mnknowledge

ПОДГОТОВКА НА УЧИТЕЛИ ПО ИНФОРМАТИКА /СПЕЦИАЛИЗИРАНО ОБУЧЕНИЕ ПО C#/

д-р Красимир Манев

д-р Марияна Райкова

Пано Панов

С ПОДКРЕПАТА НА: DIGITAL LIGHTS

Компютърно програмиране. Езици, системи и среди за програмиране

Sofia, 2022

В тази тема ще стане дума за:

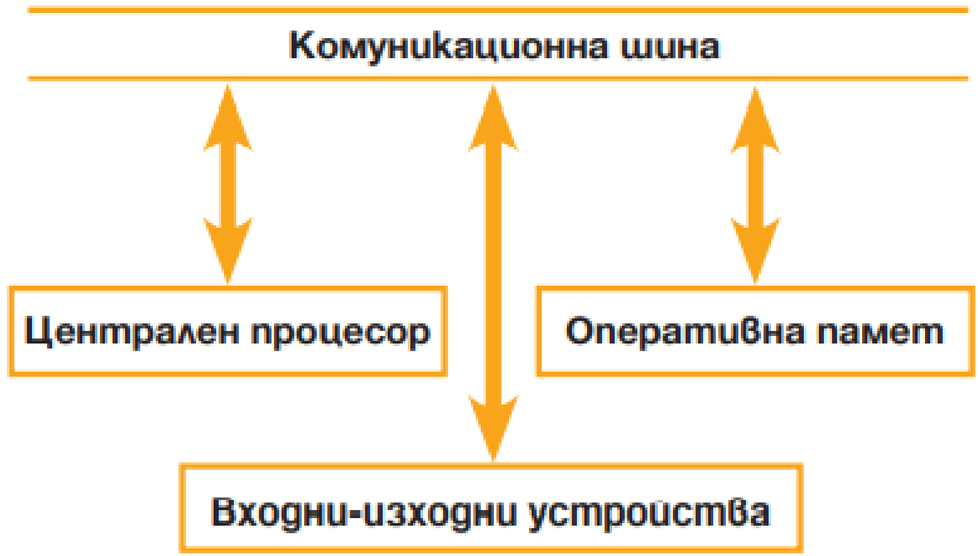
- Основните елементи на компютъра, предназначени да осигурят изпълняването на програми.
- Език за програмиране и видовете езици за програмиране
- Елементите на една система за създаване на компютърни програми
- Интегрирането на елементите на система за програмиране в среда за програмиране
- Алгоритмите – основа на компютърното програмиране

Компютърната система

- *Компютърните системи* са симбиоза от *апаратни средства* (*хардуер*) и *програмни средства* (софтуер), като всяка от двете компоненти има своето решаващо значение за функционирането на компютъра. Този курс е посветен на създаването на *компютърни програми* (за по-кратко оттук нататък ще казваме просто *програми*).
- Съвкупността от софтуера на една компютърна система можем да разделим на две групи, в зависимост от кръга потребители, за които представляват интерес. Едни програми са полезни за всеки потребител – наричаме ги системен софтуер, а други са полезни само за отделни потребители – приложен софтуер.
- В системния софтуер също могат да се обособят две групи програми. В първата група са комплекс от взаимно свързани програми, наричани *операционна система* (ОС), предназначението и основните функции на които учениците познават от уроците по ИТ.
- В другата група програмите са независими една от друга и се наричат *инструментални програми*. Инструменталните програми решават сериозни задачи, подпомагат работата предимно на квалифицирани специалисти в областта на информатиката и затова се считат за част от системния софтуер. В този курс ще се запознаем с инструменталните програми предназначени за създаване на програми.
- За целите ни е важно да разгледаме какво представлява хардуерът от гледна точка на програмиста.

Компютърната система

- Компютърният хардуер е много сложна машина предназначена за автоматизирано изпълнение на програми (модерно въплъщение на идеята на Жакард).
- За целите ни не е необходимо да познаваме в подробности устройството на хардуера. Затова ще го представим в най-общи черти – представяне, което наричаме фон Нойманова архитектура.
- Четири са основните компоненти на фон Нойманова компютърна архитектура:
  - *Оперативната памет* (в която, следвайки Атанасов, се сменят данни и програми);
  - *Централният процесор* (който изпълнява програмите);
  - Разнообразни входно-изходни устройства, за въвеждане/съхраняване и извеждане на данните в и от ОП;
  - Комуникационната шина, през която се осъществява обменът на данни между ЦП, ОП и В/И устройства.



Компютърната система

- Оперативната памет е последователност от елементи, които могат да съхраняват само две стойности 0 и 1 – *битове*. Всеки 8 последователни бита на ОП образуват байт. Тъй като всеки бит на байта може да заема стойност 0 или 1, в един байт (B) можем да запишем 2.2.2.2.2.2.2.2 = 2^8 = 256 различни стойности.
- Байтовете на ОП са номерирани с естествени числа, които наричаме адреси.
- За определяне размера на ОП (както и на В/И устройства за съхраняване на данни) се използват специализирани мерни единици – 1024 B = 1 кибибайт (KiB), 1024 KiB = 1 мебибайт (MiB), 1024 MiB = 1 гибибайт (GiB), 1024 GiB = 1 тибибайт (TiB) и т.н. **Забележка.** С директива на ЕС от 2008 г. се препоръчва да се избягвя традиционното наричане килобайт (означаващо 1000 B), мегабайт (1000 KB), гигабайт и т.н. като не съответстващи на реалността.
- За целите на програмирането, в компютърната архитектура са определени няколко вида полета от паметта с различна дължина – 1, 2, 4, и т.н. байта, съдържанието на които се интерпретира като някаква данна – знак, малко цяло число - със знак или без знак, голямо цяло число – със знак или без знак, дробно число и т.н. С всяко такова поле свързваме понятието *базов тип данна*, с което ще се срещаме често по-нататък. Мястото на полето в паметта се определя от адреса на най-младшия му байт.

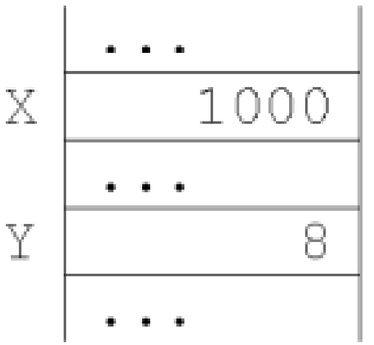
Компютърната система

- Съгласно принцип на фон Нойман, компютърът е машина с програмно управление.
- Устройството, което осъществява програмното управление на компютъра, е *Централният процесор* (ЦП).
- ЦП изпълнява *инструкции*, всяка от които предписва някаква *операция* с данни от ОП. *Кодът* на инструкцията определя операцията, а *аргументите* – данните, с които да се извърши предписаната операция.
- Инструкциите, както и данните се съхраняват в ОП. Както и при данните, мястото на всяка инструкция в паметта се определя от нейния *адрес*. Поредица от инструкции наричаме *програма*. **Забележка.** За да не усложняваме нещата, в примерите по-долу, за адреси на инструкциите използваме последователни естествени числа.
- Когато ЦП изпълнява програма, той извлича инструкция и зададените в нея аргументи от ОП, стартира схемата за извършване на операцията, връща резултата в ОП и определя следващата за изпълнение инструкция – *основен цикъл на хардуера*.
- Системата от инструкции на компютъра и правилата за изпълнението им наричаме *машинен език*.

Компютърната система

- На Фигурата е показана абстрактна програма от пет инструкции и част от ОП. Числото преди всяка инструкция е нейният адрес
- За удобство на програмиста, двоичните кодове на инструкциите и адресите на данните са представени с *напомнящи имена*.
- След изпълнението на всяка от тези инструкции ЦП преминава към изпълнение на следващата инструкция.
- Първата инструкция предписва да се съберат целите с адреси X и Y, а резултатът да се съхрани в полето с адрес X. Така съдържанието на X ще стане 1008 и ЦП ще премине към втората инструкция.
- Тя прибавя 5 към съдържанието на полето X и то става 1013. Знакът # в името означава, че вторият аргумент е стойност, а не адрес.
- Третата инструкция увеличава съдържането на Y с единица, а четвъртата – заменя съдържанието на X с това на Y.
- Инструкцията КРАЙ е указание за процесора да преустанови работата на програмата.

```
1000 СЪБЕРИ X Y
1001 СЪБЕРИ# X 5
1002 УВЕЛИЧИ Y
1003 ПРЕМЕСТИ X Y
1004 КРАЙ
```



Компютърната система

- Всеки машинен език има инструкции за *разклоняване* на програмата. На Фигурата (горе) са показани примери на такива инструкции.

```
ПРИ=0 X A
ПРИ>=0 X A
СКОЧИ A
```

- Инструкцията с код ПРИ=0 предписва да се провери съдържанието на първия аргумент и ако то е 0, ЦП трябва да премине към изпълнение на инструкцията с адрес във втория аргумент – *условен преход*. В противен случай, т.е. ако X не е нула, ЦП трябва да продължи със следващата инструкция на програмата в ОП.
- Инструкцията с код СКОЧИ предписва на ЦП да премине към изпълнение на инструкцията, чийто адрес е зададен като аргумент – *безусловен преход*.
- На Фигурата (долу) е показана програма, която намира и показва на екрана (инструкцията ПОКАЖИ) абсолютната стойност на сумата на числата, намиращи се в паметта на адреси X и Y. Проследете работата на програмата, ако X = 3 и Y = –7.

```
1000 СЪБЕРИ X Y
1001 ПРИ>=0 X 1003
1002 УМНОЖИ# X -1
1003 ПОКАЖИ X
1004 КРАЙ
```

Компютърната система

- Водени от убеждението, че всеки начинаещ програмист трябва да познава концепцията машинен език, да развием нашия абстрактен машинен език до състояние да може да се пишат на него несложни програми.
- Инструкциите като СЪБЕРИ X Y, където X и Y са адреси на две полета от паметта, съдържащи цели числа, наричаме *с пряка адресация*.
- Да добавим към нашия машинен език и инструкциите с пряка адресация ИЗВАДИ X Y, УМНОЖИ X Y, РАЗДЕЛИ X Y (за целочислено деление) и ОСТАТЪК X Y (за намиране на остатъка при целочислено деление на съдържанието на адресите X и Y).
- Към инструкцията с пряка адресация УВЕЛИЧИ X да добавим аналогична инструкция НАМАЛИ X, която намалява стойността намираща се на адрес X с 1.
- Към инструкцията с пряка адресация ПОКАЖИ X да добавим аналогична инструкция ВЪВЕДИ X, която въвежда от клавиатурата цяло число в полето с адрес X.

Компютърната система

- Инструкциите като СЪБЕРИ# X V, където X е адрес на поле от паметта, а V е цяло число, наричаме *с непосредствен операнд*.
- Да добавим към нашия машинен език и съответни инструкции с непосредствен операнд ИЗВАДИ# X V, УМНОЖИ# X V, РАЗДЕЛИ# X V (за целочислено деление) и ОСТАТЪК# X V (за намиране на остатъка при целочислено деление на съдържанието на адрес X и цяло V).
- Към инструкцията с пряка адресация ПРЕМЕСТИ X Y да добавим аналогична инструкция с непосредствен операнд ПРЕМЕСТИ# X V, която замества стойността намираща се на адрес X с цялото V.
- Към инструкциите за преход да добавим аналогични инструкция ПРИ!=0 X A, ПРИ<0 X A, ПРИ<=0 X A, ПРИ<0 X A.
- Към инструкцията без аргументи КРАЙ, която винаги поставяме само в края на програмата, да добавим и инструкцията СПРИ, която също преустановява работата на програмата, но може да бъде поставяна между другите оператори там, където прекратяване на програмата се налага без да се стига до инструкцията КРАЙ.

Компютърната система

- Много важна част на един машинен език са инструкциите *с косвена адресация*.
- За пример да разгледаме инструкцията с косвена адресация СЪБЕРИ@ X U, където X отново е адрес на първия аргумент, но U е адрес на поле, което съдържа не втория аргумент, а неговия адрес. Такава инструкция ни позволява, като меним съдържанието на полето U, при всяко изпълнение на инструкцията да добавяме към полето с адрес X съдържанията на различни полета. Аналогични инструкции ще имаме и за другите аритметични операции.
- Така с една и съща инструкция ВЪВЕДИ@ U ще можем да въвеждаме цели стойности в различни полета на паметта.
- Да илюстрираме ролята на инструкциите с косвена адресация с програма, която въвежда в последователни адреси на паметта зададени от клавиатурата N цели числа a_0, a_1,…,a_{N-1}, като стойността на N също се задава от клавиатурата, намира и извежда сумата им.

```
1000 ВЪВЕДИ N
1001 ПРЕМЕСТИ# S 0
1002 ПРЕМЕСТИ# X 1013
1003 ВЪВЕДИ@ X
1004 СЪБЕРИ@ S X
1005 СЪБЕРИ# X 1
1006 ИЗВАДИ# N 1
1007 ПРИ>0 N 1003
1008 ИЗВЕДИ S
1009 КРАЙ
1010 // място за N
1011 // място за S
1012 // място за X
1013 // място за a[0]
1014 // място за a[1]
. . .
```

Езици за програмиране

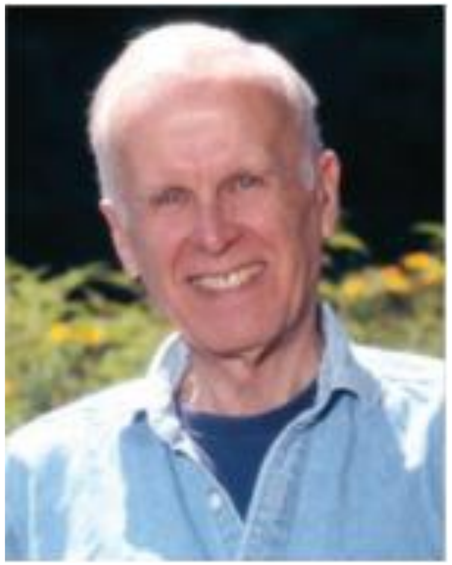
- Основен инструмент за създаване на програми е *езикът за програмиране*.
- Единственият език, който компютърът „разбира“, е неговият машинен език. Първите програми са написани на машинните езици на съответните компютри. Машинните езици обикновено имат повече от 100 инструкции, в които кодовете на операциите и адресите на аргументите се изписват в цифров вид.
- Запомнянето на кодовете на инструкциите, *синтаксиса* им (как се изписва всяка инструкция) и *семантиката* им (какво точно предписва всяка инструкция) е доста трудоемко. За избягване на грешки се налагат чести справки в описанието на езика.
- Освен това програмата, написана на машинния език на една компютърна архитектура, не може да бъде пренесена на друга компютърна архитектура.
- За облекчаване работата на програмистите, за всеки машинен език, се създава *асемблерен език*, при който кодовете на операциите и адресите на операндите се заменят с имена. По-горе, представяйки инструкции и програми, дефинирахме измислен език, който по същество е асемблерен.
- Трансформирането на програма, написана на асемблерен език, в програма на съответния машинен език е просто и се извършва със съответна програма – *асемблер*.

Езици за програмиране

- Програмирането на асемблерен език е доста по-лесно от програмирането на машинен език и до днес не е загубило своята роля.
- С разширяване на сферата на приложение на компютрите, в 60-те години на миналия век се появяват езици, първоначалната цел на които е програмистите да обменят алгоритми помежду си, затова отначало са наричани *алгоритмични*.
- Алгоритмичните езици са от *по-високо ниво* от асемблерните. Един *оператор* на такъв език замества няколко машинни инструкции.
- Тези езици се доближават до човешкия, защото операторите им се съставят от разбираеми за човека фрази – if…then… (ако… то…), while…do… (докато … прави …). Програмите се четат по-лесно, улеснява се обменът на алгоритми и обучението на програмисти. Затова те изместват асемблерните и се наричат езици за програмиране.
- До идеята за такъв език пръв е достигнал Конрад Цузе , създателят на първите реално действащи програмируеми, но не електронни, изчислителни устройства. През 1945 г. той проектира езика Plancalcul, който остава незавършен, но редица елементи от този проект се срещат по-късно в много езици за програмиране.

Езици за програмиране

- През 1954-57 г., под ръководството на Джон Бекъс (горе), в IBM е разработен езикът FORTRAN (FORmula TRANslation, превод на формули). Предназначен поначало за числени пресмятания, по-късно той се използва като универсален език и оказва голямо влияние върху развитието на езиците за програмиране.
- Езикът FORTRAN претърпя голяма еволюция, като следващите му версии FORTRAN II, FORTRAN IV, FORTRAN 77, FORTRAN 85 отразяваха развитието на езиците за програмиране, а най-новата му версия FORTRAN 9x се използва и до днес.
- По същото време, колектив от американската армия, под ръководството на адмирал Грейс Хопър (долу), работи над създаването на езици за програмиране за обработка на икономическа информация. Най-сполучлив от тях е езикът COBOL (COmmercial and Business Oriented Language, език за бизнес приложения). Счита се, че 60% от софтуера за управление на бизнеса, който работи и в момента, е написан на COBOL.



Езици за програмиране

- В края на 50-те години специален научен комитет се занимава с разработването на принципите на универсален език за програмиране, който да не е ориентиран само към един тип задачи. Създаденият от този комитет език **Algol** (ALGOrithmic Language, алгоритмичен език) не успява да се наложи в практиката, но оказва голямо влияние върху развитие на езиците за програмиране – наричан е „латинският на езиците за програмирне“.
- Дълго време, чак до наши дни, много популярни бяха езиците **BASIC** (създаден от Джон Кемени и Томас Курц) и Pascal (създаден от Никлаус Вирт).
- Важен за развитието на езиците за програмиране е езикът **C** създаден в началото на 70-те години от Брайън Кернигън (горе) и Денис Ричи (долу). Създаден е с идеята на него да се създават програми, близки по качества до тези, написани на асемблерен език. Много от най-използваните днес езици за програмиране – C++, Java, Perl, C# и др. – са много силно повлияни от C.

Транслатори

- Както вече споменахме, компютърът разбира само своя машинен език и затова програмите написани на други езици трябва да се преведат на машинен.
- FORTRAN имал успех, защото Бекъс&Co създали програма – *транслатор*, която превеждала програмите от FORTRAN на машинен език. Разработките на Бекъс предизвикват революция в програмирането.
- Различаваме два вида транслатори. Първият вид са *компилаторите,* които избират за всяка конструкция на езика подходящ фрагмент на машинен език, и от всички фрагменти съставят (компилират) програма. В резултат на работата на компилатора се получава пълен превод на *изходния текст* (*изходния код*, source code) на програмата на машинен език – *изпълнима програма* (executable).
- *Интерпретаторите* също подбират последователност от машинни инструкции за всеки от операторите, но тя се изпълнява веднага (операторът се интерпретира), след което се преминава към интерпретиране на следващия оператор в програмата. Когато оператор се достигне още веднъж, ще бъде преведен и интерпретиран отново.
- Интерпретатори се създават много по-лесно от компилаторите. Интерпретирането на програмата обаче е много по-бавно от работата на компилираната програма.

Стандартни подпрограми

- При програмиране често се налага един и същ програмен код, след замяна на стойностите на някои негови елементи (наричани *параметри*), да се използва отново и отново.
- За да не се налага програмистите да пишат този код многократно, такива парчета код се компилират и оформят по подходящ начин – наричаме ги подпрограми.
- Освен това, програмирането на работата с входно-изходните устройства, предвид тяхното разнообразие по вид и производител, е трудно. Не е по силите на никой програмист да се справи лесно с това. При въвеждане на нови устройства, се налага кодовете за управлението им (почти винаги на асемблерен език) да се създадат еднократно от познавачи на устройствата и също да се оформят като подпрограми.
- Затова е естествено с всеки език да за програмиране да се предлагат една или повече *библиотеки стандартни програми* (в компилиран вид) – както универсални (с най-общо предназначение), така и специализирани (за компютърна графика, за работа в мрежа и т.н.).
- Когато компилираната програма ще използва стандартни подпрограми, те трябва да се *свържат* с програмата. Това прави програмата *свързващ редактор*, която в съвременен вариант съпровожда съответния компилатор.

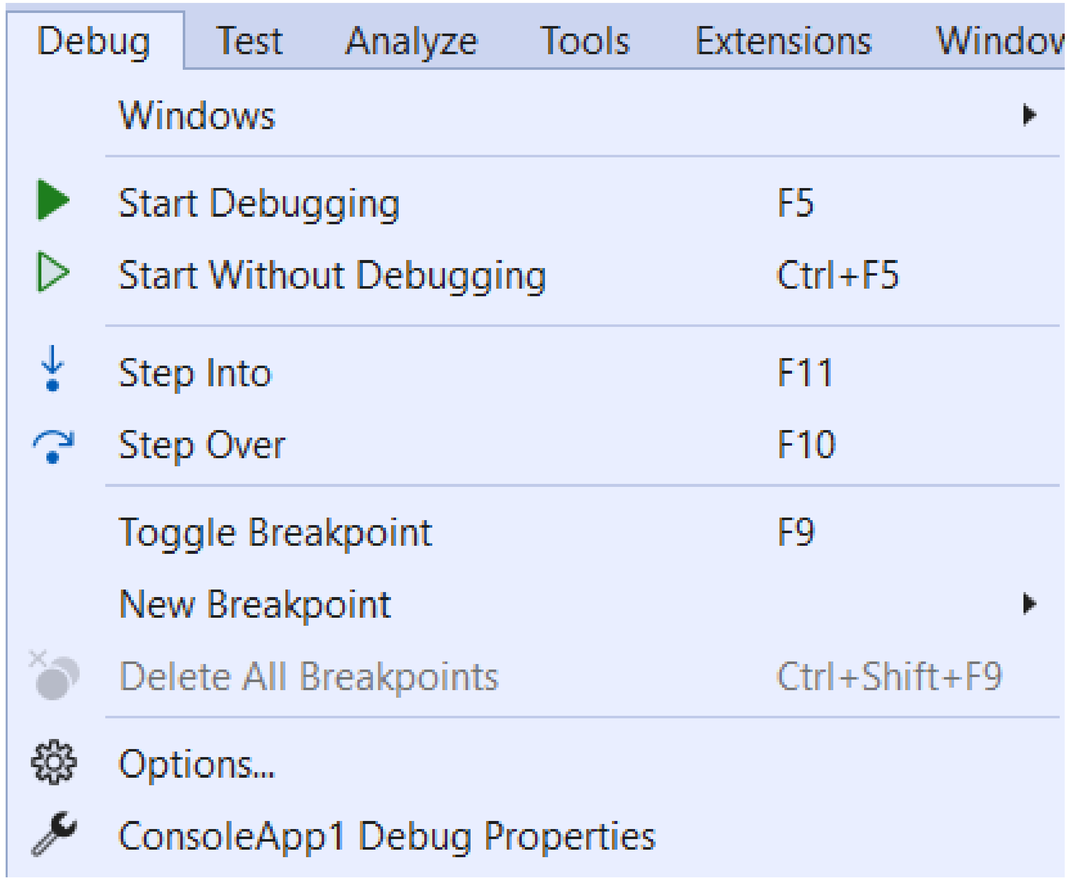
Текстов редактор

- Основен елемент от създаването на една програма е написването на изходния текст на програмата.
- Това става с обичайното средство за работа с текстове – *текстов редактор*.
- Тъй като добре познаваме тези инструментални програми от уроците по ИТ, тук няма да се спираме подробно на тях. Ще отбележим само съществуването на специализирани текстови редактори за създаване на програми, които предлагат редица допълнителни иполезни за програмиста възможности:
  - Оцветяват различните типове лексични елементи на програмата в различни цветове.
  - Подреждат кода така, че блоковете на програмата да са ясно видими, което да облекчи четенето на кода;
  - текстовите редактори, специфични за
  - Един ЕП извършват синтактичен анализ при въвеждане на текста и дават подсказки за грешки и т.н.

```
1   using System;
    0 references
2   class Program
3   {
        0 references
4       static void Main(string[] args)
5       {
6           int a, b, c; string s;
7           s = Console.ReadLine(); a = int.Parse(s);
8           s = Console.ReadLine(); b = int.Parse(s);
9           Console.WriteLine("Смарт:");
10          Console.WriteLine("a={0}\n b={1}", a, b);
11          c = a; a = b; b = c;
12          Console.WriteLine("Ново")
13          Console.WriteLine("a={0}\n b={1}", a, b);
14      }
15  }
```

Дебъгер

- След създаването на програма, която да е работоспособна, не е изключено тя да не дава исканите резултати, т.е. да е логически неправилна.
- Изключително тежкият етап на проверка на кода и изчистването му от логическите грешки може да бъде улеснен от специална програма – *дебъгер* (от debug, „изчистване на буболечки“, т.е. грешки в компютърна програма).
- Дебъгерът има следните възможности:
  - Програмистът може да посочи редове в кода, преди изпълнението на които програмата да спира – т.н. *контролни точки* (breakpoints).
  - При спряла в контролна точка програма, може да се продължи изпълняването ѝ стъпка по стъпка докъдето е нужно, след което да се продължи нормално изпълнение до следваща контролна точка;
  - С дебъгера може да се наблюдават стойности на променливите след всяко спиране така, че да се намери къде е допусната грешка.



Препроцесор

- За много от езиците за програмиране съществува специализирана инструментална програма – *препроцесор*, която обработва програмата, преди да бъде предоставена на компилатора.
- В резултат от изпълнение на *директиви-те* на препроцесора в кода на програмата се извършват някакви изменения – добавя се код или се игнорира част от кода.
- Типични директиви на препроцесора са:
  - Дефиниране/вдигане на булев „флаг“;
  - Включване на фрагмент от кода, когато някакъв „флаг“ (DEBUG) е вдигнат;
  - Игнориран на фрагмент от кода, когато съответният „флаг“ не е вдигнат;
  - Включване на един от няколко алтернативни фрагменти от кода, в зависимост от това кои флагове са вдигнати и т. н.

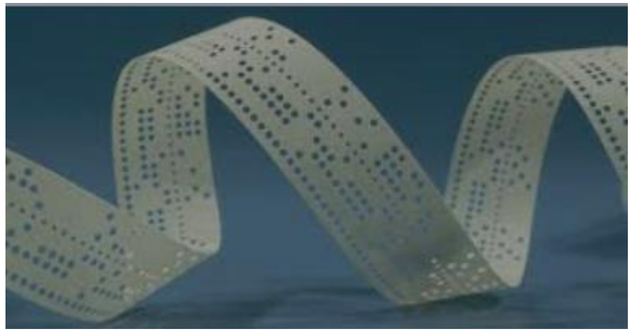
```
1   #define DEBUG
2   using System;
    0 references
3   class Program
        0 references
4   {   static void Main(string[] args)
5       {
6           . . .
7
8   #if DEBUG
9           Console.WriteLine("Старо:");
10          Console.WriteLine("a={0} b={1}", a, b);
11  #endif
12          c = a; a = b; b = c;
13          Console.WriteLine("Ново:");
14          Console.WriteLine("a={0} b={1}", a, b);
15      }
16  }
```

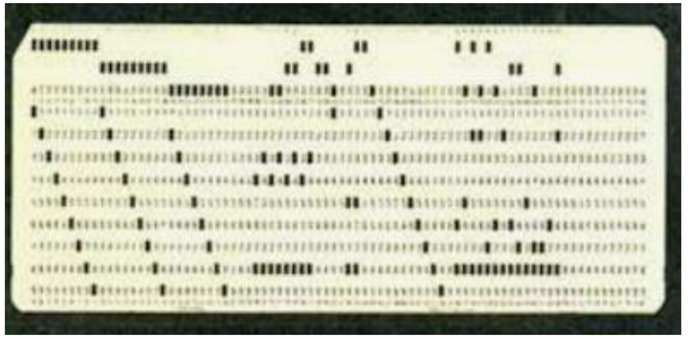
Система и среда за програмиране

- Съвкупността от всички инструментални програми – текстов редактор, препрцесор, компилатор, свързващ редактор, библиотеки стандартни продпрограми, дебъгер и т.н. наричаме *система за програмиране*.
- В зората на програмирането компонентите на системата за програмиране са самостоятелни и независими една от друга програми, които програмистът извиква за изпълнение според това, на какъв етап е от създаването на програмата.
- И в наши дни има програмисти, които предпочитат да използват програмите на системата за програмиране по този начин.
- По-късно, за облекчение на програмистите, компонентите на системата за програмиране започват да се интегрират в едно приложение, което наричаме *интегрирана среда за програмиране* (Integrated Development Environment, IDE).
- Едни от първите популярни интегрирани среди за програмиране с буквено-цифров интерфейс, предназначени за работа само с един език бяха Turbo Pascal и Turbo C.
- Постепенно интегрираните среди с графичен интерфейс (Delphi за Pascal) изместиха тези с буквено-цифров интерфейс, а съвременните IDE вече позволяват работа с няколко езика.
  В следващи лекции ще се запознаем с работата в интегрираната среда за програмиране с графичен интерфейс MS Visual Studio.

История на програмирането

- Първите програмисти – учени и университетски преподаватели създаваха програми за собствени нужди на машинен или асемблерен език.
- Програмистът често беше и единствен потребител, затова не се налагаше нито да се планират логика и функционалност, нито да се документират - *манифактура*.
- Проверката за работоспособност на програмата се извършваше в процеса на използване и когато се установяваха грешки, авторът-потребител внасяше необходимите корекции.
- Не се използваше софтуерен инструментариум. Програмите се пишеха на хартия, перфорираха се върху перфоленти или перфокарти, както и входните данни и се въвеждаха в компютъра от устройства за четене на такива носители.
- Резултатите от работата на програмата се отпечатваха от механични печатащи устройства върху хартиена лента.





История на програмирането

- С изобретяването на езиците за програмиране от по-високо ниво, писането на програми стана значително по-лесно. Възникна професията *програмист*. Програмистите пишат освен програми за свои нужди – компилатори, ОС и т.н. – и програми за нуждите на други потребители.
- Необходимостта от програми става все по-голяма, а програмите – все по-сложни. От манифактура се преминава към по-съвършеното *технологично програмиране*.
- Налага се стилът *структурно програмиране*. Сложните програми се разбиват на модули (подпрограми) и написването им може да бъде възложено на различни програмисти.
- Налага се подпрограмите да бъдат документирани, както и по-сложните програми, за да може потребителите да работят с тях без помощта на създателите. Така програмите се превръщат в *програмни продукти* - стока, с тях се търгува, както с всяка друга стока.
- Неизменни етапи при създаване на програми са *тестването* и *съпровождането* им. При тестването се откриват и поправят грешки, добаят се удобствата за потребителя и др.
- По време на съпровождането се отстраняват грешки, неоткрити при тестването. Внасят се изменения, например при промени в нормативен документ и др.
- Затова, много от програмните продукти първоначално се разпространяват в пробни версии (алфа версия, бета версия и т.н.) и периодично се обновяват (update).

История на програмирането

- Истинска революция в технологията на програмирането предизвиква създаването на *компютърния терминал*, състоящ се от *входно устройство*, от което да се въвежда текстът на програмите и командите към ОС и *изходно устройство*, на което да се извежда това, което работещият на терминала въвежда („ехо“), както и резултатите от изпълнението на програмата.
- Първите компютърни терминали са управлявани от компютъра пишещи машини. Отпада използването на перфоленти и перфокарти.
- Терминалите пишещи машини скоро са заменени от монитор и клавиатура, с което всички дейности на програмиста се извършват от терминала много по-удобно.
- Създават се инструменталните програми за подпомагане работата на програмиста – редактори, дебъгери. От 2-3 дневно броят на компилациите и тестовите изпълнения стига до стотици.

История на програмирането

- С въвеждането на *графичните монитори*, и най-вече на цветните графични монитори, се появява възможността за създаване на програми с графичен интерфейс.
- Програмите с графичен интерфейс се изграждат по еднотипен начин. При стартирането си такава програма отваря на екрана един основен *прозорец*, в който са изобразени елементи на графичния интерфейс – *бутони*, *менюта*, *текстови и комбинирани текстови кутии* и др. С текстовите кутии и менютата, например, потребителят задава данни на програмата, а с бутоните и менютата - посочва действията, които програмата трябва да извърши.
- По време на работа, за изпълнение на една или друга функция, програмата може да отвори и други – работни или диалогови – прозорци.
- Средата за програмиране MS Visual Studio, запознаването с която ни предстои, както и програмите, които познаваме от уроците по ИТ, са програми с графичен интерфейс.
- Създаването на програма с графичен интерфейс може да се осъществи с класически форми на програмиране. При него се използват стандартни подпрограми за създаване на графика, за да се изчертават прозорците, като за всеки прозорец се помни мястото му на екрана, каква част от него се вижда във всеки момент и т.н. И аналогично за останалите компоненти на интерфейса, но това е много трудоемко.

История на програмирането

- Затова, при създаване на приложни програми с графичен интерфейс, се използва по-подходящ стил за програмиране, наричан *визуално програмиране* и съответни инструментални програми.
- За създаване на графичния интерфейс се използват готови *графични компоненти*, със задължителните за работата им фрагменти програмен код, а за програмиста остава само да направи някои настройки на компонентите – например по място на екрана, по големина и цветово оформяне и т.н. Естественият начин за това е, графичните компоненти да са оформени като обекти в език за *обектно-ориентирано програмиране*.
- За разлика от конзолното прприложение, където изпълнението започва и се контролира от една от подпрограмите, наричана главна, програмата с графичен интерфейс отваря главния си прозорец, след което проследява действията на програмиста, наричани *събития* и в съответствие със възникналото от действията на програмиста събитие, извършва необходимите действия. Създаването на такива програми наричаме *събитийно програмиране*.
- Средата за програмиране MS Visual Studio, както показва и името ѝ, е среда осигуряваща възможност както на програми с буквено-цифров, така и с графичен интерфейс, с езици за обектно-ориентирано визуално и събитийно програмиране.

Алгоритми

- Понятието *алгоритъм* е основно за информатиката. То произлиза от името на средноазиатския математик Мохамед ибн Муса ал Хорезми. В съчинението „За индийското смятане“ той разглежда представянето на числата в десетична бройна система и извършването на аритметични операции с тях. В средните векове с algorismus или algorithmus са обозначавали правилата за извършване на тези операции. Постепенно смисълът на понятието се разширява и се достига до съвременното му разбиране, отразено в различни речници както следва:
- *Речник на българския език*, изд. БАН, том I, 1977: Система от правила, които определят последователност от изчислителни операции, прилагането на които над зададени обекти води до решението на дадена задача. Алгоритъм на Евклид.
- *Larousse de la langue Francaise*, Lexis, 1979: Съвкупност от правила или предписания за получаване с краен брой операции на определен резултат.
- *Webster’s New Colegiate Dictionary*, Webster, 1980: Процедура за решаване на математически проблеми (например намиране на най-голям общ делител) с краен брой стъпки, която често съдържа повтарящи се операции.

Алгоритми

- От цитираните описания на понятието алгоритъм можем да извлечем някои основни негови характеристики:
  1. За създаване на алгоритъм е необходимо множество от *обекти* и *операции* с тях. Например, естествените числа с аритметичните операции и сравняването.
  2. Алгоритъмът решава някаква *задача* само с помощта на избраните операции. Например, посочената в едно от описанията задача – намиране на най-голям общ делител (НОД) на две естествени числа. Задаваните обекти се наричат *входни данни* или *вход*, а получаваните обекти – *резултат* или *изход*.
  3. Задачата трябва да е масова, т.е. да има много различни възможни входни данни и алгоритъмът намира решение за всяка възможна комбинация от входни данни. Ако фиксираме входните данни, получаваме *екземпляр* на задачата. Например: „Дадени са числата 12 и 30. Намерете техния НОД.“
  4. Алгоритъмът е *процедура*, задаваща последователност от операции, която, изпълнена над входните данни на екземпляр на задачата, дава очаквания резултат. Алгоритъмът на Евклид е такава процедура за намиране НОД на две положителни цели числа.
  5. Процедурата трябва да е *детерминирана* – който и да я изпълни над едни и същи входни данни, трябва да получи един и същ резултат.

Алгоритми

6. Процедурите, които наричаме алгоритми, трябва да водят до намиране на резултата чрез *краен брой стъпки*, т.е. с прилагане краен брой пъти на допустимите операции. Този брой определя *бързодействието* на алгоритъма. Ако за една задача могат да бъдат построени няколко алгоритъма с различно бързодействие, добре би било да можем да разпознаем най-бързия.

7. В алгоритъма е възможно някои последователности от операции да бъдат повтаряни многократно. Такива повторения се наричат *цикли*. Организирането на цикли е същностна черта на процеса на създаване на алгоритми.

- В предишна лекция разгледахме такива процедури за решаване на задачите:
  - да се представи в двоична бройна система число, зададено в десетична система и обратно;
  - да се намери сумата на две естествени числа в двоична система;
  - да се сравнят по големина две естествени числа в позиционна система;
  - да се преобразува зададено число от осмична в шестнадесетична система.
- В тази лекция пък разгледахме алгоритъм за намиране сумата на зададени N числа, имплементирана като програма на нашия асемблерен език.

Алгоритми

- Естествените езици не могат да бъдат използвани за описване на алгоритми, заради възможност от неясно и двусмислено изразяване. Асемблерният ни език не допуска двусмислици. В следващи лекции, за представяне на алгоритми, ще използваме и езика C#.
- Представянето на алгоритми с програми е крайната ни цел, но умението да се програмира се постига трудно. Затова добре е начинаещите програмисти да свикнат да описват алгоритмите си по по-лесен начин, преди да ги програмират.
- Алгоритмите можем да представяме на ограничен естествен език, който не допуска двусмислие. Да разгледаме три процедури за решаване на уравнението a.x + b = 0.
- Процедурата а. не е алгоритмична – при a == 0 стъпка 3 не може да бъде изпълнена. Процедурата б. е алгоритмична, но неправилна – при a == 0 и b==0 води до неверен резултат. Процедурата в. е правилният алгоритъм за решаване на уравнението.

```
1. Въведи a
2. Въведи b
3. Пресметни x = -b/a
4. Изведи x
5. Край
```

а.

```
1. Въведи a
2. Въведи b
3. Ако a==0 премини_към 6
4. x = -b/a
5. Изведи x и премини_към 7
6. Изведи „Няма решение“
7. Край
```

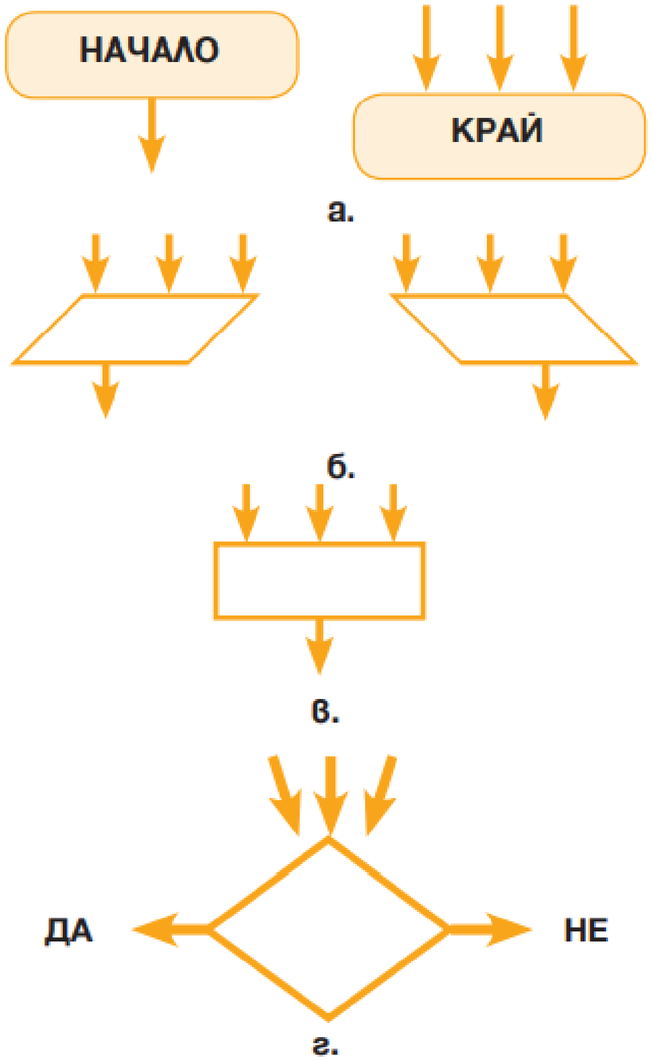
б.

```
1. Въведи a
2. Въведи b
3. Ако a==0 премини_към 6
4. x = -b/a
5. Изведи x и премини_към 8
6. Ако b==0 изведи „Всяко число
е решение“ и премини_към 8
7. Изведи „Няма решение“
8. Край
```

в.

Алгоритми

- *Блок-схемният език* е използван много в зората на програмирането, остава най-подходящото средство за описване на алгоритми. Той е *графичен* и позволява да се представят разклоненията, които при линейното представяне с ограничен естествен език или език за програмиране са по-трудни за проследяване.
- Основните блокове на езика са представени на фигурата, като там където има 1 стрелка трябва да има точно една стрелка, а там където са 3 – може да има 1 или повече стрелки. Стрелката посочва с кой блок трябва да продължи изпълнението на алгоритъма.



а. Блокът НАЧАЛО може да е само един и от там започва изпълнението, а с блокове КРАЙ изпълнението завършва.

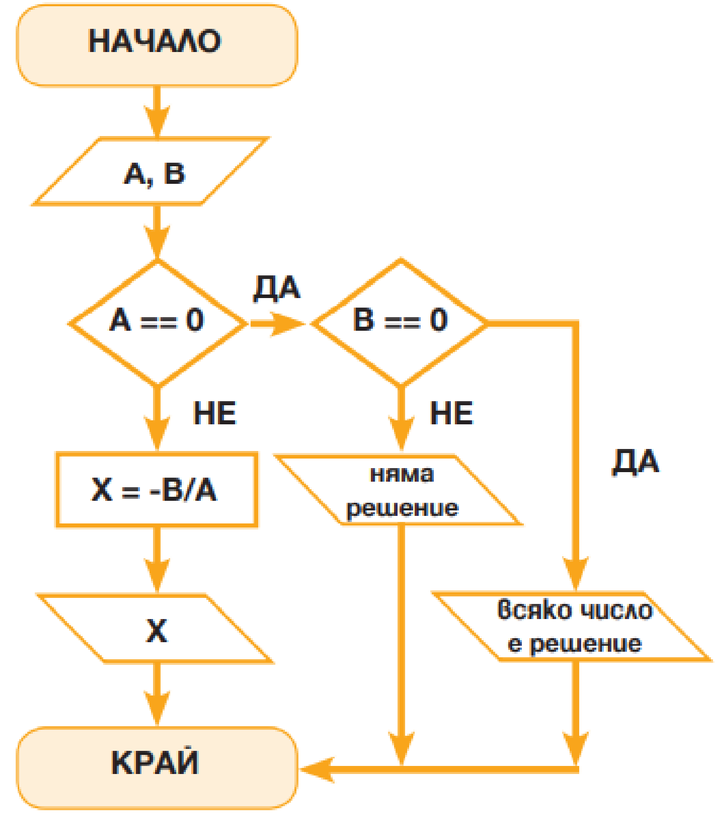
б. В тези блокове описваме данните, които трябва да се въведат (вляво) или изведат (вдясно).

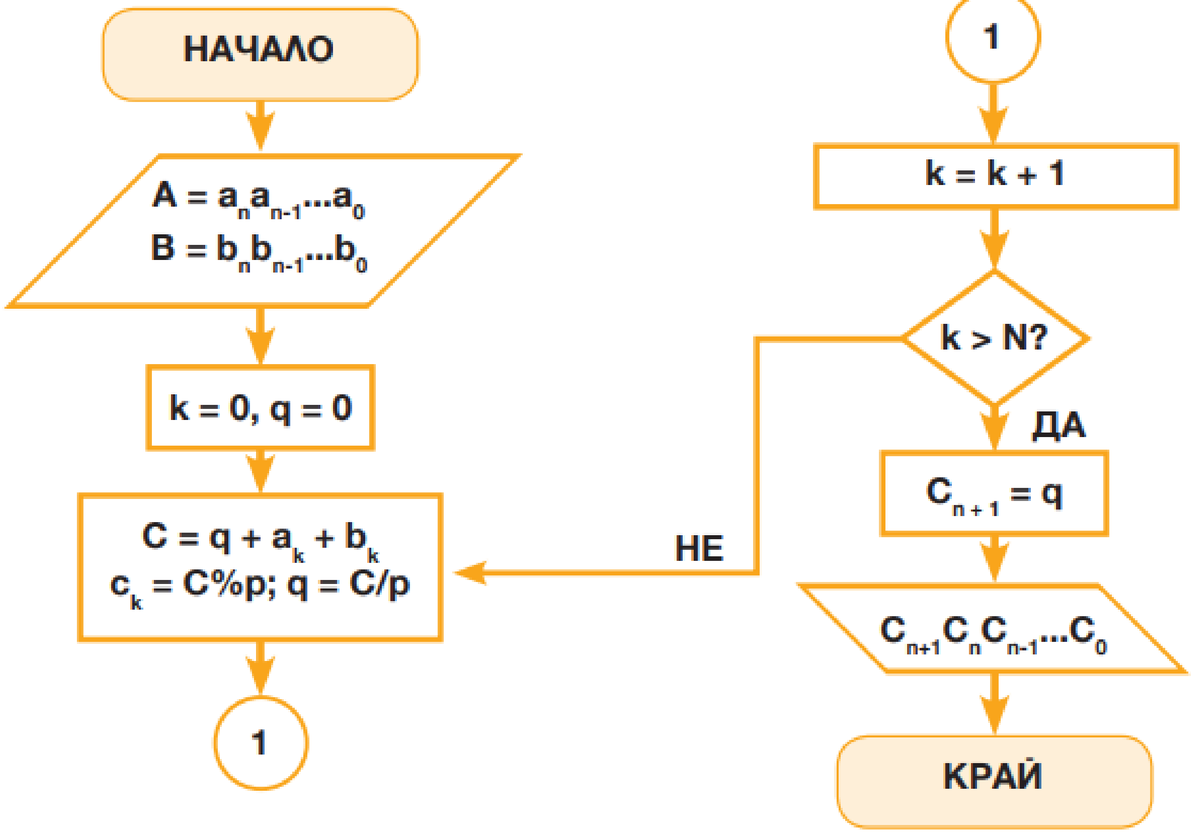
в. В такива блокове описваме безусловните действия, които алгоритъмът трябва да извърши

г. В такива блокове описваме проверката на някакви условия върху данните – когато условието е изпълнено напускаме през стрелката ДА, иначе – през стрелката НЕ

Алгоритми

- *Примери*





Въпроси и задачи

Езици за програмиране

1. Защо съставянето и проверката на програми на машинен език е бавно и трудоемко?
2. С какво асемблерните езици улесняват работата на програмистите?
3. Какво е предназначението на транслаторите?
4. Сравнете работата на компилатора и интерпретатора.

   Отговорите могат да се намерят в лекцията.

5. Възможно ли е за даден език за програмиране да има и компилатор, и интерпретатор?

   Отговор: Не само е възможно, но за някои езици тази възможност е реализирана.

6. По какво езиците за програмиране се различават от естествените езици?

   Упътване: Отговорът е в раздела Описание на алгоритми.

7. Съвременните езици за програмиране са с **почти** еднакви възможности. Защо не съществува единствен, общоприет и универсален език за програмиране?

   Упътване: Отговорът на въпроса се крие в думата почти.

Алгоритми

8. Обяснете изискванията алгоритъмът да е детерминарана и крайна процедура.

9. Защо кулинарните рецепти не са алгоритми?

10. Ако за дадена задача има няколко алгоритъма, какви могат да бъдат критериите, по които да се избере този от тях, който да се използва?

Отговорите могат да се намерят в лекцията.

11. Дайте пример на процедура, която удовлетворява всички изисквания за алгоритъм, освен изискването за:

а. детерминираност – **отговор:** *Въведете N цели числа и изведете някое от тях.*

б. крайност – **отговор:** *Въвеждайте цели числа, докато въведете 0.*

в. масовост – **отговор**: *Намерете и изведете сумата на числата 2 и 3.*

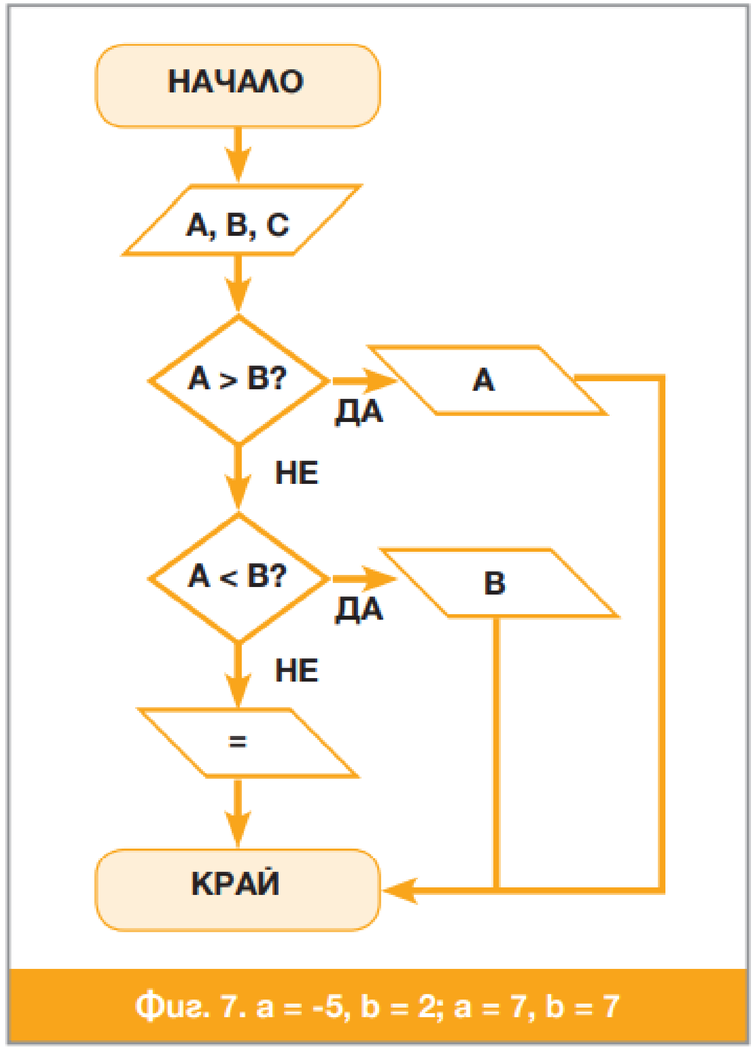
г. наличие на вход – **отговор**: *Намерете и изведете квадратите на числата от 1 до 10*.

12. Защо многократното повтаряне на едни и същи, на пръв поглед, стъпки довежда до получаването на смислени резултати?

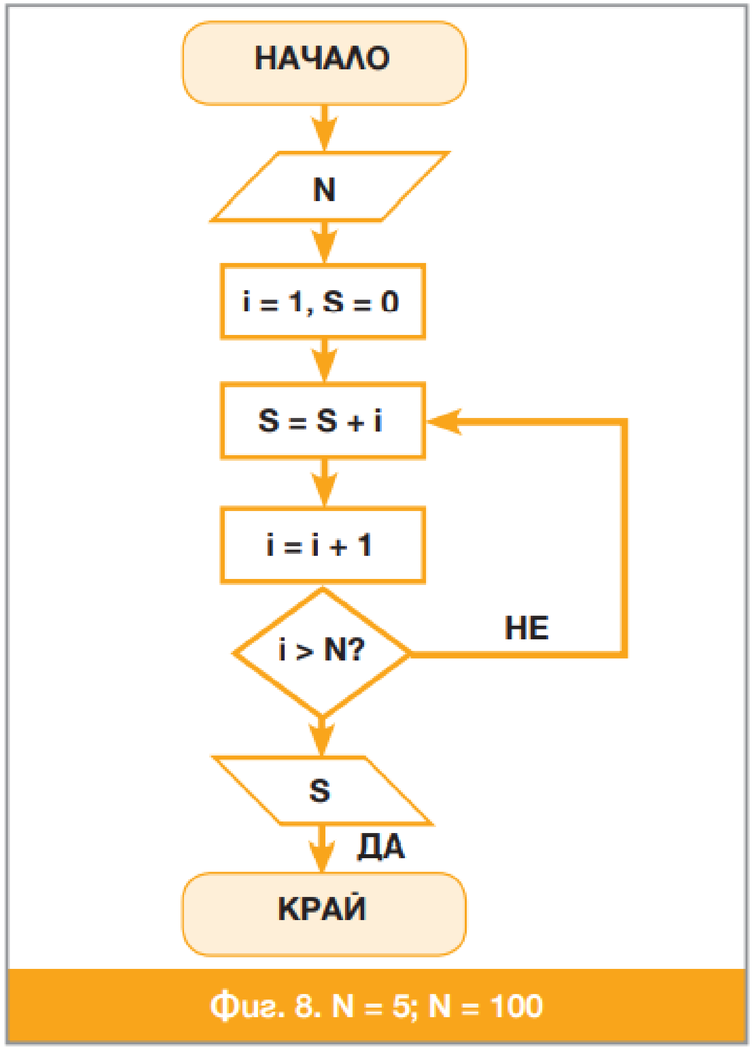
Отговор: Цикличните процедури водят да смислени резултати в резултат на това, че някои от параметрите на цикъла, променят стойностите си преди изпълнението на следващата стъпка.

Алгоритми

6. За всяка от блок-схемите по-долу определете какъв ще бъде изведеният резултат при зададените стойности на входните данни:



Фиг. 7. a = -5, b = 2; a = 7, b = 7



Фиг. 8. N = 5; N = 100

Програмиране на асемблерен език

- 1. Напишете програма на нашия абстрактен асемблерен език, която въвежда указаните в условието стойности, пресмята зададения израз и извежда стойността му:

а. A + B + C;

```
1000 ВЪВЕДИ A
1001 ВЪВЕДИ B
1002 ВЪВЕДИ C
1003 ПРЕМЕСТИ X A
1004 СЪБЕРИ X B
1005 СЪБЕРИ X C
1006 ИЗВЕДИ X
1007 КРАЙ
```

б. A + B.C;

```
1000 ВЪВЕДИ A
1001 ВЪВЕДИ B
1002 ВЪВЕДИ C
1003 ПРЕМЕСТИ T B
1004 УМНОЖИ T C
1005 ПРЕМЕСТИ X A
1006 СЪБЕРИ X T
1007 ИЗВЕДИ X
1008 КРАЙ
```

в. A.B + C.D

```
1000 ВЪВЕДИ A
1001 ВЪВЕДИ B
1002 ВЪВЕДИ C
1003 ВЪВЕДИ D
1004 ПРЕМЕСТИ T C
1005 УМНОЖИ T D
1006 ПРЕМЕСТИ X A
1007 УМНОЖИ X B
1008 СЪБЕРИ X T
1009 ИЗВЕДИ X
1010 КРАЙ
```

Програмиране на асемблерен език

- 2. Напишете програма, която въвежда данните, пресмята верността на условията и извежда 0, когато условието не е изпълнено или 1 – ако е изпълнено:
  а. A + B + C < 0; б. A + B > C; в. A.B ≠ C.D.

```
1000 ВЪВЕДИ A
1001 ВЪВЕДИ B
1002 ВЪВЕДИ C
1003 ПРЕМЕСТИ X A
1004 СЪБЕРИ X B
1005 СЪБЕРИ X C
1006 ПРИ<0 X 1009
1007 ИЗВЕДИ# 0
1008 СТОП
1009 ИЗВЕДИ# 1
1010 КРАЙ
```

```
1000 ВЪВЕДИ A
1001 ВЪВЕДИ B
1002 ВЪВЕДИ C
1003 ПРЕМЕСТИ X A
1004 СЪБЕРИ X B
1005 ИЗВАДИ X C
1006 ПРИ>0 X 1009
1007 ИЗВЕДИ# 1
1008 СТОП
1009 ИЗВЕДИ# 1
1010 КРАЙ
```

```
1000 ВЪВЕДИ A
1001 ВЪВЕДИ B
1002 ВЪВЕДИ C
1003 ВЪВЕДИ D
1004 ПРЕМЕСТИ T C
1005 УМНОЖИ T D
1006 ПРЕМЕСТИ X A
1007 УМНОЖИ X B
1008 ИЗВАДИ X T
1009 ПРИ=0 X 1012
1010 ИЗВЕДИ# 1
1011 СТОП
1012 ИЗВЕДИ# 0
1013 КРАЙ
```

Програмиране на асемблерен език

3. Съставете блок-схема на алгоритъм и напишете фрагмент от програма, който размества стойностите на адреси A и B:

Решение:

```
1100 . . .
1101 ПРЕМЕСТИ T A
1102 ПРЕМЕСТИ A B
1103 ПРЕМЕСТИ B T
1104 …
```

4. Съставете блок-схема на алгоритъм и напишете програма, която размества стойностите на адреси A, B, C така че да са наредени в ненамаляващ ред:

Решение:

```
1000 ВЪВЕДИ A
1001 ВЪВЕДИ B
1002 ВЪВЕДИ C
1003 ПРЕМЕСТИ X A
1004 ИЗВАДИ X B
1005 ПРИ<0 X 1009
1006 ПРЕМЕСТИ T A
1007 ПРЕМЕСТИ A B
1008 ПРЕМЕСТИ B T
1009 ПРЕМЕСТИ X B
1010 ИЗВАДИ X C
1011 ПРИ<0 X 1015
1012 ПРЕМЕСТИ T B
1013 ПРЕМЕСТИ B C
1014 ПРЕМЕСТИ C T
1015 ПРЕМЕСТИ X A
1016 ИЗВАДИ X B
1017 ПРИ<0 X 1021
1018 ПРЕМЕСТИ T A
1019 ПРЕМЕСТИ A B
1020 ПРЕМЕСТИ B T
1021 КРАЙ
```

Програмиране на асемблерен език

5. Съставете блок-схема на алгоритъм и напишете програма която въвежда целите A и B и проверява дали B дели A без остатък. Ако B дели A без остатък, тогава програмата да извежда 1, а в противен случай – 0.

6. Съставете блок-схема и напишете програма, която въвежда целите неотрицателни A и B. Ако някое от въведените числа е 0, програмата извежда -1, а в противен случай извежда решението на уравнението A.x = B като частно и остатък при деление. Ако остатъкът е 0, тогава програмата не бива да го извежда.

7. Съставете блок-схема и напишете програма, която въвежда цяло N и N цели числа и намира и извежда най-малкото от тях.

8. Какво ще промените в програмата от Зад. 7 за до може да извежда и най-голямото от въведените числа?

Изгревът е близо

Благодаря за вниманието!

Готов съм да отговарям на вашите въпроси