

Пловдивски университет

„Паисий Хилендарски“

**ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА**

# КАТЕДРА „ОБУЧЕНИЕ ПО МАТЕМАТИКА, ИФОРМАТИКА И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ“

# Дипломна работа

# Тема: Учебно помагало „Въведение в програмирането със C#“ за 8 клас общообразователна подготовка

**Научен ръководител:**

проф. д-р Коста Гъров

**Дипломант:** Дейвид Младенов

**Факултетен номер:** 1601651011

**Специалност:** ИТМОМ

**Съдържание:**

1. Увод
2. Учебна програма по информатика за VIII клас (общообразователна подготовка)
3. Учебно помагало „Въведение в програмирането със C#“
4. Заключение
5. Приложение с решения на всички задачи
6. Увод

България може да се похвали с факта, че е една от първите държави в света, които въвеждат изучаването на информатика в средните училища. През 60-те години в много окръжни градове на страната се създават т.н. математически паралелки. В тях учениците изучават предметите “Програмиране” и “Числени методи”. В часовете по “Програмиране” са застъпени темите: Бройни системи, Математически и логически основи на ЕИМ (Електронно изчислителни машини), Алгоритми и начини за изразяването им, Програмиране на АЛГОЛ и ФОРТРАН. По “Числени методи” се разглеждат: Матрици и детерминанти, Приближено решаване на уравнения и системи линейни уравнения, Приближаване на функции, Симплекс-метод, Транспортна задача. Занятията се водят теоретично, като практическа работа се извършва само с електронни калкулатори. Провеждат се и наблюдения върху работата на ЕИМ в големи изчислителни центрове.

В началото на 70-те години се създават математическите гимназии, в които обучението по информатика се провежда по същия начин, както в математическите паралелки. През 1976 г. в Математическата гимназия в Пловдив се открива първият Учебен електронно-изчислителен център в страната, снабден с ЕИМ ИЗОТ 310. По-късно такива центрове се откриват и в София, Русе и Варна. Те са оборудвани с големи ЕИМ от серията ЕС. Наличието на УЕИЦ позволява съществено да се увеличи делът на практическата дейност на учениците.

В началото на 80-те години се прави опит знания по информатика да се преподават в часовете по математика. През 1981 г. в учебника по алгебра за 9. клас около половината от съдържанието е отделено на информатични знания. Този учебник съдържа раздели за бройни системи, алгоритми, приближени стойности, за електронните калкулатори “ЕЛКА” и за автоматични сметачни машини. Подобен подход – да се интегрират елементи от информатиката в други учебни дисциплини, е възприет в системата от училища на Проблемната група по образование (ПГО), където се осъществява експериментално обучение под ръководството на академик Благовест Сендов.

През 1983 г. се осъществява поредната реформа в нашето средно образование. Според нея в първата степен на обучение (1.­10.) клас се осъществява общото образование на учениците, а във втората степен (11.­12.) клас се овладява професия в т. н. Учебно-професиоални комплекси (УПК). За учениците от математическите гимназии и някои техникуми е въведена подготовка за професията “Оператор-програмист на ЕИМ”. Изучават се следните общотехнически и специални предмети: Информатика, Приложна математика, Операционни системи, Програмиране и алгоритмични езици, Електронно-изчислителна техника, Икономика на производството и стопанското управление, Електротехника и електроника. Независимо от редица слабости в някои от учебните програми, въвеждането на тази професия е голяма стъпка напред при изучаване на знания по информатика в средното училище.

Междувременно в Института по техническа кибернетика и роботика (ИТКР) се създава първият български микрокомпютър “ИМКО-2”. През март 1983 г. в МГ “Акад. К. Попов”, Пловдив се открива първият в страната компютърен кабинет, оборудван с 8-битови компютри “Правец 82”. Следва период на бързо насищане на българските училища с микрокомпютри от този вид.

Така в средата на 80-те години информатиката е навлязла под различни форми в средното училище, но е неравномерно застъпена в различните класове, не обхваща най-съществените за средното образование въпроси, недостатъчна е по обем и с незначителни изключения преподаването е с незадоволително качество. Аналогично е положението с материалната база – използва се предимно амортизирана изчислителна техника и все още малък брой микрокомпютри.

През 1986 г. МНП взима изключително важното решение да въведе отделен учебен предмет “Информатика” в 10. и 11. клас на средното общообразователно училище (СОУ). Учебната програма в значителна степен е преходна. Тя е съобразена със съществуващото положение в страната: широко разпространение на микрокомпютъра “Правец 82” и езика Бейсик.

За пръв път в нашата образование бяха разработени 3 различни учебника по една учебна дисциплина – Информатика, ориентирани към ученици от различните видове училища. Авторски колектив под ръководството на професор П. Бърнев написа учебници, предназначени за т. н. “елитни” училища – езикови и математически гимназии – [20] и [21]. А. Ангелов, К. Гъров и О. Гавраилов създадоха учебници, предназначен за масовото средно училище – [7] и [8]. Колектив под ръководството на проф. Л. Даковски написа учебник, ориентиран към учениците от техникумите и СПТУ.

С бързи темпове продължи оборудването на училищата с микрокомпютри “Правец”. По данни на МНП за периода 1984 – 1989 г. българските училища са оборудвани с около 15 000 компютри.

През 1990 г. се извърши промяна в учебния план на СОУ, при която учебният предмет Информатика се изучава в 9. и 10. клас. Съгласно учебната програма в 9. клас, основните понятия и процеси се демонстрират с програми на езика ИНФО. Този алгоритмичен език е подобен на езика за програмиране Паскал, като командите се изписват на български език. В 10 клас е предвидено изучаването на различни приложения на компютрите в практиката.

През 1994 г. настъпи поредната промяна в учебния план на СОУ, при която Информатика се изучава само в 11. клас с 2 часа седмично. Преподаването се извършва по учебното помагало “Информатика” с автори О. Гавраилов и К. Гъров [23]. За училища, оборудвани с 16 битови компютри в 9. и 10. клас се въвежда нов учебен предмет “Информационни технологии” (ИТ). ИТ, базирани на компютърни системи, формират един от най-бързо развиващите се технологични клонове в съвременния стопански и обществен живот. Добилият популярност термин “информационни технологии” се използва в практиката за отбелязване на много широк спектър от конкретни продукти, технологии, технологични процеси и дейности, както в областта на производството на компютърни, комуникациони и офис-системи, така и в областта на създаването и експлоатацията на софтуерни продукти. В Указанието на Министерството на науката и образованието (МНО) за организиране на обучението по Информационни технологии в българските училища [87], се посочва следното работно определение за ИТ:

Технологии, свързани с разработването и/или използването на програмни продукти и системи, предназначени да автоматизират дейностите по реализиране на основните информационни процеси (събиране, съхраняване, преработка и разпространение на информация) чрез използване на компютри, ще наричаме информационни.

Целта на това определение е прагматично да стесни обхвата, очертае рамките и уточни значението на понятието “информационни технологии” за нуждите на СОУ, като по този начин формира критерии, според които да се определя съдържанието на обучението по ИТ. По учебния предмет ИТ са разработени и утвърдени от МОН следните осем учебни програми:

• Операционни системи с текстов интерфейс;

• Текстообработка;

• Електронни таблици;

• Бази от данни;

• Компютърна графика (проектиране);

• Информационни технологии за математически изследвания;

• Графичен потребителски интерфейс MS Windows;

• “Интернет за начинаещи” и “Интернет за напреднали”.

По време на обучението се изучават три или четири от посочените модули, в зависимост от вида на техниката, профила на училището и желанията на учениците.

При така очертаната картина на училищната информатика в края на XX век можем да посочим някои проблеми, свързани с нея:

• Информатиката като нов учебен предмет трябваше да се “вмести” в учебния план сред вече утвърдени учебни дисциплини. Това естествено наложи намаляване на хорариума на други дисциплини и породи негативно отношение на някои специалисти от МОН към новия предмет. Информатиката започна да променя своето място от едни класове в други и да намалява обема си в учебния план.

• Опитът “да се размиват” знанията по информатика в часовете по математика едва ли може да се приеме като успешен. Достатъчно е да се спомене фактът, че не всички преподаватели по математика имат необходимата квалификация по информатика, за да стане ясно, че реални положителни резултати от този подход не могат да се очакват.

• Наличната компютърна техника в страната по това време като количество и качество не създава предпоставки за всеобщо и единно обучение по информатика.

• Подготовката на учители по информатика не е на необходимото равнище. Бързото развитие на ИТ поставя въпроса и за преквалификацията им през определен период от време.

В началото на ХХI век у нас започна нова реформа в образователната система. Тя бе породена както от настъпилите обществено-политически промени в България, така и от технологичните промени в световен мащаб. Съвременното обществено развитие, преходът от индустриално към постиндустриално общество в световен мащаб налага глобализация на икономиката, висока мобилност на работната сила и изисква нов тип компетентности на личността в социалната и професионалната сфера. Тези процеси рефлектират особено силно в сферата на образованието. Образователната политика на държавата отчита както социално-икономическите характеристики на съвременното общество, така и перспективите за бъдещото му развитие. В този смисъл днес се налага преосмисляне на способността на българското училище да отговори адекватно на новите предизвикателства.

Наши и чужди изследвания от последните години показват, че учебното съдържание по различните предмети в България:

• е с подчертано академичен характер за голяма част от тях;

• е голямо по обем;

• в недостатъчна степен отчита възрастовите особености на децата и учениците;

• е подчинено предимно на изискването за запаметяване и възпроизвеждане на конкретни знания;

• е затворено в отделните предмети, без да позволява достатъчно продуктивни връзки между тях;

• в недостатъчна степен се опира на личния опит на ученика;

• като цяло не е ориентирано към формиране на готовност за реализация в съвременното общество.

През последните години световните тенденции, свързани със съдържанието и организацията на обучението, могат да се обединят в четири групи:

1) Ориентация към разбиране и осмисляне на знанието.

2) Ограничаване ролята на репродуктивното знание.

3) Стимулиране на творческата активност на учениците.

4) Утвърждаване на единни държавни изисквания.

Необходимостта от държавни изисквания (стандарти) за учебно съдържание е призната единодушно. Това се потвърждава и от факта, че в повечето европейски страни през последните години разработването на такива стандарти се реализира като приоритетна задача. По този начин образователните институции се ангажират с конкретна отговорност за качеството на образованието на национално ниво.

През 1999 г. със заповед на Министъра на образованието и науката са сформирани работни групи за изготвяне на държавни образователни изисквания по съответните учебни предмети. Проектите за стандартите бяха обсъждани от учители и преподаватели във ВУЗ. След одобряването им, бяха публикувани на 18.05.2000 г. в Държавен вестник и вече определят насоките на средното образование през периода 2000­2008 година [39].

Централно място в стандартите заемат т. н. ядра на учебно съдържание ­ до 5-6 на брой за даден предмет. В стандартите за 9.­12. клас, където попадат учебните предмети Информатиката и ИТ за всяко ядро се определят по две равнища ­ първо и второ.

Първо равнище: Стандартите за това равнище определят общообразователния минимум по предмета. Това са знанията, уменията и отношенията, които учениците могат и трябва да усвоят в рамките на задължителната подготовка. За предмета Информатика тя е 2 часа седмично в 9. клас, а за ИТ – по 1 час седмично в 9. и 10. клас. Тези стандарти трябва да са постижими за около 80% от учениците.

Второ равнище: Стандартите за това равнище определят съдържанието на профилираната подготовка по съответния предмет. Това са знанията, уменията и отношенията, които учениците могат и трябва да усвоят в допълнителни учебни часове – около 500 в 9.­12. клас.

По предмета Информатика са определени следните ядра на учебно съдържание: Информация и формални модели, Компютърни системи, Операционни системи, Алгоритми и структури от данни, Програмиране.

По Информационни технологии ядрата са: Решаване на проблеми с ИТ, Комуникиране, Контрол и управление на обекти, Моделиране, Интегриране на дейности и продукти в ИТ.

На базата на държавните образователни изисквания бързо бяха съставени учебни програми по учебните предмети Информатика и Информационни технологии [88]. През 2001 г. бяха проведени конкурси за написване на учебници по двата учебни предмета. Пълен списък на одобрените учебници за българските училища е даден в [67].

Поради бързото развитие на информационните и комуникационни технологии и масовото им приложение в практиката от учебната 2006-2007 година ИТ започнаха да се изучават в задължителната подготовка в 5.-7. клас, а в избираемата подготовка от 1.-4. клас. Днес можем да констатираме, че Информатиката и Информационните технологии са утвърдени учебни предмети в българското училище, които се радват на голям интерес от страна на учениците. Изучаването на информатика в училище дава стабилна основа на младите ученици за развитие в областта.

Обект на тази разработка ще бъде съставянето на учебно помагало за упражнение съгласно „Учебна програма по информатика за VIII клас“ (Общообразователна подготовка) и далеч по-модерни технологии за програмиране от споменатите по-горе в лицето на езика C# и .NET Core 3.0 платформата.

C# е съвременен обектно-ориентиран език за програмиране разработен от Microsoft, който се появява пред света през 2000-та година. В днешно време C# е един от най-популярните езици за програмиране и се използва от милиони разработчици по цял свят. Като език от високо ниво C#, се слави с лесния си синтаксис и се препоръчва за начинаещи, които имат интерес в сферата на програмирането и именно за това съм избрал точно него за примерите в тази разработка.

1. **Учебна програма по информатика за VIII клас (общообразователна подготовка)**

**Очаквани резултати от обучението в края на класа**

|  |  |
| --- | --- |
| **Област на компетентност** | **Знания, умения и отношения**  **В резултат на обучението ученикът:** |
| **Информатика** | Описва предмета и ролята на информатиката за моделиране |
| Познава представянето на информация във вид на данни |
| Посочва примери на обекти и явления, при които е практически приложимо използването на средствата на обектно-ориентираното моделиране |
| **Обектно-ориентирано**  **програмиране** | Обяснява основните етапи при създаване и изпълнение на компютърна програма |
| Прилага обектно-ориентиран подход при създаване на несложна компютърна програма |
| Описва основни начини за създаване, изпълнение и тестване на програмен проект в интегрирана среда за разработка с използване на визуални графични средства |
| Използва библиотеки от готови компоненти |
| Спазва добър стил на програмиране |
| **Графичен**  **потребителски**  **интерфейс** | Разбира и използва основни компоненти на среда за визуално програмиране при разработка на софтуер |
| Проектира графичен потребителски интерфейс с визуални средства |
| Избира подходяща графична компонента в съответствие с необходимата функционалност на графичния интерфейс |
| Умее да настройва свойствата на графичните компоненти |
| Програмира подразбиращи се събития за основни компоненти от графичния интерфейс |
| **Алгоритми и структури**  **от данни** | Разбира същността на „тип данни“ |
| Разграничава различни типове данни |
| Определя подходящ тип данни за определена задача |
| Разбира същността на алгоритмите и начини за описанието им |
| Прилага основни управляващи конструкции |
| Структурира данни в едномерен масив |
| Прилага основни алгоритми за намиране на сума, минимален/максимален елемент и средно аритметично |
| Чете и записва данни в текстов файл |
| **Софтуерни**  **приложения** | Използва визуално програмиране за решаване на несложни задачи |
| Създава програми за графично изобразяване на геометрични обекти със стандартни средства в езика |
| Създава програмни приложения с мултимедийни компоненти |
| Представя аргументирано разработено софтуерно приложение пред публика |

**Учебно съдържание**

|  |  |  |
| --- | --- | --- |
| **Теми** | **Компетентности като очаквани резултати от обучението** | **Нови понятия** |
| **1. Основи на информатиката** | | |
| 1.1. Числата и техните представяния | * Разширява и обобщава знанията, свързани с числата и технитепредставяния:   + непозиционни бройни системи;   + същност на позиционните бройни системи;   + формат на числата в десетична, двоична и шестнадесетична бройна система. * Превръща числа от десетична в двоична бройна система и обратно. * Извършва събиране, изваждане и умножение на две числа в двоична бройна система. * Дава примери за използване на двоична и шестнадесетична бройна система. | * непозиционни бройни системи * позиционни бройни системи * експоненциален формат и неговото предназначение * двоична бройна система * шестнадесетична бройна система |
| 1.2. Информационни дейности и процеси | * Обяснява предмета на информатиката и ролята ѝ в съвременното общество. * Изброява и описва основните информационни дейности събиране, съхраняване, преработка и разпространение и общата схема на информационните потоци. * Описва понятието информационен процес и дава примери на информационни процеси, свързани с решаване на житейски задачи. * Различава понятията информация и данни. * Обяснява и илюстрира с примери връзката между информация и данни. * • Обяснява и илюстрира с примери същността на дискретното представяне на информацията за трансформирането ѝ в данни. | * основни информационни дейности * информация, данни, дискретно представяне на информацията |
| 1.3. Алгоритми и езици за програмиране | * Дефинира понятието алгоритъм и описва основните му характеристики (резултатност, крайност, детерминираност, масовост). * Описва и проследява несложни, линейни и разклонени алгоритми с различни средства. * Описва същността, структурата и разновидностите на цикличните алгоритмични конструкции. * Обяснява същността и функционалното предназначение на език за програмиране. * Обяснява същността и предназначението на транслатор (интерпретатор, компилатор). * Представя исторически факти, свързани със създаването и развитието на съвременните езици и среди за програмиране. | * алгоритъм * език за програмиране * транслатор |
| **2. Среда за визуално програмиране** | | |
| 2.1. Интегрирана среда за визуално програмиране | * Отваря проект в интегрирана среда за визуално програмиране. * Редактира дизайна на графичния потребителски интерфейс на приложение в интегрирана среда за програмиране. * Запазва проект на приложение чрез средствата на интегрирана среда за програмиране. * Стартира приложение с графичен потребителски интерфейс чрез средствата на интегрирана среда за програмиране. * Разпознава основни компоненти на интегрирана среда за програмиране – графичен и текстов редактор, панел с контроли, панел за свойства на обект, панел за съобщения, панел за преглед на структурата на приложението. | * интегрирана среда за програмиране * свързващ редактор (linker) * програма за откриване и отстраняване на грешки (debugger) * редактор за проектиране на дизайн на графичен потребителски интерфейс |
| 2.2. Основни етапи на създаване и изпълнение на компютърна програма | * Анализира задача с несложен математически модел. * Създава математически модел за решаване на несложна задача. * Съпоставя математически модел с програмно решение на даден проблем. * Открива основните компоненти на математически модел в демонстрирано програмно решение на даден проблем. * Стартира чрез средствата на интегрирана среда предварително подготвена компютърна програма с графичен потребителски интерфейс. * Тества предварително подготвен несложен проект. * Разпознава видовете грешки при програмиране. * Разчита и прави предположение за естеството на синтактична грешка в даден проект. * Открива и прави предположение за причината за логическа грешка в дадено приложение. * Открива и прави предположение за причината за грешка по време на изпълнение на приложението. | * синтактични грешки в компютърна програма * логически грешки в компютърна програма * грешки по време на изпълнение на програмата |
| 2.3. Проектиране на графичен потребителски интерфейс | * Знае предназначението на основни контейнери и контроли – форма, етикет, текстово поле, бутон, диалогова кутия. * Разпознава основни свойства на графични обекти-контроли – име, състояние, етикет, фон, настройка на шрифт и др. * Проектира несложна форма, съдържаща етикет, текстово поле, бутон. * Настройва основни свойства на форма, етикет, текстово поле и бутон. * Именува обекти-контроли съгласно общоприета конвенция. * Задава функционалност на бутон, свързана с извеждането на статично съобщение в диалогова кутия. | * интерфейсен компонент (контрола) * контейнер на контроли * свойство на обект * метод на обект |
| **3. Програмиране** | | |
| **3.1. Основни типове данни** | | |
| 3.1.1. Тип низ | * Познава правила за именуване на константи и променливи. * Декларира, описва и инициализира променливи и константи от тип низ. * Присвоява стойност на променлива от тип низ. * Въвежда и извежда данни от тип низ в/от текстово поле. * Извежда данни от тип низ в/от етикет. * Извършва конкатенация на низове. * Използва стандартни методи на интерфейсни компоненти за форматиране на текст. | * символ * низ * множество на допустимите данни * множество на допустимите операции * име, тип стойност на променлива * име, тип и стойност на константа * присвояване на стойност * конкатенация |
| 3.1.2. Целочислени типове данни | * Декларира, описва и инициализира променливи и константи от целочислен тип данни. * Използва вградени функции за преобразуване на низ в цяло число и обратното. * Въвежда и извежда данни от целочислен тип. * Използва различни целочислени типове данни. * Познава целочислените аритметични операции и техния приоритет. * Конструира аритметични изрази, съдържащи само целочислени данни, спазвайки синтаксиса и семантиката на конкретния език за програмиране. * Прилага и анализира резултатите от операциите – събиране, изваждане, умножение, деление, цяла част и остатък от целочислено деление. * Реализира модел за решаване на задачи, базиран на целочислени типове данни. | * целочислен тип данни * конвенция за именуване на константи и променливи |
| 3.1.3. Реални типове данни | * Декларира, описва и инициализира променливи и константи от реален тип. * Използва вградени функции за преобразуване на низ в реално число и обратното. * Въвежда и извежда данни от реален тип данни. * Използва различни реални типове данни. * Познава приоритетите на аритметичните операции при реални типове данни. * Конструира аритметични изрази, съдържащ реални типове данни, спазвайки синтаксиса и семантиката на конкретния език за програмиране. * Прилага и анализира резултатите от операциите – събиране, изваждане, умножение, деление. * Реализира модел за решаване на задачи, базиран на реални типове данни. | * реален тип данни |
| 3.1.4. Аритметични изрази и вградени математически функции. Приоритет на операциите | * Оценява числената стойност на аритметичен израз, записан на език за програмиране. * Записва аритметичен израз със средствата на език за програмиране. * Прилага и използва вградени в езика за програмиране математически функции – абсолютна стойност, повдигане на степен, закръгляване, извличане на цялата част на реално число. * Използва приоритет на операциите в аритметични изрази, съдържащи вградени функции. * Създава аритметични изрази, съдържащ различни типове данни, като се съобразява със съвместимостта им. * Описва синтаксис и семантика на оператор за присвояване. * Форматира изхода на реално число. | * аритметичен израз в език за програмиране * вградени математически функции в език за програмиране * съвместимост на типове данни * форматиран изход |
| **3.2 Създаване на компютърна програма за решаване на конкретна задача** | | |
|  | * Знае основните етапи при създаване на компютърна програма. * Анализира и проектира решението на конкретна задача. * Създава математическия модел за решаване на задачата. * Разработва алгоритъм за решаване на задачата. * Определя входно-изходни данни и техните типове. * Структурира и разработва графичен интерфейс, като използва обекти и декларира променливи. * Създава и описва програмния код. * Стартира, тества и валидира готовия проект. * Открива синтактични и логически грешки в програмата. * Отстранява синтактични и логически грешки при програмиране. * Спазва изисквания за оформяне на програмния код, включващи подравняване, коментари, именуване на програмните единици. | * коментари * оформяне на програмния код |
| **3.3. Програмни конструкции за реализация на разклонен алгоритъм** | | |
| 3.3.1. Булев тип данни | * Обосновава необходимостта от разклоняване на алгоритмичния процес. * Използва константите от булев тип данни. * Дава примери, в които се използва булев тип данни. * Декларира булева променлива. * Присвоява стойност на булева променлива. * Записва на език за програмиране булев израз, съдържащ операция за сравнение. * Изписва синтактично правилно на езика за програмиране основните логически операции – логическо отрицание, дизюнкция, конюнкция. * Познава приоритета на логическите операции. * Пресмята без използване на компютър стойността на булев израз. * Съставя със средствата на език за програмиране сложен булев израз, отговарящ на дадена логическа ситуация. | * булев тип данни * булеви константи – false, true * логически операции * приоритет на логическите операции * булев израз |
| 3.3.2. Условен оператор | * Описва синтаксиса и семантиката на кратка и пълна форма на условен оператор. * Описва разклонен алгоритъм с помощта на условен оператор. * Използва условен оператор за проверка на коректността на входните данни за програма. * Използва условен оператор за обработка на свойства на радиобутон и поле за отметка. | * условен оператор * съставен оператор |
| 3.3.3. Вложени условни оператори | * Обяснява семантиката на вложени условни оператори в кратка и пълна форма. * Проиграва изпълнението на фрагмент на програма, съдържаща вложени условни оператори. * Записва синтактично и логически правилно вложени условни оператори. * Създава модел и алгоритъм за решаване на задача чрез използване на вложени условни конструкции. * Реализира модел за решаване на задача чрез използване на вложени условни оператори. * Заменя вложен условен оператор с единичен и обратно. | * вложен условен оператор |
| **3.4. Програмни конструкции за реализация на циклични алгоритми** | | |
| 3.4.1. Циклични алгоритми | * Посочва елементите на циклична конструкция – инициализация, тяло на цикъла и условие на цикъла. * Оценява необходимостта от използването на алгоритми с циклични конструкции с предусловие и постусловие. * Записва синтактично и логически правилно оператори за цикъл с предусловие, постусловие и управлявани от брояч. * Прилага алгоритми с циклични конструкции за проверка на входни данни. * Оценява необходимостта от използване на алгоритми с циклични конструкции с условие или управлявани от брояч. * Преобразува програмен код, съдържащ циклична конструкция, управлявана от брояч в циклична конструкция или управлявана от условие. * Открива синтактични и логически грешки в програмния код на алгоритми с циклична конструкция. * Прилага циклични алгоритми за управление на графичен потребителски интерфейс. | * структура на циклични алгоритмични конструкции * оператор за цикъл с предусловие * оператор за цикъл с постусловие * оператор за цикъл, управляван от брояч * списъчно поле |
| 3.4.2. Приложение на условни и циклични конструкции | * Прилага циклични алгоритми за изчертаване на графични примитиви. * Използва циклични алгоритми за въвеждане и извеждане на данни от файл. * Прилага програмни конструкции за реализация на алгоритми за намиране на сума, минимален/максимален елемент, средно аритметично и др. в редици от числа, въвеждани от потребителския интерфейс/клавиатурата. | * графичен примитиви * кутия за изображения * текстов файл |
| **3.5. Тестване и верификация на програма** | | |
|  | * Обяснява и разграничава понятията тестване и верификация. * Дефинира тестови данни. * Дефинира очаквани резултати от тестването при определени входни данни. * Използва инструмент за откриване и отстраняване на грешки (debugger). * Прилага процедури за тестване и верификация на вече създадени програми. | * тестване * верификация * тестови данни |
| **3.6. Съставни типове данни. Едномерен масив** | | |
| 3.6.1. Едномерен масив | * Разбира необходимостта от използване на масиви. * Идентифицира елементите на масив. * Разпознава индекс и стойност на елемент на масив. * Дефинира масив със средствата на език за програмиране. * Създава и инициализира масив със средствата на език за програмиране. * Осъществява достъп до елемент на масив. * Обхожда, въвежда и извежда стойностите на елементите на масив. * Използва списъчно поле за извеждане на стойностите на елементите на масив. | * логическо описание на масив * базов тип на масив * индекс и стойност на елемент от масив |
| 3.6.2. Основни алгоритми за работа с едномерен масив | * Пресмята сбор и произведение на стойностите на елементите на едномерен масив. * Търси елемент от масива с максимална и минимална стойност. * Търси елементи от масива, отговарящи на дадено условие. | * последователно търсене |
| **4. Създаване на софтуерен проект** | | |
|  | * Описва етапите при реализиране на софтуерен проект. * Извършва проучване и анализ на решения за даден групов проект. * Създава модел за решаване на проблема, поставен в заданието на проекта. * Проектира графичен потребителски интерфейс. * Създава програмен код за реализация на модела. * Създава тестови примери с входни данни и очаквани резултати. * Изготвя документация за софтуерния проект. * Презентира и защитава готовия софтуерен проект. |  |

Основната цел на настоящата дипломна работа е да се състави учебно помагало за преподаване на програмиране по представената учебна програма. Помагалото може да бъде ползвано, както от учители, така и от ученици.

1. **Учебно помагало „Въведение в програмирането със C#“**
   * + 1. **Задачите в обучението по програмиране**

В тази глава ще срещнете креативни авторски задачи от различно естество, които са пряко свързани с упражнение върху материала обхващащ учебният план за 8 клас общообразователна подготовка. Задачите са направени така, че да се покаже практическото приложение на изучаваният материал, а не само прости примери чрез, които учениците да не могат да разберат как да прилагат изученото. Всички задачи са приложими за различни езици за програмиране, но в тази разработка ще работим изцяло в контекста на езика C# и .NET Core 3.0 платформата, като към момента на разработка това е най-новата и стабилна версия на платформата. За интегрирана среда за разработка на софтуер (IDE – Integrated Development Environment) ще използваме Visual Studio 2019, което е достъпно да бъде свалено от сайта на Microsoft, напълно безплатно (Visual Studio 2019 Community).

* + - 1. **Бройни системи**

Бройната система представлява символен метод за представяне на числата посредством ограничен брой символи, наречени цифри. Съществуват два вида бройни системи – непозиционни и позиционни.

Непозиционна бройна система наричаме тази бройна система, чийто стойности на всяка цифра не зависят от позицията им. Примери за такива бройни системи са гръцката и римската:

|  |  |
| --- | --- |
| **Гръцка цифра** | **Десетична стойност** |
| Ι | 1 |
| Г | 5 |
| Δ | 10 |
| Η | 100 |
| Χ | 1000 |
| Μ | 10000 |

|  |  |
| --- | --- |
| **Римска цифра** | **Десетична стойност** |
| I | 1 |
| V | 5 |
| X | 10 |
| L | 50 |
| C | 100 |
| D | 500 |
| M | 1000 |

Позиционна бройна система наричаме тази бройна система, чийто стойности на всяка цифра зависят от позицията им. Това означава, че стойността на цифрата в числото не е строго определена и зависи от това на коя позиция се намира съответната цифра в дадено число. Примери за такива бройни системи са двоичната, осмичната, десетичната и шестнадесетичната:

|  |  |  |  |
| --- | --- | --- | --- |
| **Двоична** | **Осмична** | **Десетична** | **Шестнадесетична** |
| 0000 | 0 | 0 | 0 |
| 0001 | 1 | 1 | 1 |
| 0010 | 2 | 2 | 2 |
| 0011 | 3 | 3 | 3 |
| 0100 | 4 | 4 | 4 |
| 0101 | 5 | 5 | 5 |
| 0110 | 6 | 6 | 6 |
| 0111 | 7 | 7 | 7 |
| 1000 | 10 | 8 | 8 |
| 1001 | 11 | 9 | 9 |
| 1010 | 12 | 10 | A |
| 1011 | 13 | 11 | B |
| 1100 | 14 | 12 | C |
| 1101 | 15 | 13 | D |
| 1110 | 16 | 14 | E |
| 1111 | 17 | 15 | F |

Как се образуват числата от десетичната бройна система? За нашият пример ще използваме числото 27865, представяме всяка една цифра образуваща числото като започнем от ляво на дясно по следната формула:

27865 съдържа 5 символа, така че решението на задачата изглежда по следния начин:

27865 =

За разлика от десетичната бройна система, която е лесно разбираема от човека двоичната бройна система е по-сложно, но същевременно разбираема за изчислителната машина (компютър). За представяне на число в двоична бройна система се използват само числата 0 и 1. Прието е, когато едно число се записва в бройна система, различна от десетичната, във вид на индекс в долната му част да се отразява, коя бройна система е използвана за представянето му. Например със записа 1110(2) означаваме число в двоична бройна система. Важно е да знаем, че ако едно двоично число завършва на 0, то е четно, а ако завършва на 1, то е нечетно.

Преобразуването на число от двоична в десетична бройна система става, когато всяка една двоична цифра се умножава по 2 на степен, позицията, на която се намира:

11001(2) = 124 + 123 + 022 + 021 + 120 = 16(10) + 8(10) + 1(10) = 25(10)

От това следва, че 11001(2) = 25(10)

При преминаване от десетична в двоична бройна система, се извършва преобразуване на десетичното число в двоично. За целите на преобразуването се извършва делене на две с остатък. Така се получават частно и остатък, който се отделя.

148 : 2 = 74 остатък 0;

74 : 2 = 37 остатък 0;

37 : 2 = 18 остатък 1;

18 : 2 = 9 остатък 0;

9 : 2 = 4 остатък 1;

4 : 2 = 2 остатък 0;

2 : 2 = 1 остатък 0;

1 : 2 = 0 остатък 1;

След като вече семе извършили деленето, записваме стойностите на остатъците в ред, обратен на тяхното получаване, както следва:

148(10) = 10010100 (2)

Аритметични правила за събиране, изваждане и умножение на двоични числа:

|  |  |  |
| --- | --- | --- |
| 0 + 0 = 0 | 0 – 0 = 0 | 0 x 0 = 0 |
| 1 + 0 = 1 | 1 – 0 = 1 | 1 x 0 = 0 |
| 0 + 1 = 1 | 1 – 1 = 0 | 0 x 1 = 0 |
| 1 + 1 = 10 | 10 – 1 = 1 | 1 x 1 = 1 |

* + - 1. **Интегрирана среда за разработка (IDE)**

Интегрираната среда за разработка представлява софтуерно приложение, което разполага с пълния набор от инструменти нужни на програмиста да създаде собствено софтуерно приложение. То се състои от няколко основни компонента: редактор на код, компилатор/интерпретатор, дебъгер, система за контрол на версиите. В контекста на тази разработка интегрираната среда за разработка, която ще използваме е Visual Studio 2019. След като сте изтеглили и инсталирали Visual Studio 2019, може да преминем към запознаване с неговият интерфейс.

A screenshot of a cell phone

Description automatically generated

След отваряне на Visual Studio 2019, попадаме на началния екран където има три основни секции. В секцията номерирана с номер 1 на снимката, виждаме последно отваряните проекти, в секция номер 2 са някои основни преки пътища, които за момента няма да използваме. Секция номер 3 е желаната от нас натискаме върху надписа. Попадаме в един празен екран, забелязваме че интегрираната среда за разработка наподобява много на обикновена програма тоест има менюта и бутони. Основни секции на интегрираната среда за разработка:

1. стандартно навигационно меню (navigation menu)
2. често използвани функции на средата и преки пътища (shortcuts)
3. файловете на решението (solution explorer)
4. списък с компилационни грешки и предупреждения (errors and warnings)
5. текстови редактор на кода (text editor)

A screenshot of a computer

Description automatically generated

* + - 1. **Създаване на решение (Solution)**

A screenshot of a cell phone

Description automatically generated

След като сме в интегрираната среда за разработка, от навигацията горе в ляво избираме File 🡪 New 🡪 Project.

В отворения прозорец има търсачка в нея изписваме „Blank“ и средата ни показва предложение за „Blank Solution“, селектираме го и натискаме бутона „Next“.

A screenshot of a video game

Description automatically generated

В първата секция слагаме говорещо за решението име за целта на демото го именуваме „MyFirstSolution“. От втората секция избираме местоположението на решението. След като сме готови натискаме бутона „Create“

A screenshot of a video game

Description automatically generated

В „Solution Explorer“ може да видим, че нашето празно решение е създадено успешно.

A black and red text

Description automatically generated

* + - 1. Първа програма

В това упражнение ще се запознаем с това как да създадем нашата първа програма, чийто цел ще бъде изписването на текст в конзолата. Ще разберем още как се стартира тя и някои важни концепции относно именуването и.

След като вече имаме създадено решение, а именно решението от предишното упражнение, което беше именувано „MyFirstSolution“ вече може да добавим първата ни програма към него.

A screenshot of a computer screen

Description automatically generated

1 – натискаме десен бутон на мишката върху решението

2 – избираме от менюто Add

3 – натискаме върху New Project

4 – избираме Console App (.NET Core)

5 – натискаме Next

A screenshot of a cell phone

Description automatically generated

6 – на полето „Project name“ трябва да обърнем малко повече внимание. Име специална конвенция за именуване на нашите проекти, която гласи следното, проект или решение винаги се именуват с описателни имена, като първата буква винаги е главна, ако името съдържа няколко думи никога не се слагат паузи, вместо паузи изписваме всяка следваща дума долепена до предходната като първата и буква трябва да е главна.

Примери:

* грешен пример – „Hello world program exercise“
* верен пример – „HelloWorldProgramExercise“

7 – Натискаме бутона Create

Важно е да се споменем, че когато добавяме програма във вече създадено решение, както е в случая, не променяме местоположението му (Location) за да бъде създадена програмата вътре във вече създаденото решение!

A screenshot of a video game

Description automatically generated

8 – както вече знаем това е решението

9 – програмата, която създадохме

10 – файла, в който ще въвеждаме кода на програмата ни

A picture containing clock

Description automatically generated

A screenshot of a cell phone

Description automatically generated

11 – отваряща и затваряща скоба на namespace HelloWorld, което означава, че това е обхвата (scope) на този namespace или казано по-просто, обхвата на програмата, която създадохме с име „HelloWorld“, който отговаря на секцията от 9-та стъпка

12 – отваряща и затваряща скоба на class Program, също както на предходната стъпка двете скоби дефинират обхвата на класа Program, който съответства на секцията от стъпка 10

13 – отваряща и затваряща скоба на метода Main, както при предходните два примера скобите и тук дефинират обхвата на метода Main, за методи ще говорим по-късно в тази разработка, за момента е нужно да знаем, че всичкия код, който искаме да бъде изпълнен трябва да се намира между отварящата и затварящата скоба на Main метода, защото той е началната точка при стартиране на конзолно приложение

14 – секция за писане на логически код

Нека се фокусираме върху последната секция и разгледаме по-подробно кода, който се състои в случая от един ред, но същевременно ще изпълни заданието ни. Ако разбираме Английски език, командите които са изписани ще ни се сторят доста описателни и естествени. Буквално казваме на компютъра „Конзола, напиши ред, който да съдържа следният текст 🡪 „Hello World!““. До тук сме се справили с задачата, но никъде не виждаме нито конзола, нито текст, това е защото нашата програма все още не е стартирана, за да я стартираме трябва да натиснем клавишната комбинация Ctrl + F5. След натискане на тази клавишна комбинацията, програмата се компилира тоест, проверява се за грешки, предупреждения или други проблеми, които могат да възпрепятстват нейното изпълнение и се стартира след това. Желаният резултат от изпълнението и трябва да изглежда по следния начин:

A screenshot of a computer

Description automatically generated

Може да се поздравим, понеже програмата изпълни исканията ни напълно, нямаше грешки или предупреждения. Ако за момента това Ви се струва малко, ако изпълните всички задания до края на курса, ще се научите да правите далеч по-сложни и полезни приложения. Нека обобщим наученото с решението на няколко задачи.

Когато имаме повече от един проект в нашето решение и искаме при натискане на комбинацията Ctrl + F5 да се стартира проекта, по който работим в момента трябва да направим следната настройка:

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

1 – десен бутон върху решението

2 – избираме Set Startup Project

3 – отбелязваме Current selection и натискаме OK

**Задача 1:** Създайте решение (Solution) в което ще събирате всички задачи от обучението в тази разработка. Именувайте го по следния начин „IntroToProgrammingCSharpTasks“.

**Задача 2:** **TheHiProgram**

Създайте конзолно приложение, което изписва следния текст „Hello! I am <вашето име>“.

Желано поведение на програмата:

A close up of a logo

Description automatically generated

* + - 1. Примитивни типове данни и променливи

Всички модерни програми в днешно време съдържат променливи в кода си. Най-просто казано променливата (variable) представлява резервирано място в оперативната памет на компютъра, където ние можем да запазваме данни от различни видове (числа, символи, текст и т.н). От името им може да разберем, че тази стойност може да бъде променяна по време на изпълнение на програмата.

Типовете данни представляват диапазони от стойности, които имат еднакви характеристики. Съществуват няколко групи примитивни типове данни в C#, те се наричат примитивни защото са част от езика:

* Целочислени – sbyte, byte, short, ushort, uint, int, long, ulong
* Реални с плаваща запетая – float, double
* Реални с десетична точност – decimal
* Булев – bool
* Символен – char
* Символен низ – string
* Обект – object

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип данни** | **Стойност по подразбиране** | **Минимална стойност** | **Максимална стойност** |
| sbyte | 0 | -128 | 127 |
| byte | 0 | 0 | 255 |
| short | 0 | -32768 | 32767 |
| ushort | 0 | 0 | 65535 |
| int | 0 | -2147483648 | 2147483647 |
| uint | 0u | 0 | 4294967295 |
| long | 0L | -9223372036854775808 | 9223372036854775807 |
| ulong | 0u | 0 | 18446744073709551615 |
| float | 0.0f |  |  |
| double | 0.0d |  |  |
| decimal | 0.0m |  |  |
| bool | false | Има само две стойности true и false | |
| char | ‘\u0000’ | ‘\u0000’ | ‘\uffff’ |
| string | null | - | - |
| object | null | - | - |

Декларирането на променлива става по указаният начин, за целта на примера ще деклариране няколко променливи от различен тип:

A screen shot of a computer

Description automatically generated

В числата от едно до пет на схемата, са показани стъпките за деклариране на променлива:

1 – тип данни

2 – говорещо за променливата име

3 – знак равно, който означава присвояване на стойност

4 – началната стойност, която искаме да притежава нашата променлива

5 – знак точка и запетая, който означава край на декларацията

В останалите числа на схемата ще се запознаем с някои особености и конвенции:

6 – променливите в C# винаги се именуват с малка буква, ако името за да бъде достатъчно описателно е нужно да съдържа повече от една дума, то тогава всяка следваща дума се изписва с главна буква. В имена на променливи не се допуска слагане на паузи, служебни думи от езика или числа!

7 – двойни кавички използваме при декларация на символен низ

8 – единични кавички използване при декларация на един символ

**Задача 3: NumbersInDifferentNumeralSystems**

Създайте конзолно приложение, което пресмята стойността на числото 555 в двоична, осмична и шестнадесетична бройна система и изписва резултатите в конзолата.

Желано поведение на програмата:

A screenshot of a cell phone

Description automatically generated

* + - 1. **Оператори**

Операторите позволяват обработката на примитивни типове данни и обекти. Те представляват специални символи и се делят на няколко основни вида:

* аритметични
* логически
* побитови
* за сравнение
* за присвояване
* за преобразуване на типовете

|  |  |
| --- | --- |
| **Категория** | **Оператори** |
| аритметични | -, +, \*, /, %, ++, -- |
| логически | &&, ||, !, ^ |
| побитови | &, |, ^, ~, <<, >> |
| за сравнение | **==, !=, >, <, >=, <=** |
| за присвояване | =, +=, -=, \*=, /=, %=, &=, |=, ^=, <<=, >>= |
| съединяване на символни низове | + |
| за работа с типове | (type), as, is, typeof, sizeof |
| други | ., new, (), [], ?:, ?? |

Също както в математиката и в програмирането има приоритет на операциите, в таблицата долу са показани операциите с най-висок приоритет към операциите с най-нисък.

|  |  |
| --- | --- |
| **Приоритет** | **Оператори** |
| най-висок  най-нисък | ++, --, new, (type), typeof, sizeof |
| ++, --, +, -, !, ~ |
| \*, /, % |
| + (за свързване на низове) |
| +, - |
| <<, >> |
| <, >, <=, >=, is, as |
| ==, != |
| &, ^, | |
| && |
| || |
| ?:, ?? |
| =, \*=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |= |

Логическите оператори работят с булеви стойности (true и false).

|  |  |
| --- | --- |
| **Оператор** | **Значение** |
| && | логическо и |
| || | логическо или |
| ! | логическо отрицание |
| ^ | изключващо или |

* + - 1. Вход и изход от конзолата

До момента ни се е налагало да разпечатваме резултатите от задачите в конзолата. В тази тема ще се упражним, освен да разпечатваме нещо на нея, а и как да четем информация от нея. По този начин нашите програми, ще могат да водят така наречения „диалог“ с потребителя.

|  |  |
| --- | --- |
| **Команда** | **Действие** |
| Console.Write(); | Изписва подадения текст в конзолата, но не преминава на нов ред |
| Console.WriteLine(); | Изписва подадения текст в конзолата, като преминава на нов ред |
| Console.Read(); | Прочита следващия натиснат символ |
| Console.ReadLine(); | Прочита целия ред от символи |
| Console.ReadKey(); | Прочита кой бутон е бил натиснат |

**Задача 4: AgeAfter10Years**

Създайте конзолно приложение, което пресмята възрастта Ви след десет години и изписва резултатите в конзолата. Вашата програма трябва да извършва „диалог“ с потребителя, като го пита за неговата възраст и той трябва да я въведе. Програмата трябва да извършва правилни сметки независимо от подаденото число!

|  |  |  |
| --- | --- | --- |
|  | **Вход** | **Изход** |
| 22  -------------------  50 | You will be 32 years old after 10 years!  ----------------------------------------------------------------------------------------------------  You will be 60 years old after 10 years! |

Желано поведение на програмата:

A screenshot of a cell phone

Description automatically generated

**Задача 5: TheUniqueNumber**

Създайте конзолно приложение, което пресмята вашето уникално число. Формулата за пресмятане на вашето уникално число е:

**уникално число = (възрастта Ви след 10 години \* годината Ви на раждане) / π \* номера на месеца Ви на раждане**

Резултата трябва да бъде закръглен до 4 знак след десетичната запетая. Резултата трябва да бъде разпечатан в конзолата.

Подсказка: Проучете в интернет относно приложението на класа Math в C#.

|  |  |  |
| --- | --- | --- |
|  | **Вход** | **Изход** |
| 22  ↓  1998  ↓  3 | Your unique number is 61054.3826826! |

Желано поведение на програмата:

A screenshot of a cell phone

Description automatically generated

**Задача 6: TravelingToTheSunWithSoundSpeed**

Създайте конзолно приложение, което пресмята нужното време за да стигнете до слънцето ако пътувате със скоростта на звука. Вашата програма трябва да отпечатва резултата в конзолата.

Разстояние до слънцето: 152080000 км

Скоростта на светлината: 1234.8 км/ч

Желано поведение на програмата:

A screenshot of a cell phone

Description automatically generated

* + - 1. **Условни конструкции**

Съществуват три основни вида условни конструкции в C#, а именно if, if-else и switch case. Условната конструкция от типа if изглежда по следния начин:

if (булев израз)

{

тяло на условната конструкция;

}

Оформлението и включва, if-клауза, булев израз и тяло на условната конструкция.

Условната конструкция от типа if-else изглежда по следния начин:

int x = 2;

if (x > 3)

{

             Console.WriteLine(**"x е по-голямо от 3"**);

 }

       else

       {

             Console.WriteLine(**"x не е по-голямо от 3"**);

       }

Оформлението и също включва, if-клауза, булев израз и тяло на условната конструкция, но ако условието в if-клаузата не бъде изпълнено, ще се изпълни else-клаузата.

Важно е да се знае, че if и if-else условни конструкции могат да се влагат едни в други или казано с прости думи в тялото на една if-условна конструкция, може да имаме if-else-условна конструкция например.

Условната конструкция от типа switch case изглежда по следния начин:

switch (селектор)

{

          case целочислена-стойност-1: конструкция; break;

           case целочислена-стойност-2: конструкция; break;

           case целочислена-стойност-3: конструкция; break;

          default: конструкция; break;

}

Оформлението и включва, switch-оператора, селектор, който е израз, връщащ като резултат някаква стойност, която може да бъде сравнявана, например число или string. Ако се открие съвпадение с някой case етикет, се изпълнява съответната конструкция. Ако не се открие съвпадение, се изпълнява default конструкцията.

**Задача 7: OddOrEven**

Създайте конзолно приложение, което при зададен вход в конзолата, който е число, програмата отпечатва дали въведеното число е четно или нечетно. Препоръчително е вашата програма да извършва адекватен „диалог“ с потребителя, както е показано в примера долу. Вашата програма трябва да работи за всички подадени числа, независимо от големината им!

|  |  |  |
| --- | --- | --- |
|  | **Вход** | **Изход** |
| 7  -------------------  2 | The number is odd!  ----------------------------------------------------------------------------------------------------  The number is even! |

Желано поведение на програмата:

A screenshot of a computer

Description automatically generated

**Задача 8: NumberComparison**

Създайте конзолно приложение, което получава като вход от потребителя две числа. След това изписва в конзолата дали първото число е по-голямо, по-малко или равно на второто. Вашата програма трябва да работи за всички подадени числа, независимо от големината им!

|  |  |  |
| --- | --- | --- |
|  | **Вход** | **Изход** |
| 222 & 45  -------------------  55 & 55  -------------------  45 & 222 | 222 is greater than 45!  ----------------------------------------------------------------------------------------------------  55 is equal to 55!  ----------------------------------------------------------------------------------------------------  45 is less than 222! |

Желано поведение на програмата:

A screenshot of a computer

Description automatically generated

**Задача 9: MyNumberDiapason**

Създайте конзолно приложение, което получава като вход от потребителя число. След това отпечатва в конзолата в кой от трите диапазона попада. Трите диапазона са следните:

1. въведеното число е по-малко от 1000
2. въведеното число е между 1000 и 10000
3. въведеното число е по-голямо от 10000

|  |  |  |
| --- | --- | --- |
|  | **Вход** | **Изход** |
| 888  -------------------  5000  -------------------  15000 | The number 888 is less than 1000!  ----------------------------------------------------------------------------------------------------  The number 5000 is between 1000 and 10000!  ----------------------------------------------------------------------------------------------------  The number 88 is greater than 10000! |

Желано поведение на програмата:

A screenshot of a cell phone

Description automatically generated

**Задача 10: CalculateMyWeightInSolarSystem**

Създайте конзолно приложение, което получава като вход от потребителя число, което отговаря на номера на предефиниран списък с планетите от слънчевата система. След въведен номер на планетата вашата програма трябва да пита за теглото на потребителя в килограми. Когато потребителя въведе цялата информация вашата програма трябва да пресметне теглото на човека за избраната от него планета от слънчевата система.

Важно е да имате в предвид, че теглото се определя по формулата:

теглото в килограми в на избраната планета = гравитацията на съответната планета \* вашето тегло

Гравитацията на планетите в таблицата са закръглени до втория знак след десетичната запетая!

|  |  |
| --- | --- |
| **Планета** | **Гравитация** |
| Меркурий | 0,38 |
| Венера | 0,91 |
| Земя | 1 |
| Марс | 0,38 |
| Юпитер | 2,36 |
| Сатурн | 0,92 |
| Уран | 0,89 |
| Нептун | 1,12 |
| Плутон | 0,07 |

Желано поведение на програмата:

A screenshot of a computer

Description automatically generated

* + - 1. Цикли

В програмирането често се налага многократно изпълнение на дадена последователност от операции. 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, който никога не завършва, се нарича безкраен цикъл (infinite loop). Използването на безкраен цикъл рядко се налага, освен в случаи, когато някъде в тялото на цикъла се използва операторът break, за да бъде прекратено неговото изпълнение преждевременно.

Съществуват два основни вида цикли в C#, цикли с предусловие (условието за изпълнение на цикъла се проверява дали е валидно преди да се изпълни) и цикли със следусловие (условието за изпълнение на цикъла се проверява дали е валидно след първото завъртане).

Конструкция и начин на работа на циклите в C#:

1. while – цикъла ще повтаря докато логическия израз в кръглите скоби не стане със стойност false

while (условие)

{

          желан сорс код за изпълнение;

}

1. do while – цикъла ще се изпълни един път и след това ще провери стойността на логическия израз в кръглите скоби, ако е true ще продължи да се върти докато не стане false, а ако е false ще прекрати изпълнението си

do

{

          желан сорс код за изпълнение;

}

while (израз);

1. for - се състои от инициализационна част за брояча (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

for (инициализация; условие; обновяване)

{

          желан сорс код за изпълнение;

}

1. foreach - служи за обхождане на всички елементи на даден масив, списък или друга колекция от елементи

foreach (променлива in колекция)

{

          желан сорс код за изпълнение;

}

1. вложени цикли - представляват конструкция от няколко цикъла, разпо­ложени един в друг. Най-вътрешния цикъл се изпълнява най-много пъти, а най-външният – най-малко

for (инициализация; условие; обновяване)

{

          for (инициализация; условие; обновяване)

          {

                   желан сорс код за изпълнение;

          }

          …

}

**Задача 11: SimpleDeckOfCards**

Създайте конзолно приложение, което пита потребителя дали иска да види всички карти от едно тесте. Докато потребителя не даде отговор „yes“, не трябва да спирате да задавате въпроса. При получен отговор „yes“, вашата програма трябва да разпечата тестето в конзолата. Под тесте с карти разбираме 52 карти, които са съставени от 13 карти всяка от която има четири бои (купа, каро, спатия, пика). Програмата трябва да разпечатва в конзолата символите на боите.

Желано поведение на програмата:

A screenshot of a computer

Description automatically generated

**Задача 12: BusTicketLuckyNumber**

Компания за транспорт на пътници решава да изненада своите клиенти с лотария. Те имат скромни изисквания:

* всеки пътнически билет има уникален шест цифрен номер
* ако сумата на първите три числа от номера са равни на сумата от вторите три, то вашият билет е щастлив
* пример за щастлив пътнически билет 829955 (8 + 2 + 9 = 9 + 5 + 5)
* всички останали билети се считат за обикновени

Тук идва и вашата задача, направете конзолно приложение, което:

* принтира всички печеливши комбинации в конзолата
* пресмята и принтира колко е броя на всички печеливши билети
* пресмята и принтира колко е броя на всички обикновени билети
* пресмята и принтира колко е броя на всички билети

Желано поведение на програмата:

A screenshot of a computer

Description automatically generated

**Задача 13: MiserPiggy**

Заради своята алчност, един ден трите прасенца решили да пресметнат чие име е по-скъпо. Решили, че ще заменят всяка една буква от имената си със стойността и от ASCII таблицата. Сумата от всички букви заменени със стойностите им от ASCII таблицата е равна на стойността на името. Вашата задача е да създадете програма, която извършва сметките и отпечатва в конзолата в низходящ ред резултатите от калкулацията. Имената на прасенцата са Fiffer Pig, Fiddler Pig, Practical Pig. Направете вашата програма да може да работи и с други имена като ги взима като вход от конзолата.

|  |  |  |
| --- | --- | --- |
|  | **Вход** | **Изход** |
| Fiffer Pig  &  Fiddler Pig  &  Practical Pig | The name-wealthiest pig is Practical Pig (1235) followed by Fiddler Pig (1018) and last is Fiffer Pig (914). |

Желано поведение на програмата:

A screenshot of a cell phone

Description automatically generated

* + - 1. Масиви

Масивите представляват съвкупности от променливи, които наричаме елементи. Елементите на масивите в C# са номерирани с числата 0, 1, 2, ... N-1. Тези номера на елементи се наричат индекси. Броят елементи в даден масив се нарича дължина на масива. Всички елементи на даден масив са от един и същи тип, независимо дали е примитивен или референтен. Това ни помага да представим група от еднородни елементи като подредена свързана последователност и да ги обработваме като едно цяло. Масивите могат да бъдат от различни размерности, като най-често използвани са едномерните и двумерните масиви. Едномерните масиви се наричат още вектори, а двумерните – матрици. В C# масивите имат фиксирана дължина, която се указва при инициализирането им и определя броя на елементите им. След като веднъж е зададена дължината на масив при неговото създаване, след това не е възможно да се променя. Основни начини за декларация на масив:

int[] myArray = new int[6];

int[] myArray = { 1, 2, 3, 4, 5, 6 };

За да бъдат разпечатани стойностите на един масив, то той трябва да бъде обходен чрез цикъл и на всяка една итерация отпечатваме текущия елемент.

**Задача 14: WorkingWithSomeIntegers**

Направете програма, която генерира 100 случайни числа в диапазона [0, 132] и ги разпечатва в конзолата. След като е разпечатала случайните числа, програмата трябва да даде право на потребителя да избере информацията, която му е нужна по формата на команда. Командите за улеснение ще бъдат букви, зад всяка буква стои съответно действието:

1. отпечатва всички числа с четни индекси
2. отпечатва всички нечетни числа, които имат същевременно и нечетни индекси
3. отпечатва всички числа, които се делят на 3 без остатък
4. отпечатва всички числа, които се делят на 7 и имат остатък 1
5. отпечатва всички числа, които се намират в интервала [26, 100]
6. отпечатва всички числа, които не се намират в интервала [26, 100]

Желано поведение на програмата (примера е от команда с буква „b“):

A screenshot of a computer

Description automatically generated

**Задача 15: WorkingWithSomeLetters**

Направете програма, която генерира редица от 100 случайни символа (български букви). Да се отпечатат тези от тях, които са:

- широка гласна (а, е, о)

- тясна гласна (и, у, ъ)

- съставна гласна (ю, я)

- беззвучна съгласна (п, ф, к, т, ш, с, х, ц)

- сонорна съгласна (л, м, н, р)

- звучна съгласна (б, в, г, д, ж, з, ч)

В отпечатването на информацията трябва да се съдържа и на кое място в редицата се намира буквата.

Пример за редица от 100 случайни букви на кирилица:

“уеишщксдзцьяаожгтнвмчюйъэфхпрлбуеишщксдзцдьяаожгтнвмчюйъэфхпрлбуеишщксдзцьяаожгтнвмчюйъэфхпрлбуеишщк”

За да работи с български букви, вашата програма трябва да съдържа програмен код за енкодинг, който изглежда по следния начин:

Console.OutputEncoding = Encoding.UTF8;

Желано поведение на програмата:

A screenshot of a computer

Description automatically generated

* + - 1. Методи

В ежедневието ни, при решаването на даден проблем, особено, ако е по-сло­жен, прилагаме принципа на древните римля­ни "разделяй и владей". Съгласно този принцип, проблемът, който трябва да решим, се раз­деля на множество по-малки подпроблеми. Самостоятелно разгледани, те са по-ясно дефи­нирани и по-лесно решими, в сравнение с търсенето на реше­ние на изходния проблем като едно цяло. Накрая, от решенията на всички подпроблеми, създаваме реше­нието на цялостния проблем. По същата аналогия, когато пишем дадена програма, целта ни е с нея да решим конкретна зада­ча. За да го направим ефективно и да улесним работата си, при­ла­гаме принципа "разделяй и владей". Разбиваме поста­вената ни задача на подзадачи, разработваме решения на тези под­зада­чи и накрая ги "сглобя­ваме" в една програма. Решенията на тези подзадачи наричаме подпрограми. Метод (method) е съставна част от програмата, която решава даден проблем, може да приема параметри и да връща стойност. Структурата на един метод изглежда по следния начин:

static void ShowTheNameInConsole(string name)

{

      Console.WriteLine(name);

}

**Задача 16: GetMax**

Създайте метод на име GetMax, който взима като параметри три числа и връща стойността на най-голямото от тях. Създайте програма, която взима като вход от конзолата три числа и използва метода GetMax, след това отпечатва в конзолата резултата от метода.

|  |  |  |
| --- | --- | --- |
|  | **Вход** | **Изход** |
| 2  &  3  &  4 | The biggest number is: 4 |

Желано поведение на програмата:

A screenshot of a computer

Description automatically generated

**Задача 17: GeometryCalculator**

Създайте програма, която може да пресмята лицата на две различни геометрични фигури:

- Триъгълник

- Квадрат

Третото условие е да може да преобразува градуси в радиани. Формулата за преобразуване на градуси в радиани е (Rad = π/180 \* градусите).

В началото програмата трябва да попита потребителя, коя функция иска да използва и да показва специфично меню спрямо неговият избор. Решението трябва да бъде реализирано чрез методи.

Желано поведение на програмата:

A screenshot of a cell phone

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* + - 1. Основни методи при работа със символни низове

В модерните приложения постоянно се налага да се извършват манипулации върху символни низове (текстове). В тази тема ще се запознаем с някои от основните методи на класа String:

* String.ToLower(); - всички букви стават малки
* String.ToUpper(); - всички букви стават главни
* String.IndexOf (подниз); - връща позицията, на която се намира подниза
* String.Substring(начален индекс, дължина след началото); - връща подниз, който започва от началния индекс и продължава до броя на дължината след началото
* String.Split(символ за разделяне); - връща масив от поднизове, които са разделени по символа за разделяне
* String.Replace(подниз за замяна, желан нов подниз на негово място); - заменя стар подниз с нов
* String.Trim(); - премахва празните места в началото и края на низа

**Задача 18: StringManipulations**

string text = "Hello, I am Deivid Mladenov and I appreciate your attention!"

Създайте програма, която извършва следните манипулации върху даденият символен низ и ги разпечатва в конзолата:

* началната стойност на низа
* превръща стойността на низа в малки букви
* превръща стойността на низа в главни букви
* извежда индекса на подниза „Deivid“
* замества „attention“ с „work“

Желано поведение на програмата:

A screenshot of a computer

Description automatically generated

* + - 1. Обработка на изключения

Изключение (exception) в програмирането в общия случай представлява уведомление за дадено събитие, нарушаващо нормалната работа на една програма. Изключенията дават възможност необичайните събития да бъдат обработвани и програмата да реагира на тях по някакъв начин. Когато възникне изключение, конкретното състояние на програмата се запазва и се търси обработчик на изключението (exception handler).

Изключенията се предизвикват или "хвърлят" (throw an exception) от програмен код, който трябва да сигнализира на изпълняващата се програма за грешка или необичайна ситуация. Например ако се опитваме да въведем символен низ, но програмата очаква число, кодът, който чете входа от конзолата, ще установи това и ще хвърли изключение с подходящо съобщение за грешка.

В C# има заложен подход, чрез който се прихващат изключенията. Този подход е познат като използване на try-catch блокове. Основна try-catch структура:

try

      {

            код, който очаквате, че може да хвърли изключение

      }

      catch (Exception ex)

      {

            код за обработка на изключението

      }

**Задача 19: JustCalculator**

Създайте програма, която извършва базовите операции на калкулатор (+, -, \*, /) и изписва резултатите от действията в конзолата. Подсигурете че вашата програма обработва изключенията и изписва в конзолата точното съобщение, което конкретното изключение съдържа.

Желано поведение на програмата:

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

* + - 1. Създаване и използване на обекти

Обектно-ориентираното програмиране е модел на програмиране, който използва обекти и техните взаимодействия за изграждането на компютърни програми. По този начин се постига лесен за разбиране, опростен модел на предметната област, който дава възможност на програмиста интуитивно (чрез проста логика) да решава много от задачите, които възникват в реалния свят. Класът дефинира абстрактните характеристики на даден обект. Той е план или шаблон, чрез който се описва природата на нещо (някакъв обект). Класовете са градивните елементи на ООП и са неразделно свързани с обектите. Нещо повече, всеки обект е представител на точно един клас.

Ще дадем базов пример за това как може да създадем и използваме обект в нашата програма. За примера ще създадем едно нормално конзолно приложение и към него ще добавим обект, който ще именуваме User.

A screenshot of a computer screen

Description automatically generated

1 – Десен бутон върху конзолното приложение

2 – Избираме Add

3 – Class

A screenshot of a computer screen

Description automatically generated

4 – Подсигуряваме се че сме избрали Class

5 – Именуваме с разбираемо име, в нашият случай User

6 – Натискаме Add

A screenshot of a cell phone screen with text

Description automatically generated

7 – Виждаме, че нашият клас е успешно създаден

A screenshot of a cell phone

Description automatically generated

8 – Задаваме нивото на достъп (access modifier) public за момента, в бъдеще ако учите ООП ще се запознаете с тях в детайл

A screenshot of a computer screen

Description automatically generated

9 – Дефинираме свойствата (properties) на обекта

A screenshot of a computer screen

Description automatically generated

10 – Инстанцираме нашият обект User в Program.ts файла на конзолното ни приложение

11 – Попълваме стойности на неговите свойства

12 – Разпечатваме стойностите на обекта

A screenshot of a computer

Description automatically generated

13 – На стъпка 10 създадохме инстанция на обекта User, но той беше празен и се наложи на стъпка 11 да попълним ръчно стойностите на свойствата му. Това е много несигурен подход, за това може да използваме конструктор (constructor), в който да дефинираме нужните стойности на свойствата още при инстанцирането

A screenshot of a cell phone

Description automatically generated

14 – След създаването на конструктор с параметри, компилаторът дава съобщение за грешка и ни подсказва, че трябва да въведем нужните стойности за да бъде създадена инстанция

A screenshot of a computer screen

Description automatically generated

15 – Въвеждаме стойностите в конструктора

A screenshot of a computer

Description automatically generated

16 – Класовете ни дават право да създаваме методи вътре в тях, за това в нашият клас User ще създадем метод, който го представя като изписва данните в конзолата. Ще именуваме нашият метод с говорещо за извършваното от него действие име, а именно IntroduceMyself

A screenshot of a cell phone

Description automatically generated

17 – Извикваме метода IntroduceMyselft през създадената инстанция в Program.cs файла и виждаме резултата от неговото изпълнение след стартиране

A screenshot of a cell phone

Description automatically generated

**Задача 20: UserWithActions**

Като използвате демо за създаване на класове разширете решението, като добавите нови функционалности към него:

1. Добавете ново свойство на User класа наречено City
2. Добавете нов метод AmIAdult, който изписва в конзолата дали User е пълнолетен или не спрямо Age, който е въвел
3. Добавете нов метод WhereAmIFrom, който изписва в конзолата от къде е User спрямо City
4. Всички свойства на User трябва да се попълват от конзолата
5. Добавете меню, от което може да се избира кой метод да се активира

Желано поведение на програмата:

A screenshot of a cell phone

Description automatically generated