕНИЕ

Възможността за многократно повторение на част от програмата е сред най-големите блага на програмирането: малко човешки труд (писане) води до много компютърен труд и пот по челото на ЦП (изпълнение).

Няма ЕПВР, който да не осигурява оператори за повторение на програмен участък.

Повторението на участък е и доста опасно:

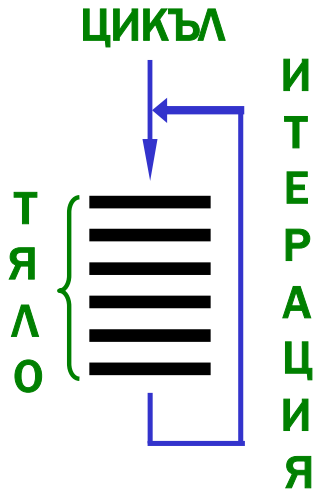
- ами ако не спре (безкрайно повторение)?
- правописна грешка във Фортран превръща оператора за повторение DO 100 I = 1, 5 в безобиден оператор за присвояване DO100I = 1.5.

НЯКОИ ОПРЕДЕЛЕНИЯ

Повторението се нарича ***цикъл***.

Повтарящият се участък се нарича ***тяло на цикъла***.

Еднократното изпълнение на повтарящия се **участък** (тялото) се нарича ***итерация***.



ПРОГ_13 25/45

ВИДОВЕ ЦИКЛИ

Съществен елемент на циклите е **определянето на** момента, в който ще трябва да последва ***нова итерация***.

Това може да бъде **реализирано по 2 начина**:

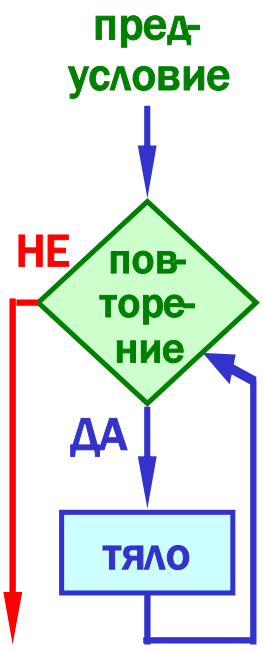
- **проверка на условие** чрез изчисляване на израз от логически тип – **цикли по условие**;
- **преброяване** на итерациите – **цикли с преброяване**.

От своя страна **проверката на условието за повторение** може да бъде осъществена:

- **до изпълнение на тялото** – **предусловие**;
- **след изпълнение на тялото** – **следусловие**.

ПРОГ_13 26/45

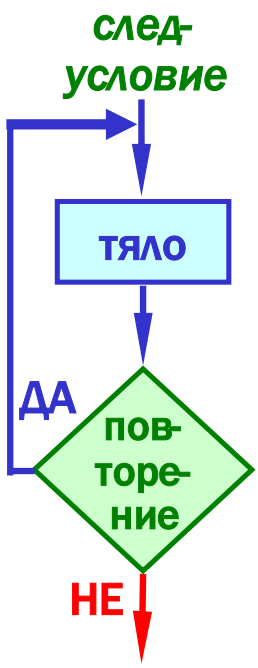
ЦИКЛИ С УСЛОВИЕ



Може би те са **по-важни**, защото **циклите с преброяване могат да се реализират** чрез тях.

Условието за повторение е израз от **логически тип**.

Тялото на цикъла, **трябва да променя** текущите стойности на **променливите**, участващи при изчисляване на условието за повторение.



ПРОГ_13 27/45

ЗАБЕЛЕЖКИ

❶ **Тялото на цикъл с предусловие може да не бъде изпълнено нито веднъж**, когато **при достигане** на цикъла **още първото изчисляване на условието** покаже **липса на повторение**.

❷ **Тялото на цикъл със следусловие се изпълнява поне веднъж**, защото **условието** за повторение **се изчислява след** неговото изпълнение.

❸ **Циклите с предусловие** решават **около 95%** от необходимите случаи.

ПРОГ_13 28/45

ПРИМЕР: ЦИКЛИ С УСЛОВИЯ В ПАСКАЛ

```
WHILE <Условие> DO <Оператор>; пред
Итерация:        еквивалент на оператора
IF <Условие> THEN BEGIN;
   <Оператор>; GOTO Итерация; END

REPEAT <Оператори> UNTIL <Условие>
Итерация: <Оператори>        еквивалент
IF NOT <Условие> THEN GOTO Итерация;
```

while = докато; until = докогато

ПРИМЕР: ЦИКЛИ С УСЛОВИЯ В СИ

```
while (<Условие>) <Оператор> пред
Итерация:        еквивалент на оператора
if (<Условие>)
   {<Оператор> goto Итерация}

do <Оператор> while (<Условие>) след
Итерация: <Оператор>        еквивалент
if (<Условие>) goto Итерация;
```

Еднотипна проверка на условието!

ПРИМЕР: ЦИКЛИ С ПРЕДУСЛОВИЕ ВЪВ ВБ

```
Do While <Условие> ↵        при истина
  <Оператори>
Loop ↵

While <Условие> ↵           също при истина
  <Оператори>
Wend ↵

Do Until <Условие> ↵        при лъжа
  <Оператори>
Loop ↵
```


ПРИМЕР: ЦИКЛИ СЪС СЛЕДУСЛОВИЕ ВЪВ ВБ

```
Do ↵
  <Оператори>
Loop While <Условие> ↵    истина

Do ↵
  <Оператори>
Loop Until <Условие> ↵    лъжа
```


ЦИКЛИ С ПРЕБРОЯВАНЕ

В редица случаи броят на повторенията е известен преди започване на цикъла (например при еднотипната обработка на всички елементи на един масив).

Тогава вместо да проверяваме условие е по-лесно да преброим повторенията.

Броенето може да бъде:

- нормално – по нарастване;
- обратно – по намаляване (предстартово броене).

ПРОГ_13 33/45

ЗАБЕЛЕЖКИ

❶ От съществено значение е в тялото на цикъла да знаем номера на повторението. За целта е подходящо да има променлива, която да брои повторенията. Тя се нарича управляваща променлива на цикъла.

❷ За улесняване на авторите на езикови процесори стойността на управляващата променлива след завършване на цикъла не е определена явно и точно.

❸ Броенето е еквивалентно на преминаване към следваща/предишна стойност (± 1).

ПРОГ_13 34/45

ЗАБЕЛЕЖКИ (прод.)

❹ За „броене“ е подходящо управляващата променлива да бъде от произволен дискретен тип, но не и от приближен.

❺ Целите числа позволяват обобщаване: увеличаване/намаляване с произволно избрано цяло число.

❻ В обобщения случай последователните стойности на управляващата променлива образуват аритметична прогресия с разлика, различна от ± 1. Това обяснява и другото име на подобен цикъл – цикъл за аритметична прогресия.

ПРИМЕР: ЦИКЪЛ С ПРЕБРОЯВАНЕ

Паскал: `FOR` <пром.> `:=` <начало> {`TO` | `DOWNTO`} <край> `DO` <оператор>

<пром.> `:=` <начало>`;` {еквивалент при TO}

`WHILE` <пром.> `<=` <край> `DO BEGIN;`
<оператор>`;` <пром.> `:= SUCC(`<пром.>`);`
`END;`

Си: вместо такъв оператор има по-общ

`for (`<начало>`;` <проверка>`;` <промяна>`)`
<оператор>

<начало>`;` `while (`<проверка>`)` еквивалент
`{` <оператор> <промяна> `}`

ПРИМЕР: ЦИКЪЛ С ПРЕБРОЯВАНЕ ВЪВ ВБ

```
For <упр. пром.> = <начало> To <край>
    [Step <стъпка>] ↵
  <оператори>
Next [<управляваща променлива>]
```

Еквивалент при <стъпка> > 0:

```
<упр. пром.> = <начало>
Do Until <упр. пром.> > <край> ↵
  <оператори>
  <упр. пром.> = <упр. пром.> + <стъпка>
Loop ↵
```


ПРИМЕР: СПЕЦИАЛНО ПРЕБРОЯВАНЕ ВЪВ ВБ

```
For Each <Елемент> In <Група>↵
  <Оператори>
Next [<Елемент>]↵
```

<Елемент> е име на обектова променлива (указател), а <Група> е колекция (специален клас) от обекти от съответен на променливата клас.

ПРЕКЪСВАНЕ НА ЦИКЪЛ

В тялото на един цикъл могат да бъдат констатирани **условия**, които:

❶ **обезсмислят** продължаване на **изпълнението на цикъла**;

❷ **изискват незабавно прекъсване на** поредната **итерация** и преминаване **към следващата итерация**.

Решение на тези въпроси се **осигурява** от оператор **go to със специфичен етикет**.

Затова **много езици** предлагат **скрит go to**.

ПРИМЕР: СКРИТИ GO TO

Паскал: **няма** такива оператори.

Си: `break;` – изход от цикъл и оператор `switch`.

`continue;` – нова итерация.

ВБ: `Exit {Do | For}` ↵

изход от цикъл – **Exit For** се използва **и при For Each.**

ВЛОЖЕНИ ЦИКЛИ

Операторите за цикъл могат да се влагат един в друг, както и всички други структурни оператори.

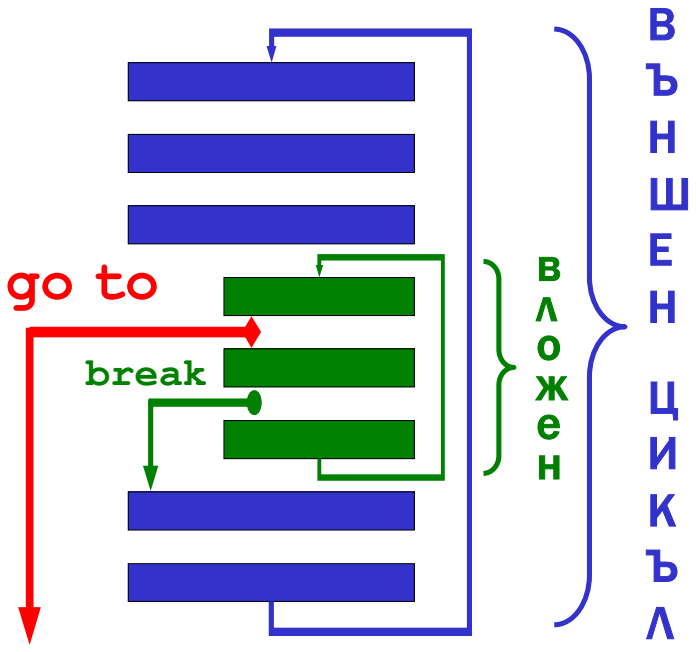
Цикълът, който е **записан в тялото да друг**, се нарича ***вложен*** (вътрешен) ***цикъл***.

Цикълът, който **съдържа в тялото си друг**, се нарича ***външен*** (обхващащ) ***цикъл***.

Скритите оператори go to дават възможности за **напускане само на един цикъл**, т. е. те **не позволяват от** тялото на един **вложен цикъл да се напуска външният цикъл**.

Това е и единственият случай, в който трябва да **се използва оператор go to**.

ПРИМЕР: ВЛОЖЕНИ ЦИКЛИ И GO TO



БЕЗКРАЙНИ ЦИКЛИ

Безкрайните цикли (чието повтаряне никога не завършва) по принцип **са опасни**.

Безкраен цикъл, **може** да бъде записан **във всеки език**.

Операторите за безусловен преход (явни и неявни) **дават възможност безкрайните цикли** все пак **да завършват** своите повторения.

Явното записване на безкраен цикъл повишава вниманието на четящия програмата да провери **дали вътре в тялото има условие за край** (**go to**, **break**, **exit**).

ПРИМЕР: БЕЗКРАЙНИ ЦИКЛИ

Паскал: `WHILE TRUE DO …;`

`REPEAT … UNTIL FALSE;`

Си: `while () …` + break!

`for (;;) …` + break!

ВБ: `Do` ↵

`. . .` + Exit Do!

`Loop` ↵

БЛАГОДАРЯ ВИ
ЗА ВНИМАНИЕТО!

БЪДЕТЕ С МЕН И
В СЛЕДВАЩАТА ЛЕКЦИЯ,
КОЯТО ЩЕ НИ ОТВЕДЕ
В НЕВЕРОЯТНИЯ СВЯТ НА
ПРОГРАМНИТЕ ЧАСТИ

