Петър Стойков
Иван Иванов

ПРАКТИЧЕСКО РЪКОВОДСТВО ПО ПРОГРАМИРАНЕ С ИЗПОЛЗВАНЕ НА C++

Част 1

Въведение в програмирането
Първоначално запознаване с Visual C++
Основни понятия в езика C++
Базови алгоритмични структури

Авторите са преподаватели по Информатика. **Доц. д-р инж. Петър Иванов Стойков** е преподавател в катедра "Компютърна информатика" към Факултета по математика и информатика на Шуменския университет "Епископ Константин Преславски", **доц. д-р инж. Иван Томов Иванов** е ръководител катедра "Информационни технологии" към Факултета по библиология, информационни системи и обществени комуникации на Специализираното висше училище по библиотекознание и информационни технологии.

ISBN 978-954-8641-19-7

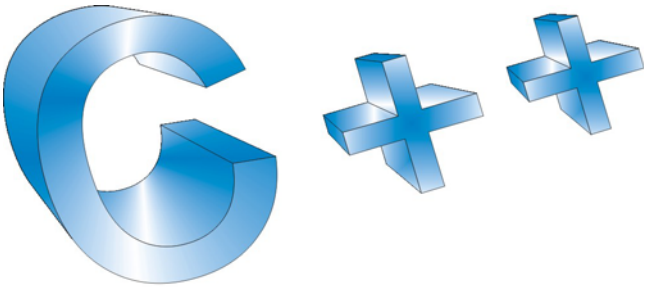
Петър Стойков
Иван Иванов

ПРАКТИЧЕСКО РЪКОВОДСТВО ПО ПРОГРАМИРАНЕ С ИЗПОЛЗВАНЕ НА C++

Част 1

Въведение в програмирането
Първоначално запознаване с Visual C++
Основни понятия в езика C++
Базови алгоритмични структури

Учебник



Издателство „Фараго”
София, 2007

Петър Стойков
Иван Иванов

ПРАКТИЧЕСКО РЪКОВОДСТВО ПО ПРОГРАМИРАНЕ С ИЗПОЛЗВАНЕ НА C++

Част 1

Въведение в програмирането
Първоначално запознаване с Visual C++
Основни понятия в езика C++
Базови алгоритмични структури

София, 2007

Peter Stoykov
Ivan Ivanov

Practical Guide for C++ Programming

Part 1

Introduction to Programming
Introduction to Visual C++
Basic concepts of the C++ language
Basic Algorithms and Data Structures

Sofia, 2007

ПРЕДГОВОР

В част първа на учебника е обоснована необходимостта от изучаване на програмирането, разкрити са основните понятия необходими при изучаването и са разгледани и класифицирани езиците за програмиране.

Дадена е първоначална информация за програмната среда Visual C++, онагледена с подходящи примери, позволяваща на обучаемите да работят със същата.

Разгледани са основните правила на синтаксиса и семантиката на езика C++, комуникацията през конзолата, структурата на програмите и операциите и математическите изрази използвани в същия.

Представени са основни понятия в алгоритмизацията.

Учебникът е предназначен за студенти изучаващи дисциплината Програмиране (с използване на C++) във Факултета по математика и информатика на Шуменския университет "Епископ Константин Преславски" и в Специализираното висше училище по библиотекознание и информационни технологии. Може да се използва и от докторанти и всички желаещи да се запознаят и започнат да програмират на езика C++. За усвояването на учебния материал е необходима само първоначална компютърна грамотност, получена в средния курс на обучение.

Съдържанието на учебника е апробирано в лекционни курсове в Шуменския университет "Епископ Константин Преславски" и Специализираното висше училище по библиотекознание и информационни технологии.

Материалът от книгата влиза изцяло в семестриалните изпити на студентите от Факултета по математика и информатика по специалностите: компютърна информатика, икономическа информатика, компютърни системи и технологии и математика и информатика, на студентите изучаващи програмиране от Факултета по технически науки и двата факултета от Шуменския университет и на студентите по информационни технологии, информационно брокерство и информационна сигурност от Специализираното висше училище по библиотекознание и информационни технологии.

СЪДЪРЖАНИЕ

Глава 1. Въведение в програмирането

1.1. Защо е необходимо да се изучава програмиране?

Вие вече сте използвали софтуер, вероятно като текстов редактор (*word processing*) или електронна таблица (*spreadsheet*), и сте решили успешно стоящите пред вас задачи. Вероятно сега се учудвате, че трябва да учите как програмистът създава софтуер (*software*). Една програма е последователно подреждане на инструкции, които управляват компютъра за да изпълни задачите които вие искате да решите и да се стигне до желаното от вас решение. Разликата между обучението по използване на готов приложен софтуер като споменатите по-горе текстов редактор или електронна таблица, което сте правили досега и това да се обучите в сложното изкуство на програмирането е, че при изучаването на готовия софтуер, вие сте се стремели да разберете как компютърът „мисли", докато при разработването на програма, вие използвайки определени правила заставяте компютъра да „мисли" по определен, необходим за вас, начин.

Езикът за програмиране (*programming language*[1]) е съвкупност от семантични конструкции и синтактични правила, които се използват за представяне на алгоритми във форма, удобна за обработка с компютър. Езикът за програмиране е изкуствен език чрез който се управлява компютъра. Съществуват много езици за програмиране. Тук ще научите как се реализира токова управление на компютъра посредством процеса на програмиране. Дори можете да откриете, че може би желаете да станете програмист.

Съществуват най-малко три достатъчно силни основания за да изучавате програмиране:

- Програмирането (*programming*) ви помага да разберете компютрите (*computers*). Компютърът е само един инструмент. Ако вие се научите да пишете най-обикновени програми, вие ще придобиете повече знания относно това как работи компютърът. Казано с думите на Алън Кей: „За информатиката компютрите са това, което са инструментите за музиката. Софтуерът е партитурата, чиято

[1] Programming language *същ.* език за програмиране = всеки изкуствен език, който може да бъде използван за дефиниране на последователност от инструкции, които могат да бъдат обработени и изпълнени от компютър. Дефинирането на понятието „език за програмиране" е трудно, но основната употреба означава, че процесът на транслация – на сорс кода, изразен с помощта на езика за програмиране, в машинен код, с който компютъра работи – може да бъде автоматизиран с помощта на друга програма, като компилатор. По този начин се изключват английският и други естествени езици, въпреки че се използват подмножества на английския, които са разбираеми от някои езици от четвърто поколение. Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 147.

интерпретация разширява кръга на нашите действия и повдига нашия дух."[2]

- Написвайки няколко програми вие увеличавате вашето равнище на увереност. Много хора намират голямо лично удовлетворение в създаването на последователност от инструкции, тоест в написването на програми, които намират решението на даден проблем.
- Изучавайки програмирането, вие бързо ще намерите отговор на въпросите: дали харесвате програмирането и дали имате аналитична гъвкава мисъл, от която се нуждаят програмистите. Дори да решите, че програмирането не е за вас, разбирайки процесът му, ще повишите възможностите си за преценка какво могат да правят програмистите и компютрите.

Необходимо е да се отбележи, преди да продължим, че вие няма да станете програмист когато прочетете тази глава, а дори и да стигнете до края на учебника. Вещината на програмирането се добива посредством практика и тренировки, които са извън компетенцията на този учебник. Все пак ще получите първоначално запознанство за това как програмистът намира решението на множество проблеми.

Изучавайки програмирането в края на краища трябва да си отговорите на съдбовния за вас въпрос: „Годен ли съм аз за добър програмист, или трябва да се посветя в друго направление на Информатиката?"

Изучаването на тази дисциплина е крайно необходимо и за студентите, които ще тръгнат в другите направления. Тъй както за програмистите е необходимо добре да познават хардуера, паметта на компютъра и приложните системни програмни продукти, то за хардуеристите и останалите колеги, които ще се посветят на други направления в информатиката и информационните технологии ще е полезно задълбочено да познават програмирането, за да могат да влязат в същността на хардуера и на приложните системи. Така например изучавайки езика С, единствено по този начин ще им стане ясно как функционира цялата Microsoft Office система.

Всеизвестен е световният глад от компютърни специалисти и в частност от програмисти. Христоматиен пример е идеята на Бил Гейтс приета от Бил Клинтън, която засега все още не дава положителни резултати. Гейтс е отправил към Клинтън предложение целящо да се справи с липсата на огромен брой компютърни специалисти. Компютърният милиардер предлага национална програма, която има за цел да обзаведе всяко училище и библиотека с достатъчно на брой компютри, за да имат хората (и особено младите!) на практика неограничен достъп до тях. Програмата засега не дава желания резултат.

---

[2] А. Кей „Софтуер-програми за компютри". Математика в миниатюри. Трета книга, „Наука и изкуство" С.1987

В САЩ в началото на XXI век имаше недостиг от 1 300 000 компютърни специалисти. Положението в Европа далече не е по-добро.

Общата перспектива в компютърния бранш е обещаваща. Бюрото по трудова статистика на САЩ информира, че през последното десетилетие на XX век има 72% нарастване на програмистите и 69% нарастване на системните аналитици (*systems analysts*). Прогнозира се тези две професии да бъдат втора и трета по количеството на нарастване на работните места. (Ако Вие проявявате любопитство, на първо място по нарастване на работните места се прогнозира професията на помощник адвокат.) Основания за непрекъснато нарастване на работещите в компютърния бранш (*computer field*) са нарастването на компютрите, нарастване на компютърните приложения (*applications*) и нарастване на потребителите на компютри.

Друго нещо което трябва да се подчертае в това въведение е, че програмирането изисква продължително и постоянно обучение, че това е професия, която ако студентът не я чувства като призвание и програмирането да стане негово хоби, ще срещне много трудности и разочарования. По принцип трябва да сте наясно, че започвайки да се занимавате с Информатика и в частност с Програмиране, вие започвате едно плуване срещу течението, което трябва да продължи с добър темп през целия творчески живот. Всяко едно, макар и кратко отпускане, ще позволи на течението да ви връща назад.

Необходимо е да се подчертае, че нито едно висше учебно заведение в света не дава на изхода си готови програмисти, тоест в света съществува огромен дефицит от програмисти, хиляди, а може би милиони атакуват тази примамлива професия, но малцина могат да отговорят на критериите й за включване в гилдията на програмистите.

По-рано, когато имаше дефицит от работна сила и държавата осигуряваше заплащане без особено да се прецизира за себестойността на произведеното, много предприятия, а дори и институти, си позволяваха да назначават хора с начални познания в областта на програмирането, които в процеса на работата се обучаваха, а и на почти всички от тях се осигуряваха и курсове за обучение.

Положението и у нас се измени коренно. Нито един мениджър в бранша не би си позволил да назначи на работа човек с минимални познания, да го обучава, а на края когато се квалифицира най-вероятно и да го загуби. Естествено, че във фирмата се усъвършенства квалификацията на всички нейни програмисти. Един от създателите на Intel – Гордън Мур в своите прогнози извежда през 1965 г. така наречения по-късно „Закон на Мур”, който гласи, че всяка година производителността на процесорите ще се удвоява. Той актуализира постулата си през 1975 и 1995 г., променяйки цикъла на удвояване съответно на 1,5 и 2 години. Тази прогноза се доказа на практика от времето на Мур до сега и няма признаци да не продължи да действа и за

в бъдеще. „Очакваме Законът на Мур да продължи да е в сила дори и след десет години" заявява говорител на Intel.[3] В тази безкрайна надпревара темпото за програмистите ще нараства, което е и едно интересно предизвикателство за всяка личност.

Традиционното развитие на кариерата в компютърния бранш е изминаването на пътя от програмиста (*programmer*) през системният аналитик (*systems analyst*) до ръководител на проект (*project manager*). Компютърните професионалисти понякога специализират в някои направления на компютърната индустрия като например: предаване на данни (*data communications*), управление на бази от данни (database management), персонални компютри, компютърна графика или компютърно оборудване. Други могат да специализират в специфични направления на бизнеса които са свързани с компютърните технологии, такива като банковото дело и застраховането. Трети тръгват по свой собствен път захващайки се с подходяща консултантска или предприемаческа дейност.

1.2. Основни понятия в програмирането

Както ви е известно, компютърната система се състои от три основни компонента: хардуер, софтуер и човешки ресурс. *Хардуерът (hardware)* е апаратната част на компютъра. Той самият сам по себе си не може да работи. Множеството от програми го управляват като казват на хардуера какво да прави. Те съставляват *софтуера (software)*. Хората са най-важният компонент на изчислителната система. Тези, които пишат програми, както вече ви е известно, се наричат програмисти (*programmers*), а тези, които просто използват възможностите на хардуера и софтуера, за да решават различни проблеми, се наричат потребители (*users*).

Хардуер (hardware)

За да функционира, компютърната система трябва да разполага със следните устройства: за вход, за обработка, за изход, за съхранение и за комуникация.

- ***Входните устройства*** *(input devices)* приемат данните и програмите, преобразуват ги в електрически сигнали и ги предават на оперативната памет, където се съхраняват временно. Устройствата за вход са най-различни: клавиатура, мишка, таблет[4], видеокамера,

[3] В. Computerworld/BG бр. 06 19.02.2000 г., стр. 6.

[4] tablet, graphics tablet *същ*. графичен таблет = устройство, използвано за въвеждане на информация за позиция на графика в инженерството, дизайна и приложенията за илюстрации. Плоска пластична правоъгълна дъска е оборудвана с топче или химикалка

микрофон, скенер и др. Стандартното входно устройство за компютъра е клавиатурата.

- ***Процесорът*** *(processor)* изпълнява командите на програмата и обработва външните данни. Тези програми и данни се съхраняват в част от *оперативната памет (internal/main memory)* , наречена RAM (*Random Access Memory – памет за четене и писане)*. Съдържанието на RAM се загубва при изключване на компютъра (енергозависима памет). Другата част от оперативната памет съдържа информация, която е фабрично записана и не може да се изтрива или променя. Нарича се ROM (*Read Only Memory –памет само за четене)*. При изключване на компютъра тя не се загубва (енергонезависима)..
- ***Изходни устройства*** *(output devices)* получават данните, които са в резултат от *работата* на процесора. Такива устройства са дисплеят, принтерът, плотерът, високоговорителят и др. Стандартното изходно устройство за компютъра е дисплеят.
- ***Външна памет*** *(external memory)* съхранява програми и данни извън компютъра. Дискетите, твърдите дискове и компактдисковете са външна памет. Дискетите се поставят в специално устройство, наречено дисково или флопидисково, а твърдите дискове се вграждат в компютъра. При изключване на компютъра съхраняваната във външната памет информация се запазва.
- ***Комуникационни устройства*** *(communications devices)* осигуряващи трансфер на данни от един компютър до друг посредством комуникационен носител, като телефон, микровълнов предавател, сателитна връзка или физически кабел.[5]

Има устройства, които могат да бъдат както входни, така и изходни. Например: флопидисковото устройство, твърдият диск. Съвкупността от входни и изходни устройства се нарича периферия (*peripheral*).

Компютрите могат да имат различни по тип и брой процесори, различни входни и изходни устройства, различен капацитет на паметите. Съвкупността от процесори, памети и периферия определя различни *конфигурации* компютри (*configurations*).

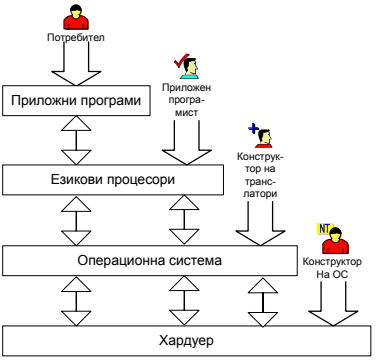
(наречена още писец) и сензорна електроника, която докладва на компютъра позицията на топчето или писеца, а той превежда тази информация в позиция на курсора на екрана. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 345.

[5] За повече виж Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 161.

Софтуер (software)

Софтуерът се нарича още програмна част или програмно осигуряване. Софтуерът се дели на два основни типа: системен и приложен софтуер. Системният софтуер включва операционни системи и инструменти за разработка на софтуер. Същността на последните са езиковите процесори.

За целите на програмирането ще използваме следното абстрактно виждане върху структурната организация на компютърната система (фиг. 1.1).



Фиг. 1.1. Структура на компютърната система

На горепосочената фигура компютърната система се разглежда на четири йерархични нива, като всяко включва определени функции и интерфейс[6]. Необходимо е да се подчертае, че използването на интерфейс на дадено ниво не означава, че не може да се използват интерфейсите на по-ниските нива.

Хардуерното ниво дава основните изчислителни ресурси на системата. Интерфейсът 1, който представлява система от машинни

---

[6] Interface същ. интерфейс
1. Точката, в която се осъществява връзката между два елемента, така че те да могат да работят заедно и да обменят информация.
2. Софтуер, който позволява на една програма да работи с потребителя (потребителски интерфейс, който може да бъде с команден ред, управляван от менюта интерфейс или графичен интерфейс), с друга програма, като операционна система или хардуера на компютъра.
3. Карта, конектор или друго устройство, което свързва частите на хардуера с компютъра, така че информацията да може да бъде придвижвана от място на място. Например стандартизираните интерфейси, като стандарта RS-232-C и SCSI, позволяват комуникация между компютри и принтери, или дискове. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 403.

команди[7] е класическия интерфейс на системата между хардуерната и софтуерната части. Той представлява, независимо от начина на реализация, абстрактно представяне на физическата структура на компютъра.

Над хардуера е операционната система, включваща средства за управление на ресурсите. Операционната система изгражда абстрактна (виртуална) машина над хардуерната част, предоставяйки нови функции, реализирани програмно. Компютърът става „разширена" машина, която служи за ефективно реализиране на компилаторите[8] и приложните програми. Операционната система предоставя интерфейс 2 под формата на системни извиквания (заявка за системно обслужване), които облекчават програмирането. При проектирането на компилаторите, както и при програмиране на асемблерен език[9], е необходима подробна информация за хардуера. Първичните програми се транслират на езика на разширената машина, включващ наред с машинните команди и команди на операционната система (системни изисквания). Затова конструкторът на транслатори[10] трябва да има достъп до интерфейс 1.

Следващото равнище дава необходимите средства на приложния програмист, като компилатори на езици на високо ниво, асемблери, интерпретатори и др. Интерфейсът 3 не се определя само от езиците за програмиране. Освен обръщенията към операционната система, предоставяни от компилатора (например за разпределение на паметта), още се използват командни езици, редактори на текстови

---

[7] Machine code *същ*. машинен код = крайният резултат от компилирането на на асемблерен език или език от по-високо ниво, като C или Pascal: поредици от нули и единици, които са заредени и изпълнени от микропроцесор. Машинният код е единствения език, който компютрите разбират; всички други езици за програмиране представляват начин за структуриране на човешки език, така че хората да могат да заставят компютрите да извършват конкретни задачи. Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 3.

[8] Compiler същ. компилатор

1. Всяка програма, която преобразува един набор от символи в друг, като следва набор от синтактични и семантични правила.

2. Програма, която преобразува изходния код на програма, написана на език от високо ниво, в обектен код преди изпълнението им. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 165

[9] Assembly language същ. асемблерен език = език за програмиране от ниско ниво, използващ съкращения и мнемонични кодове, в които всяка конструкция отговаря на една единствена машинна инструкция. Асемблерният език се преобразува в машинен език от асемблер и се отнася за даден процесор. Като предимства от използването на асенблерен език могат да се посочат по-високата скорост на изпълнение и директното взаимодействие на програмиста със системния хардуер. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 57.

[10] Translator същ. транслатор = програма, която превежда формат на данни от един език на друг. Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 291.

файлове, свързващи и зареждащи програми, помощни програми, необходими за разработката на приложни програми.

Най-високото равнище определя функциите на компютърната система като цяло. Интерфейсът 4 е комуникация с „външния свят" – потребители и обслужващ персонал (системни програмисти, администратори на системата). Той се реализира от множество приложни програми. Освен този интерфейс потребителите могат да използват и интерфейсите на по-ниските нива, например командни езици и езици за програмиране.

Операционната система (*operating system*), както вече казахме е софтуер, който управлява заделянето и използването на хардуерни ресурси като памет, процесорно време, дисково пространство и периферни устройства.

Операционната система в най-голяма степен определя облика на компютърната система. Потребителите активно използващи компютърна техника често се затрудняват да дадат определение за операционна система. Същото се е развивало във времето. За нашите млади читатели трябва да кажем, че в зората на компютрите е нямало операционни системи, учебна дисциплина под това име и съответно и изпити, но който е закъснял с раждането си, задължително трябва да усвои тази материя.

През 60-те години на миналия век операционната система е можело да бъде определена като „програмни средства осигуряващи работата на апаратурата"[11].

Днес под операционна система се разбира софтуер, който управлява заделянето и използването на хардуерни ресурси като памет, процесорно време, дисково пространство и периферни устройства. Операционната система е фундаменталният софтуер, на който разчитат приложенията.[12]

На много места се използват съвкупност от определения, разкриващи отделните функции на операционната система, които варират в различни граници[13]. Ние ще изходим от факта, че операционната система изпълнява две основни малко свързани функции[14]:

---

[11] Дейтел, Г.:"Введение в оперционные системмы" В двух томах. Том 1. Москва „Мир" 1987 стр. 20.

[12] operating system Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 83.

[13] Така например в книгата на Л. Николов, Операционни системи, Трето преработено и допълнено издание, Сиела 2001 са посочени на стр.11 шест определения за операционна система.

[14] Това ни доближава с подхода на Таненбаум, изложен в Э. Таненбаум, Современные операционные системы, 2-е издание, Питер 2005 стр. 24.

- осигурява средства на потребителя за работа с компютъра, тоест разширява възможностите на хардуера;
- повишава ефективността от използване на компютъра, чрез рационално управление на неговите ресурси, тоест е мениджър на ресурсите му.

Операционната система като разширение на машина

Използването на повечето компютри на ниво машинен език е трудно, особено това важи за входно-изходните операции. Например, за организирането на четене на блок данни от диска, трябва да се използват 16 различни команди, всяка от които изисква по 13 параметъра (например: блок на диска, номер на сектора и т.н.). Когато завършат операциите с диска, контролерът връща 23 стойности, отразяващи наличие и тип на грешката, които очевидно трябва да се анализират.

Ако в задълженията на програмиста не влизат въпросите свързани с програмирането на входно-изходни операции, малко са тези които биха се занимавали с програмирането на тези операции. Достатъчно е програмистите при работа с диска да си го представят като съвкупност от именувани файлове. Работата с файлове се свежда към следните операции:

- отваряне на файла;
- четене и/или записване;
- затваряне на файла.

Въпроси, подобни на тези - трябва ли при записването да се използва усъвършенствана честотна модулация или в този момент в какво състояние се намира двигателят за преместване на четящите глави, не трябва да вълнуват потребителя.

Програмата, която скрива от програмиста всичките реалности на хардуера и предоставя възможност за просто и удобно преглеждане, четене или записване на различни файлове, това е, разбира се – операционната система.

Точно така, както операционната система изолира програмиста от апаратурата на диска и му предоставя прост файлов интерфейс, тя извършва нелеките задачи, свързани с обработката на прекъсванията, управлението на таймерите и оперативната памет, а също така и други задачи на ниско ниво[15]. Във всеки случай, благодарение на операционната система, потребителят общува много по-лесно и удобно

[15] Читателят при първият прочит не трябва да се притеснява от непознаването на изброените понятия с повечето от които се сблъсква за първи път. Важно е да се почувства огромния обем от задачи които решава операционната система.

с абстракцията въображаема машина, отколкото с реалната апаратура, лежаща в основата на абстрактната машина.

От тази гледна точка операционната система предоставя на потребителя разширена или виртуална машина, която леко се програмира и с която по-лесно се работи, отколкото непосредствено с апаратурата от която е съставена реалната машина.

Операционната система като система за управление на ресурси

Идеята за това, че операционната система е преди всичко система, осигуряваща удобен интерфейс на потребителите, съответства на погледа върху нея отгоре-надолу.

Другият поглед, отдолу-нагоре, дава представа за операционната система като механизъм, управляващ всички части на сложна система. Съвременните компютърни системи се състоят от процесори, памети, таймери, дискове и други запомнящи устройства, мрежов комуникационен хардуер, както и много други устройства. В съответствие с втория подход, операционната система разпределя процесорите, паметта, устройствата и данните между процесите, които се конкурират за тези ресурси. Операционната система трябва да управлява всички ресурси на компютъра по такъв начин, че да осигури максимална ефективност на функциониране. Критерии за ефективност може да бъде пропускателната способност или реактивност на системата.

Управлението на ресурсите включва решаването на две общи задачи, не зависещи от типа ресурс:

- ***планиране на ресурси*** - тоест определяне на кого, кога, а за делимите ресурси, и колко от дадения ресурс да се даде;
- ***следене състоянието на ресурсите*** - тоест поддържане на оперативна информация за това, зает ли е или не даден ресурс, а за делимите ресурси - колко от него е разпределен и колко е свободен.

За решаването на тези общи задачи по управление на ресурсите, различните операционната система използват различни алгоритми, което в крайна сметка и определя техния облик като цяло, включвайки характеристиките: производителност, област на използване и даже потребителски интерфейс. Така например, алгоритъмът за управление на процесора в значителна степен определя каква е операционната система.

Операционните системи са един от най-сложния софтуер и ще бъдат изучени от вас в отделна дисциплина.

С усъвършенстване на средствата за програмиране освен развитието на езиците за програмиране, макар и със закъснение, се развива и автоматизацията на програмирането като се развива софтуерното направление „програмна среда”.

Средата за програмиране автоматизира цялостния процес по създаването, транслирането, тестването, изпълнението и документирането на програмите, на което ще се спрем впоследствие. Те включват: *език за програмиране, транслатор (компилатор или интерпретатор), свързващ редактор, изпълнителна система, система за проверка на програми, система за поддържане на библиотеки и текстови редактори.* Тези компоненти са свързани в система чрез управляваща програма.

Приложни програми (application programs)

Приложният софтуер е клиентски или пакетен. Клиентският софтуер се прави по поръчка на лица или фирми и отговаря на спецификата на тяхната работа. По-разпространени са пакетите приложни програми. Те нямат конкретен поръчител и типични представители за тях са:

- *Текстообработка (word processing) – запознати сте с текстообработващата програма Word.*
- *Електронна таблица (electronic spreadsheet)- продуктът Excel.*
- *Система за управление на база от данни (data base) - Access.*
- *Графика (graphics) Paint, Corel Draw*
- *Комуникации (communications)* – този софтуер се използва за различни видове комуникации: четене на web-страници; провеждане на телефонни разговори чрез компютър; изпращане и получаване на съобщения по факс с използване на компютър; получаване и изпращане на електронни писма; ползване на общи ресурси – хардуерни и софтуерни. Тук от особено значение са мрежите.

Програмистите *(programmers)* са хората, които професионално се занимават с програмиране. При появата на компютрите потребителите са били и програмисти, като е трябвало по необходимост да изучат някакъв език за програмиране (най-често машинен), за да създадат нужната програма. Обикновено такива програми се ползват трудно от други потребители. По-късно и в програмирането настъпва разделение на труда. Появяват се професионалните програмисти, които вече почти нямат отношение към използваните от тях програми. Професионалните програмисти от своя страна са ***системни*** и ***приложни*** програмисти. В зависимост от размера на проекта и работната среда програмистът може да работи сам или като член от екип, да участвува в част или в целия процес от проектирането до изпълнението, или пък да напише цялата или част от програмата.

Под ***програмиране*** (*programming*) се разбира умението и науката за създаване на компютърни програми[16]. Програмирането започва с овладяването на един или повече езици за програмиране като Basic, C, Pascal, C++ или Асемблер. Познаването на някакъв език не е достатъчно за създаване на добра програма. Трябват много повече знания, като например експертни познания по теория на алгоритмите, проектиране на потребителски интерфейс и познаване характеристиките на хардуерните устройства. Компютрите са строго логически машини и затова програмирането изисква подобен логически подход към проектирането, написването (кодирането), тестването и дебъгването[17] на програми. Езиците на ниското ниво, като асемблерните езици, изискват и познаване възможностите на микропроцесора, както и на основните вградени в него инструкции. При модулния подход, препоръчван от много програмисти, проектът се разделя на малки, управляеми модули – самостоятелни функционални единици, които могат да бъдат проектирани, написани, тествани и дебъгвани поотделно, преди да бъдат използвани в по-голяма програма.

1.3. Етапи при разработване на програма

Разработването на една програма включва стъпки подобни на работата по решаването на една задача. Процесът на програмирането преминава през пет главни етапа:

- *Дефиниране на проблема (Defining the Problem)*
- *Планиране на решението (Planning the Solution)*
- *Кодиране на програмата (Coding the Program)*
- *Тестване на програмата (Testing the Program)*
- *Документиране на програмата (Documenting the Program)*

Съществува набор от подобрени методи за програмиране, които се прилагат от години. Някои от тях включват в себе си обикновени общи идеи за програмирането, а други се основават на постиженията в областта на техниката на програмиране.

Технологията на програмирането определя последователните етапи при създаването на една определена система. Тя е необходима при прилагането на индустриални методи в програмирането – разделение на труда, паралелно извършване на някои дейности, съвместимост на отделно създадените програмни подсистеми. Това изисква спазването на определени стандарти и дисциплина в програмирането.

---

[16] Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 146.

[17] Debug гл. дебъгвам = откривам и коригирам логически или синтактически грешки в програма, или неизправност в хардуер. При хардуера се използва по-често терминът отстраняване на проблем (troubleshoot), особено когато проблемът е голям. Английско-български тълковен компютърен речник том 1: A-L стр. 212.

В някои литературни източници процесът на разработването на един програмен продукт се разделя на три основни дейности: *Определяне на подхода към задачата, Кодиране на програмата* и *Тестване на програмата.*

Ще ги обсъдим като се спираме основно на петте етапа.

1.3.1. Дефиниране на проблема (Defining the Problem)

В този етап програмистът уточнява входните данни за програмата, по какъв начин ще бъдат обработвани данните от програмата и резултатите, които трябва да се получат от същата. Изяснява се в какъв вид и как ще бъдат обявени крайните резултати. Главно внимание се отделя на последователността при въвеждането на информация в компютъра. Определя се последователността при извършването на основните операции в процеса на обработването. Логическата схема, която се изгражда, има по-общ характер и отделните операции не се разглеждат подробно.

Ако вие сте един програмист, който е натоварен със задача да разработи реално един програмен продукт, то вие трябва да се срещнете с потребителите от организацията, която е клиент, за да анализирате проблема. Ако организацията-клиент има системни аналитици, това в значителна степен ще ви облекчи работата, понеже с тяхна помощ ще си изясните нещата и ще може да ви скицират проекта. Точното определяне на задачата по дефинирането на проблема съдържа определяне на какво вие знаете (определяне на данните за въвеждане) и какво вие ще представите като резултат. Целесъобразно е след изясняването на всичко това да се направи едно писмено споразумение, което да съдържа вида на въвежданата информация, обработката й и очаквания резултат. Това не е лек процес. Той е тясно свързан с процеса на системния анализ, който трябва също добре да се владее от програмиста.

1.3.2. Планиране на решението (Planning the Solution)

През вторият етап се съставя логическа схема на програмата, тоест се написва нейния алгоритъм. Алгоритъмът е всяка точно и еднозначно описана крайна последователност от действия (операции) над определени входни данни, която винаги води до резултат (изход), който е във функционална зависимост от стойностите на входните данни.

Същата може да се опише вербално, но поради особеностите на естествения език, тя няма да отговаря на изискването за еднозначност в тълкуването на описанието. Два са начините за планиране решението на един проблем: да се начертае ***блок-схема*** (*flowchart*) или да се напише

псевдокод (*pseudocode*[18]), а са възможни и двата подхода. По същество блок-схемата представлява изобразяване решението на проблема стъпка по стъпка. Тя представлява съчетаване на стрелки, представляващи посоките които взема изпълнението на програмата и правоъгълници и други символи представящи действия. Това е една карта която показва на вашата програма какво да прави и по какъв начин да го прави. Американският национален институт по стандартизация (American National Standards Institute - ANSI) е разработил точно определени символи за блок-схемата. Малко програмисти използват за нуждите на практиката блок-схеми, но те си остават добър инструментариум, като визуално средство за обучение.[19]

Псевдокодът е нестандартен език, подобен на английския, който позволява да формулирате вашето решение с по-голяма прецизност отколкото можете да направите това на обикновен английски език, но с по-малка прецизност отколкото се изисква когато вие използвате един формализиран програмен език. Необходимостта от добро владение на английския език го прави непопулярен у нас. Псевдокодът позволява вие да се фокусирате върху програмната логика, без да се концентрирате все още върху прецизния синтаксис на един специфичен програмен език. При това псевдокодът не е изпълним на компютъра.

В някои литературни източници този етап, съвсем основателно е разделен на два отделни етапа:

Създаването на математически или информационен модел и избор на метод за решаване на математическата задача или за обработване на данните.

Проектиране – определят се модулите на програмната система и информационните връзки между тях. За всеки модул се определят входните данни и изходните резултати. Тук етапът на алгоритмизацията е изведен в етапа на програмирането.

Етапите „Дефиниране на проблема" (*Defining the problem*) и „Планиране на решението" (*Planning the solution*) са обединени в един - ***„Определяне на подхода към задачата"***, в който се препоръчва да се извърши следното:

---

[18] pseudocode същ. псевдокод. Всяка неформална, прозрачна нотация, на която е написано описанието на една програма или алгоритъм. Много програмисти пишат своите програми първо с псевдокод, който наподобява смесица от английски и любимия им език за програмиране, като C или Pascal, след което го транслират ред по ред в действителния използван език. Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 150.

[19] В учебната литература се забелязва известно подценяване на този въпрос.

1. Преди всичко да се определи задачата.

Програмата се съставя за да реши някаква задача. Очевидно, за да можете да пристъпите към решаването на някоя задача, вие трябва да разберете нейната същност, т. е. програмистът преди да започне да пише някоя програма, той трябва да знае какво трябва да прави програмата или системата, която той разработва. Затова програмистът обикновено е принуден сам да събира информацията, която му е необходима за написване на програмата.

Трудността на този подход се заключава в това, че само от програмиста зависи избягването на грешката при изпълнението на задачата (работата на системата). Ако програмистът не разбере напълно условието на задачата (работата на системата), е невъзможно да се избегнат грешки. При това изясняване програмистът може да се натъкне на маса логически противоречия, и той е длъжен да си изясни в детайли всичко, за да може програмата да съответства на заданието.

Едно от решенията на този проблем - при сериозна система - е да се възложи на програмиста[20] да направи описание на работата на системата, понеже на него му е необходима повече информация за задачата и да определи проблема в писмен вид. Такова описание става основа на целия проект и затова е необходимо да се представи на ръководството на проекта, за да може да се обсъди и коментира преди да се продължи напред. В резултат на всички тези стъпки, програмистът преди да започне да пише програмата ще е разбрал напълно проблема и ще има писмено обяснение на това в какво се състои задачата.

2. Да се раздели програмирането от решаването на задачата.

Трябва да е ясно, че в началото на разработването на системата, трябва да се различават два етапа: програмиране и решаване на задачата. Те не трябва да се смесват. Първият, очевиден етап, във всеки проект е решаването на задачата. Ако има просто решение на задачата, то програмирането/кодирането ще премине лесно. Обаче, в повечето случаи, трябва първо да се реши задачата, а след това същата да се програмира. Някои програмисти се дразнят от такъв подход. Вместо първоначално да се опитат да решат задачата, те преминават веднага към кодиране и се опитват да решат задачата, направо чрез програмиране. Този подход се разви много силно с появата на персоналните компютри.

[20] В бившите ведомствени институти за разработване на информационни технологии имаше „лукса", в работния колектив да бъде включен, специалист от проблемната област, който се занимаваше с този проблем, т. е. съществуваше разделение на труда и имаше: специалист в приложната област, алгоритмист и програмист.

3. Да се използва структурният подход.

Програмирането в значителна степен може да се опрости с прилагането на програмиране „отгоре надолу" и на структурния подход, който е застъпен широко в C++.

4. Разработката да се извършва с насоченост към крайния потребител.

Това правило се забравя често и все повече и повече програмисти се стараят да пишат все по-дълги програми. Програмата, която е решение на задачата, трябва в същото време да бъде решение и за крайния потребител. Програмистът трябва да познава и разбира крайния потребител за когото се създава програмата. В много случаи именно този краен потребител, който така или иначе ще проверява задачата, предявява много свои изисквания. Ако тези изисквания не се изпълнят, крайният потребител няма интерес от използването на дадената система.

Много важно е крайният потребител от първия път да може да работи с програмата без каквото и да е ръководство или инструкция. Човек, който е станал притежател на някаква програма, вероятно няма да гледа в ръководството, докато не се измъчи достатъчно много и не му свърши търпението. Заради това вашата програма при общуването с потребителя трябва да създава дружелюбна атмосфера и да насочва действията на потребителя, даващи му инструкции, необходими за работа със системата. Но това съвсем не значи, че не трябва да се разработи добро ръководство, обясняващо, къде се използва програмата/системата и как се работи с нея. Дори отначало, ако го четат малко потребители, то в края на краища при използване на по-сложни функции на програмата/системата те ще са принудени да се обърнат към него. Следователно, ръководството е необходимо и то трябва да бъде такова, че с него да се работи лесно.

5. За разработване на системата да се използва група от програмисти.

Малка група от двама-трима програмисти обикновено може да създаде по-добра програма, отколкото един програмист, тъй като при работата им се обменят идеи между отделните членове на групата. Групата може да даде няколко решения на един и същи проблем. Това дава възможност да се избере най-добрия подход или решение на задачата. Другото преимущество на такъв стил на работа е, че в този случай е значително по-трудно да се съсредоточи върху една идея или на едно единствено решение, или да се вземе неправилно решение.

6. Да се отделя логическото от физическото.

Основната идея е много проста: когато пишете програма, разделяйте тези части, които вземат решение, от тези които управляват хардуера. Логическата секция на програмата не е длъжна да разбира как функционира апаратурата. Физическата секция отговаря за функционирането на отделни устройства.

1.3.3. Кодиране на програмата (Coding the Program)

Третият етап е етап на съставяне на програмата. Отделните блокове на блок-схемата се превръщат в поредица указания, които се наричат оператори от език, разбираем за съответния компютър. Програмистът транслира логиката от блок-схемата или от псевдокода, или от някой друг инструментариум, на един програмен език. Както вече отбелязахме, езикът за програмиране представлява установени правила които ви снабдяват с подходящ начин за инструктиране на компютъра какви операции да изпълнява. Има много езици за програмиране и като примери можем да посочим: BASIC, COBOL, Pascal, FORTRAN, C, C++.

Въпреки че, програмните езици оперират с граматика, до известна степен подобна на граматиката на английския език, те са много по-прецизни. За да може вашата програма да работи, вие трябва точно да спазвате правилата, синтаксисът, на езика който вие използвате. Разбира се, само коректното използване на езика не е гаранция, че вашата програма ще работи, така както говорейки граматически коректно на английски вие трябва да знаете и върху какво говорите. Коректното използване на езика е първостепенно задължение. След това вашата кодирана програма трябва да бъде въведена от клавиатурата, във форма, която компютъра може да разбере.

При писането на програма, програмистите обикновено използват текстов редактор, който донякъде прилича на текстообработващата програма Word, за да създадат един файл, който да съдържа текста на програмата. При това, като начинаещи, вие вероятно ще пожелаете първо да напишете вашия програмен код на хартиен носител.

1.3.4. Тестване на програмата (Testing the Program)

Четвъртият етап, проверка на готовата програма, е най-тежкият етап в процеса на създаването на програмата. Грешките, които могат да се допуснат в програмата, са два вида: синтактични и логически. Синтактичните грешки са следствие от неправилно използване на правилата на съответния програмен език. Тези грешки се откриват много лесно при транслирането на програмата (при превеждането им на

машинен език). Логическите грешки са трудно откриваеми, тъй като те са следствие от грешки в логиката на алгоритъма. През този етап програмата се изпълнява с примерни данни, за които са известни резултатите, които трябва да се получат. Целта е да се провери правилната работа на програмата във всички възможни случаи.

Евентуално, след като кодирате програмата, вие ще трябва да я тествате на компютъра. Тази стъпка включва следните фази:

- ***Проверка на редактирането***. Тази фаза, подобна на правене на коректури, понякога се анулира от автора на програмата който търси кратка процедура и гори от нетърпение да стартира програмата на компютъра, след като веднъж вече е написана. Обаче с внимателна редакционна проверка вие може да откриете няколко грешки и е възможно да спестите време от продължителната работа на програмата при нейното транслиране. При редакторската проверка вие просто мислено проследявате и проверявате логиката на програмата като се опитвате да я предпазите от грешки – да я направите безгрешна и изпълнима. Много организации използват допълнително в тази фаза една повторна проверка, един процес в който група от програмисти, ваши колеги, преглеждат вашата програма и предлагат съвети за подобряването й по един колегиален начин.
- ***Транслиране***. Транслаторът е програма която (1) проверява синтаксиса на вашата програма, за да се убеди че програмният език е използван коректно, дава ви съобщения за всички синтактични грешки, извършвайки диагностика, и (2) след това преработва вашата програма във формат, който компютърът може да разбере. Като страничен продукт на процеса е факта, че транслаторът ви съобщава, ако вие в отделни случаи сте използвали неправилно езикът за програмиране. Тези групи (типове) от грешки се наричат синтактически грешки (*syntax errors*). Транслаторът изработва съобщения описващи грешките (*error messages*[21]). Например ако на C++ вие неправилно запишете

```
N+2*(I+J));
```

където има две затварящи кръгли скоби вместо една – вие ще получите съобщение, което гласи:

```
error C2059: syntax error : ')'
```

Съобщението за грешка може да бъде с различна формулировка при различните транслатори. Програмите се превеждат обикновено от компилатор (*compiler*)[22]. Компилаторът превежда цялата програма на

---

[21] error message *същ*. съобщение за грешка = съобщение от системата или програмата, показващо, че е възникнала грешка, която трябва да бъде отстранена. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 282.

[22] compiler същ. компилатор = 1. Всяка програма, която преобразува един набор от символи в друг, като следва набор от синтактични и семантични правила. 2. Програма, която преобразува изходния код на програма, написана на език от високо ниво, в обектен

един път. Както е показана на фигура 1.2, транслирането включва вашата изходна (*original*) програма, наричана изходен модул (*source module*[23]), която се превръща с помощта на компилатора в обектен модул (*object module*[24]). Предварително написаните програми (*prewritten programs*) от системната библиотека (*system library*[25]) могат да бъдат добавени по време на фазата свързване/зареждане (*link*[26]/*load*[27]) , чиито резултат е изпълнимият (*load*) модул. След това изпълнимият модул може да бъде изпълнен от компютъра.

- ***Отстраняване на грешките в написаната програма*** (*debugging*[28]). Отстраняване на грешките в написаната програма (*debugging*,- в руската литература се използва термина *отладка*) е термин, който се използва широко в програмирането. Той означава откриването (*detecting*[29]), определяне на местоположението им (*locating*)

код преди изпълнението им. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 165

[23] source code същ. сорс код = програмни конструкции, написани на език от високо ниво или асемблерен език от програмист или разработчик, които немогат да бъдат прочетени директно от компютъра. Преди да може да бъде изпълнен от компютъра, сорс кодът трябва да бъде компилиран в обектен код. Английско-български тълковен компютърен речник Т.2 A-L, Софтпрес 2005, стр. 238

[24] object module *същ.* обектен модул = (в програмирането) версия с обектен (компилиран) код на файл със сорс код, който обикновено е сбор от процедури и е готов да бъде свързан с други обектни модули. Английско-български тълковен компютърен речник Т.2 A-L, Софтпрес 2005, стр. 75

[25] library същ. библиотека

1. В програмирането, колекция от процедури, съхранявани във файл. Всяка група от инструкции в дадена библиотека има име и всяка изпълнява различна задача.

2. Колекция от софтуерни файлове и файлове с данни. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 446

[26] link гл. свързвам

1. Създавам изпълнима програма от компилирани модули (програми, процедури или библиотеки), като сливам обектния код (обектен код на асемблерен език, изпълним машинен код или вариация на машинен код) на програмата и определям взаимните връзки (например функция на библиотека, извиквана от програма).

2. Свързвам два елемента в структура от данни, като използвам променливи за индекси или указатели. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 450

link същ. връзка, препратка. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 450

[27] load гл. зареждам = прехвърлям информация от място за съхранение в памет за обработка, ако това са данни, или за изпълнение, ако това е програмен код. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 453

[28] debug гл. дебъгвам = откривам и коригирам логически или синтактически грешки в програма, или неизправност в хардуер. При хардуера се използва по-често терминът отстраняване на проблем (troubleshoot), особено когато проблемът е голям. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 212

[29] detection *същ.* детекция =откриване на определено състояние, което оказва влияние върху компютърната система или върху данните, с които работи тя. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 221

и коригиране на грешките (*bugs*[30]), обикновено при стартиране (*running*) на програмата. Такива грешки са логически грешки, като например – програмна команда съобщаваща на компютъра да повтаря дадена операция многократно (да я „върти" в цикъл), но не му казва кога да излезе от това повторение (известно като „зацикляне на програмата"). В тази фаза вие изпълнявате програмата с тестови данни (*test data*[31]), които сами сте разработили. Тоест решили сте задачата ръчно с тези данни и сверявате крайните резултати. Вие трябва да планирате данните за теста много грижливо, за да сте сигурни, че се извършва тестване на всички разклонения на програмата.

Текстът на програмата се въвежда чрез текстов редактор и се записва в изходния модул (*source*). Транслира се с помощта на компилатора (*compiler*) в обектен модул (*object*), който представя програмата във форма, която свързващият редактор може да обработи. Компилаторът може да издаде диагностично съобщение (*diagnostic messages*), указващо синтактична грешка (*syntax errors*). Компилаторът може да отпечата изходната програма, нарича се още листинг/разпечатка (*listing*[32]) на програмата. След като програмата се

---

[30] bug *същ*. бъг, грешка

1. Грешка в кода или логиката на програмата, която води до неправилна работа на програмата или до генериране на неправилни резултати. Малки грешки, например неочакваното поведение на курсора, могат да бъдат неудобни или дразнещ, но не увреждат информацията. По-сериозните грешки могат да изискват потребителя да рестартира програмата или компютъра, като в този случай ще загуби всичката незаписана до този момент работа. Още по-лоши са грешките, които повреждат записаните данни, без да се уведоми потребителя. Всички тези грешки трябва да бъдат открити и коригирани от процес, познат като дебъгване. Заради потенциалния риск, грозящ важни данни, комерсиалните програми се тестват и дебъгват възможно най-пълно, преди да бъдат пуснати на пазара. След като програмата стане достъпна, допълнителните незначителни грешки се коригират в следващата актуализация. По-сериозните грешки могат понякога да се коригират със софтуер, наречен „кръпка", който избягва проблема или по някакъв друг начин смекчава ефектите му

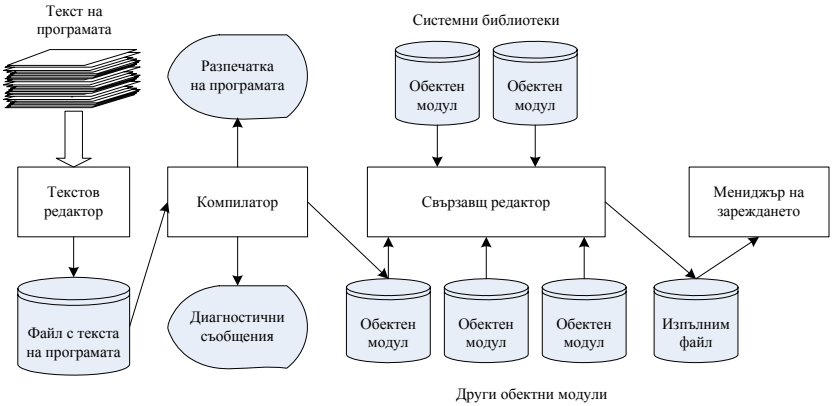
2. Повтарящ се физически проблем, който пречи на система или набор от компоненти да работят правилно заедно.

Макар че се водят спорове за произхода на тази дефиниция, компютърният фолклор определя първата употреба на думата bug (буквално насекомо, буболечка) от гледна точка на проблем, възникнал в резултат на молец, уловен между контактите на реле в машината (въпреки че етимологично молецът не е истинска буболечка). Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 108

[31] Test data *същ*. тестови данни = група стойности, използвани за тестване на това, дали дадена програма функционира правилно. Причините за избирането на определени тестови данни включва проверката на известен резултат (очакван резултат) и тестване на граничните условия, които могат да накарат програмата да работи. Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 275.

[32] Listing *същ*. листинг, разпечатка = отпечатано копие на сорс кода на програма. Някои компилатори и асемблери при желание извеждат листинг за асемблирането по време на компилация или асемблиране. Такива листинги с код често съдържат допълнителна информация, като номера на редовете, дълбочина на вложения блок и табица за справки. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 452.

компилира успешно, обектният модул се свързва във фазата свързване (*linking*) със системните библиотечни програми (*system library programs*) от които се нуждае и в резултат се получава изпълним файл, готов за зареждане (*load*) и изпълнение. Нарича се още и изпълнима програма (*executable program*).



Фигура 1.2. Подготвяне на програмата за изпълнение

1.3.5. Документиране на програмата (Documenting the Program)

Документирането е един постоянен, необходим процес, макар че както и многото програмисти, вие може би горите от желание да следвате по-увлекателната компютърна дейност - да пишете програми. Документацията е писмено, подробно описание на програмния цикъл и специфичните факти за програмата. Типичните програмни материали на документацията включват произхода и същността на проблема, сбито повествователно описание на програмата, описание с помощта на логически инструментални средства като блок-схеми и псевдокодове, описание на данните за запис (*data-record descriptions*), разпечатка на програмите (*program listings*) и резултати от тестванията (*testing results*). Коментарите в програмата, сами по себе си, се смятат за най-съществената част на документацията. Много програмисти документират когато кодират (по време на писането на програмата). В един по-широк смисъл, програмната документация може да бъде част от документацията за цялата система, както вие ще се запознаете, когато обсъждаме системния анализ (*systems analysis*) и проектирането (*design*).

Добрите програмисти продължават да извършват цялостно документиране на програмата при нейното проектиране, разработване и тестване. Документацията е необходима като притурка на човешката памет и помага да се организира планирането на програмата. Също така

документацията е съдбоносна по отношение на комуникацията с други хора, които проявяват интерес към програмата, особено за други програмисти, които могат да са част от програмния екип (*programming team*). Понеже темповете на развитие са най-високи в компютърната индустрия, то писането на документация е необходимо за тези, които идват след вас да правят някои необходими модификации в програмата, или да открият някои грешки, които вие сте допуснали.

Документацията трябва да съдържа: описание на предназначението на програмата, описание на алгоритъма, самата програма и ръководство на потребителя. Ръководството на потребителя пояснява как се работи и как се използва съответната програма. То трябва да бъде точно, ясно и да се съдържа указания за работа при всички възможни реакции на разработената програма.

1.4. Езици за програмиране

Езиците за програмиране са средство за описание на алгоритми. Описаният алгоритъм чрез дадения език за програмиране представлява програма, която се изпълнява от компютър. Компютърният програмен език представлява съвкупност от символи и правила за тяхното използване, предназначени за управление действията на компютъра. Всеки език притежава своя собствена граматика и синтаксис, свой собствен начин на използване.

От съществено значение е да се прави разлика между език за програмиране и неговата компютърна реализация. Самият език съдържа правила за записване, които определят синтаксиса на коректно написана програма (програма съставена на език от високо ниво или асемблерен език). Преобразуването на програмата от език на високо ниво в машинен език се осъществява от специални програми, наречени транслатори.

По принцип повечето задачи биха могли да се решат с помощта на произволен език за програмиране, но програмите биха изглеждали съвсем различно. На практика не е възможно познаването на всички езици за програмиране, но е важно да се знае принадлежността на определен език към някоя от обособилите се групи езици. Това налага единна класификация на програмните езици, което е доста сложно.

В настоящия момент съществуват над 200 езици за програмиране, които все още се използват. В случая не броим стотиците езици, които по една или друга причина са отпаднали през годините. Разумно е да се запитаме откъде и защо са дошли всички тези езици? Наистина необходимо ли е да се усложнява света по-нататък добавяйки езици за програмиране към Вавилонската кула на естествените (човешките) езици?

Първоначално, езиците за програмиране, са били създадени от хора работещи в университетите и в правителствени институти и учреждения като бяха разработени за специални цели. Някои езици имаха по-дълъг живот, доколкото те обслужват специални цели в науката, инженерството и др. подобни. Скоро става ясно, че трябва да се въведе някаква стандартизация. Проумява се, че за сходни задачи могат да се използват едни и същи езици.

Сега съществуват няколко езика с общо предназначение и ние ще обсъдим най-популярните по-късно в тази глава. Преди да започнем да разглеждаме специфичните езици, обаче, на нас ни е необходимо да ги подредим в някакъв ред, тоест да направим някаква класификация. Съществуват различни подходи. Ще се спрем на две класификации на езиците за програмиране:

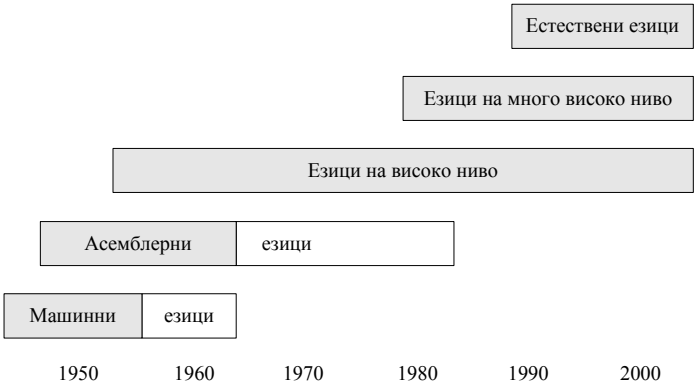
- спрямо близостта на езика до хардуера;
- от гледна точна на подходите за програмиране.

1.4.1. Класификация на езиците за програмиране от гледна точка близост до хардуера

Програмните езици могат да бъдат от по-ниско (*lower*) или по-високо (higher) ниво, в зависимост от това колко са близки до езика, който самия компютър ползва (0 и 1 ниво – ниско) или до езиците, които ползват хората (повече близки до английския език – високо). Приемаме че съществуват пет поколения компютърни езици, като от гледна точка на удобство на използване и потенциални възможности, всяко поколение е усъвършенствано спрямо предшестващото го. Петте поколения езици са:

Машинен език (machine language);
Асемблерни езици (assembly languages);
1. *Езици на високо ниво (high-level languages);*
2. *Езици на много високо ниво (very high-level languages);*
3. *Естествени езици (natural languages);*

Поколенията компютърни езици са дадени на фигура 1.3.



Щриховката показва периода на най-голяма употреба на поколението компютърен език. По-светлата част показва времето през което постепенно затихва неговото използването.

Фиг. 1.3. Развитие на поколенията компютърни езици във времето

Машинен език (Machine Language)

Хората не обичат да работят само с числа, а предпочитат да използват букви и думи. Но строго погледнато машинните езици боравят с битове и байтове. Машинният език, като най-ниско ниво на компютърните езици, представя данните и програмните инструкции като двоични числа (*binary digits*).

Асемблерни езици

Днес асемблерните езици се смятат за езици от много ниско ниво и поради тази причина те не са подходящи за програмиране, за разлика от многото съвременни езици. За времето, когато те се разработваха, обаче бяха считани за голям скок напред. За да се заменят единиците и нулите, използвани в машинните езици, асемблерните езици използват мнемонически кодове (*mnemonic codes*), абревиатурата на които е лесна за запомняне: ***A*** за събиране (*Add*), ***C*** за сравнение (*Compare*), ***MP*** за умножение (*Multiply*), ***STO*** за съхраняване на информацията в паметта и така нататък. Макар че тези кодове не са английски думи от гледище на човешкото удобство, те са все пак за предпочитане спрямо само числата 0 и 1. Нещо повече, асемблерските езици позволяват използването на имена (например ***RATE*** - стойност, цена или ***TOTAL*** - сбор, сума, цяло) за указване адреса за местоположението в паметта, вместо действителната цифра определяща адреса. Подобно на машинните езици, всеки тип компютър има собствен асемблерски език.

Програмистът, който програмира на асемблер, трябва да използва транслатор (*translator*), за да преобразува асемблерската програма в машинен език. Транслаторът е необходим, защото машинният език е единственият език, който компютърът може да изпълни. Транслаторът е асемблерска програма (написана на асемблер), която също се нарича асемблер (*assembler, assemble - събирам, монтирам, сглобявам*). Той взема програмата, която е написана на асемблерски език и я преработва в програма на машинен език. Програмистите не е необходимо да се безпокоят по отношение на машинния превод на програмата. Необходимо е само да пишат програмите на асемблерен език. Транслаторът има грижата да преведе програмата на машинен език.

Въпреки че асемблерните езици представляват стъпка напред, те имат много недостатъци. Ключов недостатък е това, че асемблерният език е детайлен до крайност, което прави асемблерското програмиране монотонно повтарящо се, скучно и предразполагащо към грешки.

Езици на високо ниво

Широкото използване на езици на високо ниво в началото на 60-те години на миналия век трансформира програмирането в нещо съвсем различно от това каквото е било. Програмите са написани по начин подобен на английския език и по този начин са направени по удобни за използване. Като резултат, един програмист може да реализира по-голям обем с по-малко усилия и сега може да се насочи към решаването на много по-сложни задачи.

Тези, така наречени езици от трето поколение (Таблица 1.1), подтикват към голямо увеличаване обработката на данни (*data processing*), което е характерно за периода на 60-те и 70-те години на миналия век. През този период броят на мейнфреймите, намиращи се в експлоатация, се увеличава от стотици на десетки хиляди. Влиянието на езиците от третото поколение върху развитието на обществото ни бе огромно.

Естествено е необходимо наличието на транслатор, за да се преведат символите с помощта на които се изразяваме в един език от високо ниво в компютърно изпълним машинен език. Този транслатор обикновено е компилатор. Съществуват много компилатори за всеки програмен език и за всеки тип компютър. Някои езици са създадени да обслужват специфични цели, като например управление на промишлени роботи или за създаване на графики. Други езици обаче, са извънредно гъвкави и се смята, че са с общо предназначение.

Езици на много високо ниво

Езиците, назовавани езици на много високо ниво (*very high-level languages*), са често известни по техния номер на езиковото поколение тоест те се наричат езици от четвърто поколение (*fourth-generation languages*) и в чуждестранната литература се срещат с опростената абревиатурата ***4GLs***. Няма консенсус за това какво представляват езиците от четвърто поколение. ***4GLs*** са по същество езици за стенографско програмиране. Едно действие/операция, което изисква стотици редове на език от трето поколение, като например ***COBOL***, обикновено изисква от пет до десет реда на език от четвърто поколение (***4GL***). Обаче постигнатият критерий за краткост на ***4GLs*** ги прави трудни за описание.

Таблица. 1.1. Популярни езици от трето поколение

Популярни езици от трето поколение	**Приложения**
FORTRAN – FORmula TRANslator (1954) (Транслатор/преводач на формули)	За нуждите на научните изследвания
COBOL – COmmon Business-Oriented Language (1959) (Език с общо предназначение ориентиран към бизнеса)	За нуждите на бизнеса
BASIC – Beginner's All purpose Symbolic Instruction Code (1965) (Символен код от инструкции за начинаещи с общо предназначение)	За нуждите на образованието и на бизнеса
Ada – кръстен на Ada, контеса на Lovelace (1980)	За целите на отбраната и с общо предназначение
Pascal – кръстен на Блез Паскал, френски философ, математик, физик писател. Автор на езика е швейцарският проф. Никлаус Уирт (1968)	Създаден е за нуждите на образованието
C – произхожда от езика B, на лабораторията Бел (Bell Labs) (1972)	Системен език с общо предназначение

Езиците от четвърто поколение имат няколко общи характеристики, които ги обединяват. Първата е че те прекъсват наистина заложената идея на развитие в предишните поколения езици – те по същество са непроцедурни езици. ***Процедурните езици*** използват процедури, също известни като подпрограми, методи или функции, съдържащи серия от изчислителни действия. Всяка процедура може да бъде извикана за изпълнение от всяка точка на главната програма, включително от други процедури или от самата себе си.

Първите три поколения езици са изцяло процедурни. В ***непроцедурните езици*** концепцията се изменя. Тук потребителят дефинира само това, което желае да направи компютърът; той не осигурява детайлите за това как точно да бъде направено. Очевидно много по-лесно и по-бързо е само да кажете какво вие желаете да направите, отколкото как да го направите. Това повишава многократно

производителността, която представлява ключова характеристика на езиците от четвърто поколение.

Легендите говорят, че езиците от четвърто поколение могат да повишат производителността от 5 до 50 пъти. Това е вярно. Повечето експерти заявяват, че средният показател на подобрение е около 10, тоест производителността се повишава 10 пъти, използвайки езици от четвърто поколение спрямо ефективността при работа с езици от трето поколение.

Не всички езици от четвърто поколение са с екстра качество, прелестни и с висока производителност. Те все още се разработват, и поради тази причина не могат да бъдат окончателно дефинирани и стандартизирани. Една от основните критики е тази, че при тези езици липсва необходимото управление и гъвкавост при планирането как да изглежда получавания резултат по ваше желание. Общото схващане за езиците от четвърто поколение е че те не правят ефективно използването на машинните ресурси, но предимството получавано от по-бързо завършване работата на една програма може далеч да превиши допълнителните експлоатационни разходи.

Обобщавайки казаното до тук за езиците от четвърто поколение, може да се каже, че те са полезни защото:

- са ориентирани към резултата, тоест те подчертават какво да стане вместо как да стане;
- повишават продуктивността при програмирането защото са лесни за писане и модифициране;
- могат да бъдат използвани с една минимална подготовка както и от програмисти, така и от непрограмисти;
- екранират потребителите от необходимостта да имат усещане за хардуера и програмните структури;

Вариант на езиците от четвърто поколение са ***езиците за запитвания*** (*query languages*[33]), които могат да бъдат използвани за придобиване на информация от бази от данни. Данните се добавят обикновено към базата от данни съобразно разработен проект (plan) с помощта на проектирани съобщения (planned reports), които могат да бъдат написани.

Но какво ще прави един потребител, който се нуждае от едно непроектирано предварително съобщение или от едно съобщение, което се различава по някакъв начин от стандартните съобщения?

---

[33] query language същ. език за заявки = подмножество на езика за обработка на данни; по конкретно тази част, която се отнася за извличането и визуализирането на данни от базата от данни. Понякога този термин се използва по-свободно за реферирание на целия език за обработка на данни. Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 158.

Той доста лесно може да научи един език на запитвания и след това ще може да въвежда запитване и да получава като резултат съобщения на неговия собствен терминал или персонален компютър.

Един стандартизиран език за запитвания, който може да се използва с няколко различни комерсиални програми за бази от данни, е Структурния език за запитвания (*Structured Query Language*[34]) популярно известен като ***SQL***. Други популярни езици за запитвания са Въпроси посредством примери (*Query-by-Example*[35]) известен като ***QBE*** и ***Intellect***.

Естествени езици (Natural Languages)

Думата естествен (*natural)* получи едва ли не такава популярност в компютърните кръгове каквато тя има в супермаркета. Езиците от пето поколение, както вие можете да се досетите, са дори по-лошо дефинирани от езиците от четвърто поколение. Те най-често се наричат естествени езици вследствие на тяхната прилика с „естествения" говорим английски език.

И за мениджъра, който отскоро се интересува от компютрите и който сега се насочва към тези езици, естествен означава подобен на човешкия език. Вместо да бъде принуден да въвежда с помощта на клавиатурата коректни команди и имена на данни в правилна последователност, мениджърът казва на компютъра какво да прави посредством написване с помощта на клавиатурата на собствени думи.

Мениджърът може да казва едни и същи неща по различни начини.

Например „Дайте ми сведение за парите от тенис ракети продадени през януари" работи също така добре както „Аз искам постъпленията от продажбите на тенис ракетите през януари".

Такива въпроси могат да съдържат грешно написани думи, липсващи членове и глаголи, да са употребени дори и жаргони.

Естественият език превежда инструкциите дадени от човека, с неправилна граматика, жаргон и други подобни неправилни неща, в

---

[34] structured query language *същ*. език за структурирани заявки = език за бази данни, използван за създаване на заявки, обновяване и управление на релационни бази от данни – де факто стандартът за бази данни. *Акроним* SQL.. Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 253.

[35] query by example *същ*. Заявка по пример = прост език за заявки, имплементиран в няколко системи за управление на релационни бази от данни. Използвйки заявка по пример, потребителят задава полетата, които да бъдат визуализирани, връзките за вмъкване и критериите за извличане директно във визуализирани на екрана формуляри. Тези формуляри са директно образно представяне на структурите от таблици и редове, съставящи базата данни. По този начин конструирането на една заявка се превръща в проста процедура от гледна точка на потребителя. *Акроним* QBE. Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 157.

код който компютъра разбира. Ако той не е сигурен, какво има предвид потребителя, задава вежливи въпроси за допълнителни пояснения.

Естествените езици понякога се отнасят към езиците, базирани на знания (*интелектуални езици - knowledge-based language*) понеже естествените езици се използват взаимодействайки с база от знания по някаква конкретна тема. Използването на един естествен език с достъп до база знания се нарича система базирана на знания (*knowledge-based system*).

1.4.2. Класификация на езиците за програмиране от гледна точна на подходите за програмиране

Програмите, написани на първите езици за програмиране са представлявали линейни последователности от елементарни операции с регистрите на процесора, където са се съхранявали данните. Тези езици са били оптимизирани към конкретния хардуер на който са се изпълнявали. Много често тези програми са били работоспособни само за процесора, за който са били разработени.

При езиците от „високо ниво" се постига увеличаване на производителността на програмистите в резултат на въведения подход за абстрахиране от конкретните детайли на хардуерната реализация. Една инструкция или оператор на езика от високо ниво включва последователност от няколко инструкции или команди на ниско ниво. Тъй като програмата по същество представлява набор от директиви, то методът за програмиране получава името *императивен*.

Повторното използване на предварително написани програмни блокове, чрез извикването им по име дава особена подредба и структурираност на програмата като цяло. Тези блокове се наричат процедури и функции.

Примери за езици от това ниво са ***FORTRAN, APL, BPL, C, ALGOL, COBOL, Pascal, Basic***.

През 60-те години възниква нов подход за програмиране – *декларативен*. Същността му е в това, че програмата вече не е последователност от команди, а описание на действия, които трябва да се извършат. Високата степен на абстракция позволява по-лесно да се тества програмата за наличие на грешки, а също така да се верифицира съответствието ѝ на техническото задание.

Един от пътищата за развитие на декларативния стил за програмиране става функционалният подход, възникнал след създаването на езика LISP. Други характерни примери за декларативни езици са SML, Haskell, Prolog.

При създаването на програми с помощта на функционалните езици програмистът се съсредоточава основно в областта на изследване и

в много малка степен се грижи за такива рутинни операции като разпределението на паметта, изчистване на паметта от ненужните впоследствие за програмата данни и т.н.

През 70-те години възниква клон на езиците за декларативно програмиране, така наречените езици за *логическо програмиране*, свързани с различни проекти в областта на изкуствения интелект. Програмите, разработени чрез логическия подход за програмиране, представляват съвкупност от правила и логически действия, допускат се логически причинно-следствени връзки.

Логическото програмиране се основава на класическата логика и се използва за логически изводи. Чрез езиците за логически изводи се създават експертни системи, ориентирани за подпомагане вземането на решение в различни сфери на човешката дейност. Като примери за такива езици могат да се посочат ***Prolog*** (*PROgramming in LOGic*) и Mercuri.

Важна стъпка в усъвършенстването на езиците за програмиране става появата на обектно ориентирания подход (ООП). Тук идеята е следната: програмата представлява описание на обекти, всеки обект има свойства или атрибути, класове, отношения между тях, способи за взаимодействие и операции с обектите или методите.

Използването на йерархия от класове на понятия от предметната област, дава възможност да се моделира реалния свят. Обектите, класовете и методите могат да бъдат полиморфни, което дава възможност за разработка на по-гъвкави и универсални програми.

Най-яркият представител на обектно ориентираните езици за програмиране е езикът C++, който представлява развитие на императивния език C. Пряк наследник и логическо продължение е езикът C#. Други подобни езици са ***Visual Basic, Eiffen, Oberon***.

С развитието на концепцията за управление на събитията през 90-те години се появи цял клас езици за програмиране, които се казват скриптови езици (scripting language; script - сценарий). Програмата, написана на такъв език представлява съвкупност от възможни сценарии за обработка на данни, като избора се интерпретира от сбъдването на едно или друго събитие. Например – щракване върху даден бутон, появата на курсора в определена зона, изменението на атрибутите на един или друг обект и т.н.

Характерни примери за скриптови езици са ***VBScript, PowerScript, LotusScript, JavaScript, Perl, PHP, Python***.

Съществено предимство на скриптовите езици е тяхната съвместимост с прогресивните инструментални средства за автоматизирано проектиране и бърза реализация на програмното осигуряване или както се наричат ***CASE*** (*Computer-Aided Software Engineering*) и ***RAD*** (*Rapid Application Development*) средства.

Още един важен клас езици за програмиране са езиците, поддържащи паралелни изчисления. Програмите, написани на тези езици

представляват съвкупност от описания на процеси, които могат да се изпълняват както едновременно така и псевдопаралелно. С паралелните изчисления се постига значително намаляване на времето за обработка особено на големи масиви от данни, постъпващи за едновременна обработка или при обработка на данни с висока интензивност – например видеоинформация или звук с високо качество и т.н. Друга област на използване са системите за реално време, в които потребителят трябва да получи отговор в много кратки срокове. Такива системи, обикновено обслужват жизнено важни обекти или се използват на места, където се вземат отговорни решения.

Като примери за езици, които поддържат паралелни изчисления могат да се посочат ***Ada***, ***Modula-2*** и ***Oz***.

Направената класификация не е единствената, тъй като критериите за класифициране могат да бъдат различни, а освен това езиците се развиват и усъвършенстват.

Като обобщение можем да изброим още веднъж използваните подходи в различните езици за програмиране:

- неструктуриран;
- структуриран или модулен;
- функционален;
- логически;
- обектно ориентиран;
- смесен;
- компонентно ориентиран (програмният проект се разглежда като множество от компоненти – използва се в ***.NET***);
- чисто обектен подход (от математическа гледна точка това е идеалният вариант, който все още не е реализиран на практика).

1.5. Начални сведения за езика C++

Бьорн Строуструп е разработил езика C++ и е създал първия му транслатор[36] Той е сътрудник на научно-изследователския изчислителен център AT&T Bell Laboratories[37] в Мюррей Хилл (Ню-Джерси, САЩ). Получил е званието магистър по математика и изчислителна техника в университета в град Аарус (Дания), а докторското звание по изчислителна техника в Кеймбриджкия университет (Англия). Строуструп е специализирал в областта на разпределените системи, операционните системи, моделирането и програмирането. Заедно с М. А. Еллис е автор на пълното ръководство на езика C++ - "Ръководство

[36] Бьорн Строуструп, създателят на C++ Програмният език C++, Том 1, ИнфоДАР, С 2001, стр. 576; Бьорн Строуструп, създателят на C++ Програмният език C++, Том 2, ИнфоДАР, С 2001, стр. 710

[37] За подробности виж в сайта: http://www.bell-labs.com/

за C++ с примери". При своята разработка той се е опирал на опита на създателите на езиците ***Симула***[38], ***Модула-2***[39] и абстрактни типове данни. Основните дейности са се извършвали в изследователския център на компанията Bell Labs.

Бьорн Строуструп е започнал разработката на езика през 1980 г., за да отговори на нуждите от език за симулационно моделиране с възможности за обектно-ориентирано програмиране – тогава относително нова идея в програмирането. Вместо да проектира език напълно отначало, д-р Страуструп решил да го изгради върху езика C[40].

Езикът C е разработен от Денис Ричи пак в Bell Laboratories през 1972 година, който го имплементира за пръв път в операционната система UNIX. Целта на Ричи е била да осигури език, който би позволил на програмистите достъп до хардуера почти такъв, какъвто има от асемблерните езици, но с конструкции на структурното програмиране, подобно на тези, които се намират в езиците от по-високо равнище. Наречен е по този начин, защото непосредственият му предшественик е езикът за програмиране B[41]. Впоследствие Деннис Ритчи като мениджър на неголяма изследователска група способства за разпространяване на операционни системи, езици за програмиране и съвременно компютърно оборудване. Достижения на тази група изследователи са и операционните системи PLAN 9 и Inferno.

Макар да се счита, че C е повече машинно-независим асемблерен език, отколкото език на високо ниво, тясната му връзка с операционната система UNIX, огромната му популярност и стандартизацията от организацията ANSI (American National Standards Institute) вероятно го приближават най-много до представата за стандартен език за програмиране на пазара за микрокомпютри/работни станции. C е компилаторен език съдържащ малък набор от вградени функции, които зависят от конкретната машина. Останалата част от функциите на C са машинно независими и се намират в библиотеки, до които може да се осъществи достъп от програми на C. Програмите на C

---

[38] SIMULA *същ. Съкр*. на **simu**lation **la**nguage. Език за програмиране с общо предназначение, базиран на ALGOL 60, със специални възможности, предназначен за подпомагане на описанието и симулация на активни процеси. Visual C++ е базиран на аспекти от този език. Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 226.

[39] Modula-2 *същ*. Модулен език от високо ниво, проектиран през 1980 г. от Никлаус Уирт. Modula-2, който произхожда от Pascal, е известен с подчертаното си модулно програмиране, ранната му поддръжка на абстрактни данни и липсата на стандартни функции и процедури. Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 36.

[40] Ал Стивънс, Уолнъм К., C++ библия, АлексСофт, С. 2000, стр. 912.

[41] „Преди C++ бе C, преди C бе B, преди B бе A. Преди A, бе мрак." Програмистки фолклор.

се състоят от една или повече функции, дефинирани от програмиста и поради тази причина е структурен език за програмиране.

Бьорн Строуструп добавя възможности към оригиналния език C, за да създаде така наречените от него „C за обектно-ориентирано програмиране". Тези класове дефинират програмни обекти със специфични възможности, които трансформират процедурната същност на C в обектно-ориентирания език за програмиране C++.

През 1997 година е приет международен стандарт за C, в който фактически са обобщени изводите от неговото 20-годишно развитие. Приетият стандарт осигури съгласуваност на всички реализации на езика C++. Не по-малко важен резултат от стандартизацията е фактът, че в процеса на изработването и утвърждаването на стандарта на езика са уточнени и допълнени редица съществени възможности.

Стандартната версия на C++ бе дефинирана от съвместен комитет на Американския национален институт по стандартите (American National Standards Institute - ANSI) и Индустриалната организация за стандартизация (Industry Organization for Standardization - ISO). Тази версия е известна като ANSI C++ и е преносима на всяка платформа и всяка среда за разработка. Неговият номер е ISO/IEC 14882.

Езикът C++ е универсален език за програмиране, за който допълнително са разработени комплект разнообразни библиотеки. По тази причина, строго погледнато, той осигурява възможност практически да се реши която и да е задача за програмиране. В сила са обаче различни причини (не винаги технически) и затова той се употребява за едни типове задачи по-често, а за други по-рядко.

C++ като приемник на езика C се използва широко в системното програмиране. С него могат да се пишат високоефективни програми, в това число операционни системи, драйвери и т.н. Езикът C++ е един от основните езици за разработване на транслатори.

Доколкото системното програмно осигуряване е написано често на езика C или C++, то и програмните интерфейси към подсистемите на ОС често също се пишат на C++. Съответно, тези програми, даже и приложните, които взаимодействуват с операционните системи, са написан на езика C++.

Разпределените системи, функциониращи на различни компютри, също се разработват на езика C++. За това способства факта, че в широко разпространените компоненти на моделите CORBA и COM има удобни интерфейси на езика C++.

Обработката на сложни структури от данни - текст, бизнес-информация, Интернет страници и т.н. - са една от най-разпространените възможности за използване на езика. В приложното програмиране е навярно по-лесно да се изброят тези области, където езикът C++ се използва малко.

Графичен потребителски интерфейс се разработва на езика C++ в случаите когато е необходимо да се разработват сложни, нестандартни интерфейси. Простите програми най-често се пишат на езиците ***Visual Basic***[42], ***Java***[43] и т.н.

Програмирането за Internet се извършва основно с езиците: ***Java***, ***VBScript***[44], ***Perl***[45].

Обобщавайки, може да се каже, че езикът C++ днес е един от най разпространените езици за програмиране в света.

Най-кратката програма на езика C++

Най-кратката програма на езика C++ изглежда по следния начин:

---

[42] Visual Basic *същ. Търговска* марка на Microsoft Corporation за версия на езика за програмиране Basic за визуално програмиране на високо ниво. Visual Basic е проектиран за изграждане на Windows базирани приложения. Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 330.

[43] Java *същ.* Обектно-ориентиран език за програмиране, разработен от Sun Microsystems, Inc. Подобен на C++, Java е по-малък, по-преносим и се използва по-лесно от C++, защото е по-праволинеен и сам управлява паметта. Java е проектиран да бъде сигурен и платформено неутрален (което означава, че може да се изпълнява на всяка платформа) поради факта, че Java програмите се компилират в байткод, който не е обработен до точката на зависимост от специфични за платформата инструкции и се изпълнява на компютър с помощта на специална софтуерна среда, известна като виртуална машина. Тези характеристики на Java го правят полезен език за програмиране на уеб приложения, тъй като потребителите имат достъп до Интернет от различни видове компютри. Java се използва за програмиране на малки приложения или аплети за World Wide Web, както и за създаване на разпределени мрежови приложения. Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 423.

[44] VBScript = Visual Basic, Scripting Edition *същ.* Подмножество на езика за програмиране Visual Basic, оптимизирано за уеб-ориентирано програмиране. Както при Java Script, кодът на Visual Basic, Scripting Edition се влага в HTML документа. Тази версия е включена в уеб браузера Internet Explorer. Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 330.

[45] Perl *същ.* Акроним на **P**ractical **E**xtraction and **R**eport **L**anguage. Интерпретаторен език, базиран на C и няколко UNIX помощни програми. Perl има мощни възможности за обработка на низове за извличане на информация от текстови файлове. Perl може да сглоби някакъв низ и да го изпрати на шела като команда; затова често се използва при задачи свързани със системната администрация. Програма, написана на Perl, се нарича скрипт. Perl е създаден от Лари Уол в лабораторията Jet Propulsion Laboratory на НАСА. Английско-български тълковен компютърен речник Т.2 M-Z, Софтпрес 2005, стр. 110.

```
//Най-кратката програма
int main()
{
   return 0;
}
```

Първият ред на програмата е коментар[46], който служи само за пояснения. Признак за коментар са двете една до друга наклонени черти (//).

В програмите на C++, конструкциите, които трябва да бъдат изпълнени, се поставят във „функции". Кодът дефиниращ функция се нарича „декларация" на функция и винаги има следния синтаксис:

```
тип- данни име-на-функция ( )
{
изпълними-конструкции
}
```

main – това е име на главната функция на програмата. Изпълнението на програмата започва винаги с функцията main. Функцията има име ***main***, а след името в кръгли скоби се изброяват аргументите или параметрите на функцията (в дадения случай функцията main няма аргументи). От функцията може да се получи резултат. Ако функцията не връща нищо, то това се означава с ключовата дума ***void***. Във фигурните скоби се записва тялото на функцията – действия, които тя изпълнява. Операторът

```
return 0;
```

означава, че функцията връща резултат – цяло число 0.

След като дадена функция бъде извикана, за да изпълни съдържащите се в нея конструкции, тя връща стойност на извикващия. Тази стойност може да бъде единствено от типа данни, зададени в декларацията на функцията.

Една програма може да съдържа една или много функции, но винаги трябва да има функция с името ***main***. Функцията ***main*** е началната точка на всички програми на C++ и компилаторът няма да компилира кода, ако не намери функция с това име в програмата.

На другите функции в програмата можете да зададете каквито имена желаете, стига да отговарят на изискванията на синтаксиса на езика за въвеждането на идентификатор, които ще разгледаме впоследствие.

---

[46] Едноредов коментар – започва с две наклонени черти (//) и завършва в края на реда. Едноредовият коментар не използва символ за края на коментара, той завършва с физическия край на реда.Обикновено програмистите на C++ използват стандартния C механизъм за коментар (/* ... */), състоящ се от много редове, а едноредовия коментар използват за кратки съобщения

Фигурните скоби съдържат конструкциите, които трябва да бъдат изпълнени при извикване на функцията. Всяка конструкция трябва да завършва с точка и запетая, така като всяко изречение завършва с точка.

Ако говорим за обектно-ориентирана програма, то тя трябва да създаде обект от някакъв клас и да му изпрати съобщение. За да не се усложнява програмата, ще използваме един от готовите, предварително известните класове – класа ***ostream*** (поток за въвеждане и извеждане). Този клас е определен от файла в заглавието iostream.h[47]. Затова първото, което трябва да се направи е да се включи заглавния файл в програмата ни:

```
#include <iostream.h>
int main( )
{
   return 0;
}
```

Както виждаме кодът на програмата започва с инструкция към компилатора, която го кара да включи (***include***) информацията от стандартната входно-изходна библиотека ***iostream*** (input –output – stream – входно-изходен-поток). По-правилно е тази инструкция да се нарича „предпроцесорна" инструкция и тя трябва да бъде представяна в началото на програмата, преди действителния и код. Редът започва със знака диез (#), което указва на компилатора, че това е предпроцесорна инструкция. Името на библиотеката трябва да бъде поставено между отваряща и затваряща ъглови скоби (< и >).

В декларацията на функцията типът данни е зададен като ***int***, което означава цяло число (***integer***). Това означава, че след като изпълни конструкциите, тази функция трябва да върне целочислена стойност на операционната система.

Освен класа, заглавният файл определя глобалния обект на този клас ***cout***. Обектът се нарича глобален, доколкото достъпът до него е възможен от произволна част на програмата. Този обект изпълнява извеждането на конзолата. Във функцията ***main*** можем да се обърнем към него и да му изпратим съобщение, което ще направим в христоматийния пример за първа програма на C++:

```
#include <iostream.h>
int main()
{
   cout << "Hello world!" << endl;
   return 0;
}
```

[47] Системните заглавни файлове на програмната среда Visual C++ 6.0 не отговарят напълно на стандарта ANSI C++ и затова са с разширение .h.

Операцията за отместването << за класа ***ostream*** е определена като „изведи”. По такъв начин, програмата изпраща на обекта ***cout*** съобщения „изведи текстовия низ Hello world!” и „изведи преместване на реда” (***endl*** означава нов ред). В отговор на тези съобщения обектът cout извежда текстовия низ „Hello world!” на конзолата (монитора) и преместват курсора на следващия ред.

Текстовите низове в C++ се поставят винаги в кавички. Последната конструкция от функцията main използва ключовата дума return, за да върне стойност нула на операционната система. По традиция връщането на нулева стойност след изпълнението на програма информира операционната система, че програмата е изпълнена правилно. В резултат на изпълнението на програмата на конзолата (в случая на монитора) се появява следната информация:

```
Hello world!
```

Компилиране и изпълнение на програма

Някои работни среди на C++ са много удобни за използване. Просто пишем текста на програмата в един прозорец, натискаме бутон или меню, за да компилираме и друг бутон или меню – за да стартираме кода му. Съобщенията за грешка се появяват във втори прозорец, а резултатът от изпълнението на програмата се изобразява в трети. В такава среда ние сме напълно освободени от подробности по компилирането. При други системи трябва да се извършва всяка стъпка ръчно.

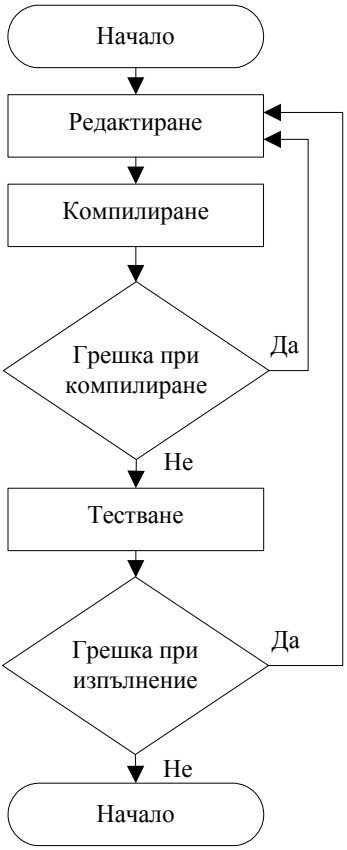
Дори ако използваме удобна среда за програмиране на C++, е добре да се знае какво точно става, защото познаването на процеса ни позволява да се справяме с възникнали проблеми.

Програмите на езика C++ първоначално се създават като обикновени текстови файлове, които обикновено се записват с файлово разширение ***cpp***[48]. С помощта на произволен текстов редактор програмистът записва инструкции, в съответствие с които компютърът ще работи, изпълнявайки конкретната програма. Те могат да бъдат написани и с произволен текстов реактор, като например Notepad в Windows или EMACS в Linux, т.е. не е необходим никакъв специализиран софтуер. Така се получава изходния (*source*) код на C++.

След това се влиза в цикъла редактиране – компилиране – настройка (фиг. 1.4). За да може компютърът да изпълни програма, написана на езика C++, е необходимо тя да се преведе на езика на машинните инструкции. Тази задача се решава от компилатора. Компилаторът чете файла с текста на написаната програма, анализира я, проверява я за евентуални грешки и, ако не открие такива, създава така

[48] cpp – Това е абревиатура на C++.

наречения обектен код. Обектният код се състои от машинни инструкции и информация за това как да се зареди програмата в паметта, преди да започне изпълнението ѝ. Обектният код е в отделен файл, обикновено с разширение ***.obj*** или ***.o***. Например обектният код за програмата ***test*** се записва в ***test.obj***.



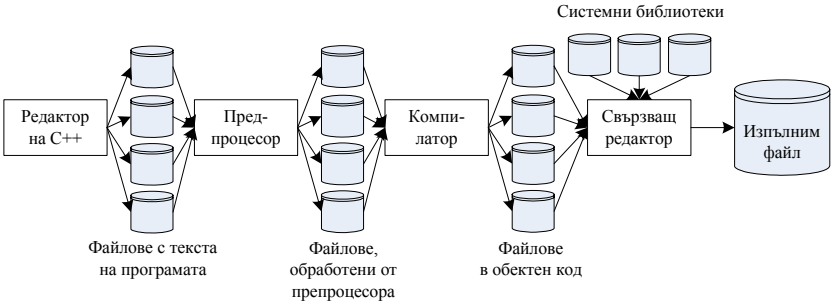
Фиг 1.4. Цикъл „редактиране - компилиране – настройка"

Обектният файл съдържа само „превода" на програмата, която сме написали. Това обаче не е достатъчно, за да се изпълни. За да се покаже низ[49] на екрана, са нужни доста операции на ниско ниво. Авторите на пакета ***iostream*** (който дефинира ***cout***) са описали всички необходими действия и са поставили нужния машинен код в библиотека. Тя е съвкупност от код, който е написан и транслиран от някой друг и е готов за използване от нас.

По-сложните програми са съставени от повече от един обектен файл и повече от една библиотека. Специализирана програма, наречена свързваща програма (*linker*), взема обектния файл и необходимите части от библиотеката на ***iostream*** и създава изпълним файл, т.е. файл с машинни инструкции в байт код, които могат да се изпълняват (фиг. 1.5).

Компилирайки програмата еднократно, същата се транслира във втори файл с машинно четивен изпълним формат, който може да се изпълнява многократно, с различни входни данни.

[49] Понятието низ в момента го възприемете интуитивно като последователност от символи (например „Hello world!"). Впоследствие във втора част на учебника ще бъде задълбочено изяснено като съставни типове данни.



Фигура 1.5. Етапи на разработка на една програма на C++

Ако използвате персонален компютър с операционна система Windows 98, Windows NT, Windows 2000 или Windows XP, то компилаторът с който разполагате най-вероятно е Visual C++. Този компилатор представлява интегрирана среда за програмиране (IDE[50]), тоест обединява текстов редактор, компилатор, дебъгер и още редица допълнителни програми.

1.6. Пример за програма на C++

Да решим следната задача: Да се напише програма, която въвежда размерите на правоъгълник и намира периметъра и лицето му. Кодът на програмата е даден на фиг. 1.6, а резултатът – на фиг. 1.13.

```
//Изчисляване на периметъра и лицето на правоъгълник
//по зададени размери
#include <iostream.h>
int main ()
{
    cout <<"Изчисляване на параметъра и лицето на правоъгълник по зададени дължини на двете му страни.\n";
    //въвеждаме първата страна
    cout <<"Въведете дължината на първата страна: ";
    double a;
    cin >> a;
    //въвеждаме втората страна
```

[50] Integrated development environment *същ.* интегрирана среда за разработка = интегрирани инструменти за разработване на софтуер. Инструментите по принцип се изпълняват от един потребителски интерфейс и се състоят от компилатор, редактор и дебъгер, както и други инструменти. *Акроним* IDE. . Английско-български тълковен компютърен речник Т.1 A-L, Софтпрес 2005, стр. 399.

```
        cout << "Въведете дължината на втората страна: ";
        double b;
        cin >>b;
        //намиране на периметъра
        double p = 2*(a+b);
        //извеждане на периметъра
        cout <<"Периметърът на правоъгълника със страни " << a <<
        " и "<< b <<" е "<< p <<'\n';
        //намиране на лицето
        double s = a*b;
        //извеждане на лицето
        cout <<"Лицето на правоъгълника със страни с дължина "
        << a << " и "<< b <<" е "<< s <<'\n';
        return 0;
}
```

Фиг. 1.6. Код на програма за намиране лице и периметър на правоъгълник по зададени дължините на двете му страни

Програмата започва с два коментарни реда:

```
//Изчисляване на периметъра и лицето на правоъгълник
//по зададени размери
```

Добрата програмистка практика налага добавяне на коментари към кода на C++. Те правят кода по-лесен за разбиране от други хора, а дори и от вас при преглед на част от кода след известно време.

Всички коментари на един ред, предшествани от двойка наклонени черти //, се игнорират от компилатора на C++.

Коментари, поставени между /* и */, също се игнорират от компилатора, дори когато заемат няколко реда. Синтаксисът за коментари /*.....*/ е наследен от езика C.

Повечето комерсиални програми на C++ започват с блок от коментари, описващи предназначението на програмата, както и друга информация, например името на автора, датата на създаване на програмата, разрешения за авторски права и т.н.

Чрез коментари може да описваме и предназначението на отделни функции и конструкции. В примера сме „прекалили” с коментарите, което е направено единствено с учебна цел.

Кодът на програмата започва с инструкция към компилатора, която го кара да включи (*include*) информация от стандартната входно-изходна библиотека iostream. По правилно е тази инструкция да се нарича „предпроцесорна” инструкция и тя трябва да бъде поставена в началото на програмата, преди действителния и код. Редът започва със знак диез (#), който обозначава предпроцесорна инструкция. Името на

библиотеката трябва да бъде представено между отварящи се и затварящи се ъглови скоби (< и>):

```
#include <iostream.h>
```

В по-новите компилатори се поддържа пространство на имената. Същото се задава с реда:

```
using namespace std;
```

Той казва на компилатора, че всички имена, които използваме в програмата принадлежат към „стандартното пространство на имената". В големите програми често се случва различни програмисти да означават различни неща с едно и също име. Те могат да избягнат така възникващите проблеми, като използват различни пространства на имената. Но за обикновените програми, които ще пишем в този курс, не са необходими отделни пространства. Винаги ще използваме обичайните пространства на имената, така че можем автоматично да добавяме директивата

```
using namespace std;
```

в началото на всяка програма. Пространството на имената е сравнително нова възможност на C++[51] и компилаторът на програмна среда ***Visual C++ 6.0***, с която работим, не го поддържа. По тази причина няма и да декларираме споменатата директива.

Както вече казахме една програма може да съдържа една или няколко функции, но винаги трябва да има функция с името „***main***", която е начална точка на всички програми на C++ и от нея започва компилацията. В примера имаме само една функция и тя естествено е декларирана като ***main***.

```
int main ( )
```

В декларацията на функцията типът данни е зададен като int, което означава цяло число (*integer*). Това означава, че след като изпълни конструкциите, тази функция трябва да върне целочислена стойност на операционната система.

Обикновените скоби след името на функцията main са празни, понеже не се задават аргумент. Те само указват на компилатора, че main е име на функция.

Фигурните скоби { } съдържат конструкциите, които трябва да бъдат изпълнени от програмата. Първата конструкция:

```
cout <<"Изчисляване на периметъра и лицето на правоъгълник по зададени дължини на двете му страни.\n";
```

извиква функцията ***cout***, която е дефинирана в стандартната входно-изходна библиотека ***iostream***. Операторът << указва, че

[51] Втората версия на C++ е създадена от Строуструп и стандартизационния комитет ANSII/ISO : Standard C++ и са включени две нови възможности: нов стил за включване на заглавни файлове и така нареченото именовано пространство (name space)

функцията cout трябва да изведе текстов низ в стандартен изход. Необходимо е да отбележим, че при програмиране на C++ низовете винаги трябва да бъдат поставени в кавички. Този низ съдържа текста и контролния знак \n[52] за нов ред. Този знак премества печатащата глава[53] в най-лява позиция на следващия реда. Реално това се разпечатва на първия ред на екрана за изход и е даден на фиг. 1.13.

Следващият ред (на редовете с коментари няма да се спираме повече) извежда подсказваща информация за въвеждане дължината на първата страна:

```
cout <<" Въведете дължината на първата страна: ";
```

Разликата от първата команда ***cout*** е тази, че в края на стринга не е въведен контролния знак ***\n*** за нов ред. При изобразяването на екрана маркерът остава на същия ред с подсказка за мястото където ще се въведе стойността.

След това трябва да въведем дължината на първата страна, която сме обозначили в програмата с променливата a. Преди да се въведе дадена стойност е необходимо да се декларира от какъв тип е променливата. Това е от съществено значение. Типът на променливата определя ареала от стойности, които може да заема и действията, който могат да се извършват с нея. На това сме посветили цялата следваща трета глава. Досега сме използвали само целочисления тип - ***int***, за да укажем типа на променливата, която функцията main връща на операционната система.

Приемаме, че дължините на страните ще задаваме като реални числа, които се обозначават с променлива от тип ***double***. Декларацията на типа на променливата a извършваме с:

```
double a;
```

При тази декларация в оперативната памет на компютъра се резервират определено количество байтове, които за програмна среда Visual C++ 6.0 е осем байта. Целесъобразно е, още тук, да се спрем малко по-подробно на архитектурата на оперативната памет, която естествено ще разгледаме с определени условности.

Разпределението на оперативната памет зависи от много фактори (типа на изчислителната и операционната система, от модела памет и др.). Най-общо се състои от области за: операционната система, програмния код, статичните данни, динамичните данни и програмния стек (фиг. 1.7.).

---

[52] \n по въздействие е еквивалентен на endl

[53] Този термин е останал от времето когато информацията се е извеждала с помощта на принтер. При монитора това съответства на преместване на маркера на нов ред.

Фигура 1.7. Области на разпределение на оперативната памет

Програмен код

Тук се записват всички функции, които изграждат потребителската програма.

Област на статичните данни

В нея са записани глобални обекти (в широкия смисъл на думата) на програмата.

Област на динамичните данни

За реализиране на динамични структури от данни (списъци, дървета, графи, ...) се използват средства за динамично разпределение на паметта Чрез тях се заделя и освобождава памет в процеса на изпълнение на програмата, а не преди това (при компилирането й).

Програмен стек

Съхранява данните на функциите на програмата. Стекът е динамична структура, организирано по правилото „последен влязъл – пръв излязъл". Той е редица от елементи, с пряк достъп до елемента от единия си край (в случая долния), наречен връх. Достъпът се реализира чрез указател[54]. Операцията включване се осъществява само пред елемент от върха, а операцията изключване – само за елемент от върха.

При деклариране на реалната променлива ***a***:

```
double a;
```

се заделят 8 байта в дъното на програмния стек и променливата a се свързва с адреса на първия байт (0x0012FF7C[55]) от заделените осем байта (фиг. 1.8).

Обектът cin се използва за генериране на вход от потребителя. Приемаме, че стойността на първата страна е 5 единици. Нанасяме в екрана на мястото посочено от маркера стойността 5:

```
Въведете дължината на първата страна: 5
```

[54] Указателят, като спецефичен тип скаларни данни, ще бъде разгледан в трета част на учебника.

[55] Адресът е конкретен. В случая това е адреса, където се записва първия байт на променливата в оперативната памет на компютъра за един от авторите на учебника.

Фигура 1.8. Организация на оперативната памет при решаване на задачата за намиране лицето и обиколката на правоъгълника при деклариране типа на променливата ***a***

При това положение 5 се намира в буфера на клавиатурата. Натискайки клавиша Enter стойността се пренася съгласно оператора за въвеждане >>:

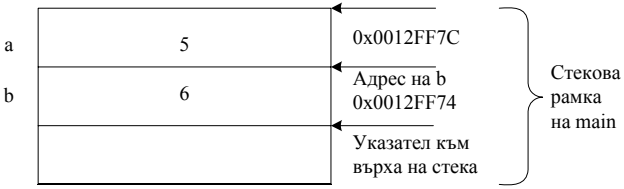
```
cin >> a;
```

и се записва в стека на програмата main, на заделените за това байтове за променливата а (виж фиг. 1.9).

Фигура 1.9. Въведената стойност на ***a*** в стека на ***main***

Със следващите три реда от програмата се резервира място за ***b*** и се въвежда стойността - в случая 6 (виж фиг. 1.10):

```
cout << " Въведете дължината на втората страна : ";
double b;
cin >>b;
```



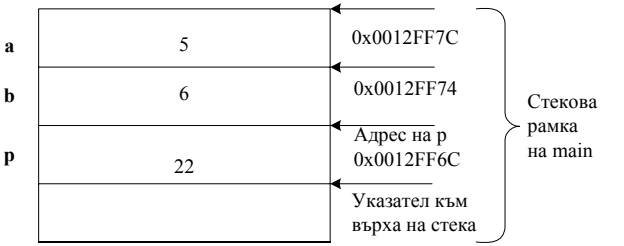
Фигура 1.10. Въведени стойности на ***a*** и ***b*** в стека на ***main***

С помощта на следващия ред:

```
double p = 2*(a+b);
```

се декларира променливата p, като реална променлива, заделят се и за нея 8 байта и след това се инициализира с резултата получен в дясната част.

Знакът = означава да се присвои на променливата стояща в ляво (в случая **p**) стойността получена от извършените изчисления в дясно. Програмата взема съответните стойности за ***a*** и ***b*** от стека на ***main*** на оперативната памет и извършва изчислението съгласно правилата на математиката (тук разликата е само, че * е знак за умножение) – виж фиг. 1.11.



Фигура 1.11. Стекът на ***main*** след изчисляването на периметъра на правоъгълника

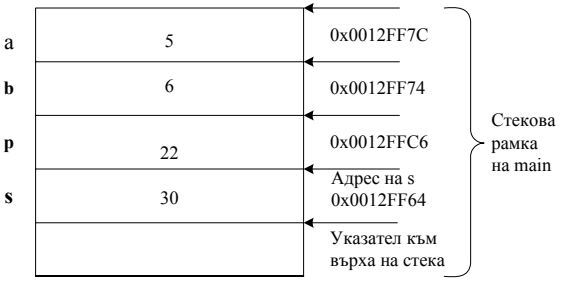
Извеждането на получената стойност за параметъра на правоъгълника се получава с изпълнението на следните програмни редове:

```
//извеждане на периметъра
cout <<"Периметърът на правоъгълника със страни " << a <<
" и " << b <<" е "<< p <<'\n';
```

Подробностите по извеждането ще бъдат разгледани впоследствие допълнително.

Аналогично при изпълнение на следните редове от програмата се изчислява лицето на правоъгълника и се извеждат стойностите му (фиг. 1.12):

```
//намиране на лицето
double s = a*b;
//извеждане на лицето
cout <<"Лицето на правоъгълника със страни с дължина "
<< a << " и "<< b <<" е "<< s <<'\n';
```



Фигура 1.12. Стекът на ***main*** след изчисляването на лицето на правоъгълника

```
Изчисляване на периметъра и лицето на правоъгълник по зададени дължини на двете му страни.
Въведете дължината на първата страна: 5
Въведете дължината на втората страна: 6
Периметърът на правоъгълника със страни 5 и 6 е 22
Лицето на правоъгълника със страни с дължина 5 и 6 е 30
```

Фигура 1.13. Резултатът от решаването на програмата, чийто код е даден на фиг. 1.6.

Последната конструкция във функцията ***main*** използва ключовата дума return, за да върне стойност нула на операционната система:

```
return 0;
```

По традиция връщането на нулева стойност след изпълнение на програмата информира операционната система, че програмата е изпълнена правилно. return изтрива и съдържанието на стековата рамка на main.

С тези обяснения би трябвало да добиете представа за съдържанието и изпълнението на една програма написана на езика C++.

За да можем да запуснем програмата на компютъра е необходимо да се запознаем с програмната среда Visual C++ 6.0, което и ще направим в следващата глава.