**Изпитна тема № 1: Изчисления, линейни, разклонени и циклични алгоритми**

**Дефиниция на основни понятия: програмиране, език за програмиране, среда за разработка (IDE), компилация и интерпретация. Променливи и типове от данни – дефиниция, сравнение на типове и употреба. Пресмятания, аритметични и логически оператори, изрази – видове и разработка. Описание на условни конструкции – пълна/ и кратка форма. Логически изрази и оператори за сравнение – видове и разработка. Оператор за многовариантен избор – описание, сравнение с условен оператор. Циклични оператори – описание и разработка. Видове цикли – сравнение, прилики и разлики. Същност на подпрограми (функции/методи), видове параметри, видове връщана стойност, предимства при употребата на подпрограми. Определяне на реда на изпълнение на фрагментите в кода (program flow). Работа със среда за разработка (IDE) – създаване и зареждане на проект в среда за разработка. Създаване и/или поправка/допълване на вече съществуващи компютърни програми, решаващи изчислителни и алгоритмични задачи със средствата на програмен език.**

**Програмирането е действие, което включва **подреждане, структуриране или съставяне на поредица от хронологични действия за постигане на дадена цел**. **Да програмираме** означава да даваме команди на компютъра какво да прави. Когато командите са няколко една след друга, те се наричат **компютърна програма**. Текстът на компютърните програми се нарича **програмен код** (или **сорс код** или за по-кратко **код**).**

**Езикът за програмиране е изкуствен език (синтаксис за изразяване), предназначен за задаване на команди, които искаме компютърът да прочете, обработи и изпълни. Чрез езиците за програмиране пишем поредици от команди (програми), които задават какво да прави компютъра. Изпълнението на компютърните програми може да се реализира с компилатор или с интерпретатор.**

**Компилаторът превежда кода от програмен език на машинен код, като за всяка от конструкциите (командите) в кода избира подходящ, предварително подготвен фрагмент от машинен код, като междувременно проверява за грешки текста на програмата. Заедно компилираните фрагменти съставят програмата в машинен код, както я очаква микропроцесорът на компютъра. След като е компилирана програмата, тя може да бъде директно изпълнена от микропроцесора в кооперация с операционната система. При компилируемите езици за програмиране компилирането на програмата се извършва задължително преди нейното изпълнение и по време на компилация се откриват синтактичните грешки (грешно зададени команди). С компилатор работят езици като C++, C#, Java, Swift и Go.**

**Някои езици за програмиране не използват компилатор, а се интерпретират директно от специализиран софтуер, наречен "интерпретатор". Интерпретаторът е "програма за изпълняване на програми", написани на някакъв програмен език. Той изпълнява командите на програмата една след друга, като разбира не само от единични команди и поредици от команди, но и от другите езикови конструкции (проверки, повторения, функции и т.н.). Езици като PHP, Python и JavaScript работят с интерпретатор и се изпълняват без да се компилират. Поради липса на предварителна компилация, при интерпретеруемите езици грешките се откриват по време на изпълнение, след като програмата започне да работи, а не предварително.**

**Средата за програмиране (Integrated Development Environment - IDE, интегрирана среда за разработка) е съвкупност от традиционни инструменти за разработване на софтуерни приложения. В средата за разработка пишем код, компилираме и изпълняваме програмите. Средите за разработка интегрират в себе си текстов редактор за писане на кода, език за програмиране, компилатор или интерпретатор и среда за изпълнение за изпълнение на програмите, дебъгер за проследяване на програмата и търсене на грешки, инструменти за дизайн на потребителски интерфейс и други инструменти и добавки. За програмиране на езика C# най-често се ползва средата за разработка Visual Studio, която се разработва и разпространява безплатно от Microsoft**

**Всички данни се записват в компютърната памет (RAM памет) в променливи. Променливите са именувани области от паметта, които пазят данни от определен тип, например число или текст. Всяка една променлива в C# има име, тип и стойност.**

**В програмирането всяка една променлива съхранява определена стойност от даден тип.**

**Типовете данни могат да бъдат например: число, буква, текст (стринг), дата, цвят, картинка, списък и др. Ето няколко примера за типове данни:**

* **тип цяло число: 1, 2, 3, 4, 5, …**
* **тип дробно число: 0.5, 3.14, -1.5, …**
* **тип буква от азбуката (символ): 'a', 'b', 'c', …**
* **тип текст (стринг): "Здрасти", "Hi", "Beer", …**
* **тип ден от седмицата: Monday, Tuesday,**

**Аритметични операции в програмирането са:**

* **Събиране на числа (оператор +)**
* **Изваждане на числа (оператор -)**
* **Умножение на числа (оператор \*)**
* **Деление на числа (оператор /)**

**Оператори за сравнение в програмирането са:**

* **Оператор < (по-малко)**
* **Оператор > (по-голямо)**
* **Оператор <= (по-малко или равно)**
* **Оператор >= (по-голямо или равно)**
* **Оператор == (равно)**
* **Оператор != (различно)**

**В програмирането често проверяваме дадени условия и извършваме различни действия, според резултата от проверката. Това става чрез условният оператор if , който има пълна и кратка форма.**

***Общият вид на оператора* (кратка форма) изглежда по следния начин:**

**if (булев израз)**

**{**

**// тяло на условната конструкция;**

**}**

***Действие на оператора*: Проверява се булевият израз (условие), ако е вярно - се изпълнява тялото на условната конструкция (команда или група от команди)**

***Общият вид на оператора* (пълна форма) изглежда по следния начин:**

**Конструкцията if може да съдържа и else клауза, с която да окажем конкретно действие в случай, че булевият израз (който е зададен в началото if (булев израз)) върне отрицателен резултат (false). Така построена, условната конструкция наричаме if-else и поведението ѝ е следното: ако резултатът от условието е позитивен (true) - извършваме едни действия, a когато е негативен (false) - други. Форматът на конструкцията е:**

**if (булево условие)**

**{**

**// тяло на условната конструкция;**

**}**

**else**

**{**

**// тяло на else-конструкция;**

**}**

**Вложени проверки**

**Доста често програмната логика налага използването на if или if-else конструкции, които се съдържат една в друга. Те биват наричани вложени if или if-else конструкции. Както се подразбира от названието "вложени", това са if или if-else конструкции, които са поставени в други if или else конструкции.**

**if (condition1)**

**{**

**if (condition2)**

**{**

**// тяло;**

**}**

**else**

**{**

**// тяло;**

**}**

**}**

**По-сложни проверки**

* **Логическо "И" (оператор &&) означава няколко условия да са изпълнени едновременно**
* **Логическо "ИЛИ" (оператор ||) означава да е изпълнено поне едно измежду няколко условия**
* **Логическо отрицание (оператор !) означава да не е изпълнено дадено условие**

**Условна конструкция switch-case**

**Конструкцията switch-case работи като поредица if-else блокове. Когато работата на програмата ни зависи от стойността на една променлива, вместо да правим последователни проверки с if-else блокове, можем да използваме условната конструкция switch. Тя се използва за избор измежду списък с възможности. Конструкцията сравнява дадена стойност с определени константи и в зависимост от резултата предприема действие. Променливата, която искаме да сравняваме, поставяме в скобите след оператора switch и се нарича "селектор". Тук типът трябва да е сравним (числа, стрингове). Последователно започва сравняването с всяка една стойност, която се намира след case етикетите. При съвпадение започва изпълнението на кода от съответното място и продължава, докато стигне оператора break. В C#, наличието на break е задължителен за всеки case, който съдържа изпълнение на програмна логика. При липса на съвпадение, се изпълнява default конструкцията, ако такава съществува.**

***Общият вид на оператора:***

**switch (селектор)**

**{**

**case стойност1:**

**конструкция;**

**break;**

**case стойност2:**

**конструкция;**

**break;**

**case стойност3:**

**конструкция;**

**break;**

**…**

**default:**

**конструкция;**

**break;**

**}**

**Цикъл (loop) е основна конструкция в програмирането, която позволява многократно изпълнение на даден фрагмент сорс код. В зависимост от вида на цикъла програмният код в него се повтаря или фиксиран брой пъти, или докато е в сила дадено условие.**

**Цикъл while -цикъл с предусловие(pre-test loop)- общ вид на оператора:**

**while (условие)**

**{**

**тяло на цикъла;**

**}**

**При while цикъла първоначално се изчислява булевият израз(условието) и ако резултатът от него е true, се изпълнява последователността от операции в тялото на цикъла. След това входното условие отново се проверява и ако е истина, отново се изпълнява тялото на цикъла. Всичко това се повтаря отново и отново докато в някакъв момент условният израз върне стойност false. В този момент цикълът приключва своята работа и програмата продължава от следващия ред веднага след тялото на цикъла.**

**Тялото на while цикъл може и да не се изпълни нито веднъж, ако в самото начало е нарушено условието на цикъла. Ако условието на цикъла никога не бъде нарушено, той ще се изпълнява безкрайно.**

**Оператор break- използва се за преждевременно излизане от цикъл, преди той да е завършил изпълнението си по естествения си начин. При срещане на оператора break цикълът се прекратява и изпълнението на програмата продължава от следващия ред веднага след тялото на цикъла. Прекра­тяването на цикъл с оператора break може да стане само от неговото тяло, когато то се изпълнява в поредната итерация на цикъла. Когато break се изпълни кодът след него в тялото на цикъла се прескача и не се изпълнява.**

**Цикъл Do-while - аналогичен е на while цикъла, само че при него проверката на булевото условие се извършва след изпълнението на операциите в цикъла. Този тип цикли се наричат цикли с условие в края (post-test loop). Общ вид на оператора:**

**do**

**{**

**код за изпълнение;**

**}**

**while (израз);**

**Първоначално се изпълнява тялото на цикъла. След това се проверява неговото условие. Ако то е истина, тялото на цикъла се повтаря, а в противен случай цикълът завършва. Тази логика се повтаря докато условието на цикъла бъде нарушено. Тялото на цикъла се повтаря най-малко един път. Ако условието на цикъла постоянно е истина, цикълът никога няма да завърши.**

**Цикъл for - по-сложен от while и do-while циклите, но за сметка на това може да решава по-сложни задачи с по-малко код - общ вид на оператора:**

**for (инициализация; условие; обновяване)**

**{**

**тяло на цикъла;**

**}**

**Той се състои от инициализационна част за брояча (в схемата int i = 0), булево условие (i < 10), израз за обновяване на брояча (i++, може да бъде i-- или например i = i + 3) и тяло на цикъла.**

**Броячът на for цикъла го отличава от останалите видове цикли. Най-често броячът се променя от дадена начална стойност към дадена крайна стойност в нарастващ ред. Броят на итерациите на даден for-цикъл най-често е известен още преди да започне изпълнението му. Един for-цикъл може да има една или няколко водещи променливи, които се движат в нарастващ ред или в намаляващ ред или с някаква стъпка. Възможно е едната водеща променлива да расте, а другата – да намалява.**

**Цикъл foreach (разширен for-цикъл) - служи за обхождане на всички елементи на даден масив, списък или друга колекция от елементи - общ вид на оператора:**

**foreach (variable in collection)**

**{**

**statements;**

**}**

**Вложените цикли представляват конструкция от няколко цикъла, разположени един в друг. Най-вътрешния цикъл се изпълнява най-много пъти, а най-външният – най-малко. Да разгледаме как изглеждат два вложени цикъла:**

|  |
| --- |
| **for (инициализация; проверка; обновяване)**  **{**  **for (инициализация; проверка; обновяване)**  **{**  **код за изпълнение;**  **}**  **…**  **}** |

**Същност на подпрограмите: В някои езици за програмиране подпрограмите могат да се срещнат под наименованията функции (functions) или процедури (procedures). В C#, те се наричат методи (methods)**

**Метод (method) е съставна част от програмата, която решава даден проблем, може да приема параметри и да връща стойност. В методите се извършва цялата обработка на данни, която програмата трябва да направи, за да реши поставената задача. Методите съдържат логиката на програмата и те са мястото, където се извършва реалната работа**

**Защо да използваме методи?- По-добро структуриране и по-добра четимост, Избягване на повторението на код, Преизползване на кода**

**Деклариране, имплементация и извикване на собствен метод:**

***Деклариране* на метод наричаме регистрирането на метода в програмата, за да бъде разпознаван в останалата част от нея.**

***Имплементация* (създаване) на метода, е реалното написване на кода, който решава конкретната задача, която методът решава. Този код се съдържа в самия метод и реализира неговата логика.**

***Извикване* е процесът на стартиране на изпълнението, на вече декларирания и създаден метод, от друго място на програмата, където трябва да се реши проблемът, за който е създаден извикваният метод.**

**Един метод може да съществува само ако е деклариран между отварящата и затварящата скоби на даден клас – "{" и "}". Допълнително изискване е методът, трябва да бъде деклариран извън имплементацията на друг метод**

**Декларация на метод представлява регистриране на метода в нашата програма. То става чрез следната декларация:**

|  |
| --- |
| **[public] [static] <return\_type> <method\_name>([<param\_list>])** |

**Задължителните елементи в декларацията на един метод са:**

**-     Тип на връщаната от метода стойност – <return\_type>.**

**-     Име на метода – <method\_name>.**

**-     Списък с параметри на метода – <param\_list> – може да е празен списък или да съдържа поредица от декларации на параметри.**

**Извикването на метод представлява стартирането на изпълнението на кода, който се намира в тялото на метода. Това става като изпишем името му, последвано от кръглите скоби () и знака ; за край на реда. Ако методът ни изисква входни данни, то те се подават в скобите (), като последователността на фактическите параметри трябва да съвпада с последователността на подадените при декларирането на метода.**

**Методи с параметри-Много често в практиката, за да бъде решен даден проблем, методът, с чиято помощ постигаме това, се нуждае от допълнителна информация, която зависи от задачата му. Именно тази информация представляват параметрите на метода и неговото поведение зависи от тях.**

**Незадължителни параметри- Те позволяват пропускането на параметри при извикването на метода. Декларирането им става чрез осигуряване на стойност по подразбиране в описанието на съответния параметър.**

**Връщане на резултат от метод - За да върне един метод резултат е нужно да внимаваме да напишем очаквания тип на резултата при декларацията на метода на мястото на void.**

**Оператор return - за да получим резултат от метода, се използва операторът return. Той трябва да бъде употребен в тялото на метода и указва на програмата да спре изпълнението му и да върне на извиквача на метода определена стойност, която се определя от израза след въпросния оператор return.**

**Вложени методи (локални функции)- Локалните функции могат да се декларират във всеки един друг метод. Когато C# компилаторът компилира такива функции, те биват превърнати в private методи.**