Увод в програмирането

Основни информационни дейности. Средства за автоматизация на смятането

1. Основни информационни дейности.

- ✓ Събиране на информация
- ✓ Съхранение
- ✓ Обработка
- ✓ Разпространение

С появата на писмеността, получаваме възможност за съхранението на по-голяма информация. Това противоречи на ограничената възможност и за това се създават средства за автоматизирана обработка:

- ✓ Инструмент + метод;
- ✓ Сметачна машина – първата машина е създадена от Блез Паскал ~ 1640г. В началото те биват механични, след това се доразвиват в ел. механични, а сега електронни (калкулатори);
- ✓ Автоматична сметачна машина – първия проект е създал и реализирал Чарлз Бабедж;

Програмите са фиксирани на външен носител (перфокарти). Джон фон Нойман премества програмата в паметта, което позволява и тя да се обработва, така както и информацията.

Сметачните и автоматично сметачните машини могат да бъдат аналогови и цифрови. Разликата между тях е в начина на представяне на числата. В аналоговите сметачни машини, числата се представят чрез физична величина (температура, налягане, ток, напрежение), а при цифровите чрез представяне на цифри, а всяка цифра отговаря на положението на даден физически елемент. Състоянията на елемента могат да се различават по краен брой, рязко разграничаващи се по между си състояния.
Например: 0 – няма напрежение, 1 – има напрежение.

Правени са експерименти в Русия и Италия с троична бройна система, т.е. с три състояния на физическия елемент. В Русия тя е била симетрична – (-1), 0, 1. Тези експерименти не успяват като практическа реализация.

Съвременните компютри се наричат Нойманови. Принципите в устройството на машините не се променят, а прогреса се отнася до техническата реализация в едни и същи граници.

В началото компютрите са били изградени от релета като основен физически елемент. (наричат се още компютри нулево поколение). През 1944г. В САЩ е осъществен първия производствен модел.

2. **Първо поколение компютри** - компютри с основен физически елемент – ел. лампи.

Наричат се още лампови компютри. Техен създател е Джон Атанасов. С това започва промишленото производство на компютрите. Основен стимул е била Втората световна война. Създавали са се за обслужването на ракети и атомни бомби. Те изисквали много пресмятания, което пък е наложило автоматизацията на изчисленията. В Русия - липсата на автоматизирани сметачни машини се компенсира с напреднали методи на изчисляване и обикновени сметачни машини.

3. **Транзисторни** – второ поколение компютри

4. **Трето поколение компютри** – с интегрални схеми.започват да се решават бизнес-приложения и намаляват изчислителните задачи.

5. **Четвърто поколение** – големи интегрални схеми.

Масивна рекламна кампания за революционния подход на новите машини. Всъщност принципа на работа е същия, а промяната се крие в намаляването на размера на компютъра, улесняване на условията, нужни за съхранението и обслужването му, падането на цената , все по-големия брой производители и ползватели. Навлиза термина „персонален компютър". Има напредък и в начина на съхраняване на информацията.

6. **Пето поколение компютри**.

Целта при тях е да не са Нойманови, т.е.да се засегне и промени принципа на работа. Този процес все още тече Започва се използването на свръх-големи компютри. Увеличава се скоростта на работа и обема на съхранение на информацията.

Тенденции:

- ✓ Създаване на мощен компютър се състои в създаването на един мощен процесор и голям RAM – в миналото
- ✓ В момента тече процес по увеличаване на броя на процесорите или обединяването на много компютри във ферми.

Важни дати:

1959г. – първа специалност по изчислителна математика във ФМИ – СУ. Това е свързано с разработката на числени методи- зачатък на информатиката.
1961г.- първи изчислителен център към БАН
1962г. - 1963г. – първия компютър (български) „Витоша" с програмно управление.
1964г. – първи ел. калкулатор „Елка"; Получава се първия съветски компютър „Минск-2".

Дефиниция:
Компютър – техническо средство за автоматизирано изпълнение на алгоритми, описани на машинен език.

Алгоритми – определение, примери, свойства, начини на изразяване (словесно, блок-схеми, алгоритмични езици)

1. **Определение** – списък от краен брой правила за извършването на действия в даден ред, които решават всяка задача от даден тип, наричаме алгоритъм.

2. **Пример за алгоритъм:**

Нека имаме редицата a_1 a_2 ...$a_к$ ... a_n. Означава ме с α най-малкото, а с β най-голямото от тях.

1) Въведи n, a_1 a_2 ... a_n – премини към т.2
2) $\alpha = a_1$, - премини към т.3
3) $\beta = a_1$, - премини към т.4
4) k=2, премини към т.5
5) Ако $a_k<\alpha$, премини към т.6; ако не – премини към т.7
6) $\alpha=a_к$, изпълни т.9
7) Ако $a_к>\beta$ изпълни т.8; иначе премини към т.9
8) $\beta=a_к$, изпълни т.9
9) k=k+1. Изпълни т.10
10) Ако k<=n, премини към т.5. Ако не – премини към т.11
11) Изведи α и β, премини към т.12
12) Край на програмата

Действията в един алгоритъм могат да бъдат два вида: условни и безусловни, съответно с два и с един наследник. Всяко изпълнение на правило се нарича стъпка от работата на алгоритъма. Броя на стъпките зависи от типа на задачата. В много от случаите правилата са много пъти по-малко на брой от стъпките.

3. **Свойства на алгоритмите:**
   a. **Определеност** – описанието на алгоритъма е ясно и разбираемо и определя еднозначно действията, които трябва да се извършат;
   b. **Масовост** – даден алгоритъм може да се използва за решаването на всяка една задача от даден тип, а конкретна задача от даден клас се дефинира по входни данни;
   c. **Резултатност** – всяко изпълнение на алгоритъма завършва за краен интервал от време. То е следствие от предположението, че всяко изпълнение се състои от краен брой стъпки и че всяка стъпка се изпълнява за краен интервал от време. От тези интервали зависи дали алгоритъма е приложим на практика;
   d. **Цикличност** – при описване на алгоритъма групи от правила могат да се изпълняват многократно.

4. **Начини на описване на алгоритъма**: словесно, чрез блок-схеми и чрез програмни езици.

При блок-схемите има три вида символни означения: овал – на начало и край на алгоритъма, правоъгълник – безусловно действие (един наследник); ромбоиди – условно действие – два наследника. Тези три вида блока са достатъчни за описването на всеки алгоритъм.

Транслатор – програма на машинен език, която като входни данни има словесно описание на алгоритъма, а като изход връща същия алгоритъм на машинен език. Така се процедира и при блок-схемите. Такива транслатори са правени, но не са практически приложими. Алгоритмите за транслиране са дълги и сложни. По тази причина се създават езици със строго дефиниран синтаксис, чрез който се улесняват алгоритмите на транслаторите. Такива езици се наричат алгоритмични езици.

Обща структура на компютрите. Процесори. Видове вътрешна памет. Програма на машинен език.

Апаратна част – Hardware – Хардуер
Програмна част – Software – Софтуер

Софтуера включва – операционни системи, среди за програмиране, приложни системи.

1. **Операционни системи** – програми, които управляват работата на централния процесор. Драйверите са необходими да управляват работата на периферните устройства.
2. **Среди за програмиране** – средства, които подпомагат за конструирането на програмата. Задължително включва:
    - **i.** Текстов редактор;
    - **ii.** Транслатор за съответния програмен език;
    - **iii.** Библиотека – програми записани в първичен код, които могат да се използват;
    - **iv.** Помощна система
    - **v.** Средства за проверка на програмите
3. **Приложни системи**

Хардуера включва: централен процесор, вътрешна памет, периферни устройства, шина

1. **Централен процесор –** включва аритметично (обработващо) устройство, управляващо устройство, регистър. Процесора има две характеристики – тактова честота и колко битов е процесора. За един такт процесора изпълнява една операция. Тази характеристика се измерва в MHz и GHz, като 1GHz означава че изпълнява 1 милиард операции за една секунда. Колко битов е процесора зависи от това колко битови са операндите. В момента навлизат 64 битови процесори. По своята архитектура, процесорите биват CISC и RISC.

CISC – complex instruction set computing – пресмятане с комплексно множество от команди. Използват се масовите компютри – бизнес приложения, интернет, мрежи, база данни и т.н. Дават слаби резултати при работа с числа със плаваща запетая.
RISC – пресмятане с редуцирано множество от команди. Използват се при научна дейност, когато се набляга на работата с голяма точност и числа с много цифри след десетичната запетая.

2. **Вътрешна памет –** постоянна ROM (read only memory) и оперативна RAM (random access memory). Вътрешната памет единствено се достъпва от процесора. Разликата между постоянната и оперативната памет е че постоянната

е достъпна единствено за четене. В началото от нея се стартират различни програми тестващи хардуера и .т.н и т.нар. BIOS (basic input-output system). Информацията записана в ROM се запазва дори и след изключване на компютъра или токов удар. В оперативната памет можем да записваме и четем различни данни. След изключване на компютъра или друго прекъсване на нормалната му работа, данните записани в RAM се изтриват.

3. **Начин на работа на хардуера –** от външен носител, входните данни заедно с програмата се прехвърлят на оперативната памет. Командата за обработка се подава в регистъра, след което управляващото устройство дешифрира командата и я изпраща на аритметичното устройство за обработка. Резултата се връща в регистъра, от там в RAM-а и отново чрез периферните устройства на външен носител.

Преноса на информация от периферията до RAM-а и регистъра и обратно става по шината. Паметта на логическо ниво се разглежда като последователност от двоични разряди. Един двоичен разряд може да приема стойности 1 и 0. Един двоичен разряд се нарича още бит. Най-малката единица от паметта, която има адрес се нарича байт. С помота на адреса се указва даден байт. Обикновено 1 байт = 8бита. Една от характеристиките на паметта е нейния обем.

Командата се състои от код на операцията и адресна част. В зависимост от кода на операцията, операциите могат да бъдат аритметични, логически, управляващи (обикновено за предаване на управлението), обменни (копиране на информация от регистъра в оперативната памет) и входно-изходни операции. Адресните части могат да са с един адрес, два адреса и три адреса. Адресите могат да са:

- ✓ **Непосредствени –** самия операнд е записан в адресната част
- ✓ **Абсолютни –** това което е записано е точен адрес в паметта или регистъра
- ✓ **Косвена регистрова адресация –** адреса е адрес на регистър където е записан истинския адрес, това се прави за да се намали адресната част и дължината на командата
- ✓ **Относителни адреси**

Периферни устройства и външна памет. Начини на представяне на числата.

Периферните устройства биват три основни вида: за въвеждане за извеждане и за въвеждане и извеждане. При периферните **устройства за въвеждане**, информацията от външен носител се прехвърля в оперативната памет.

При **устройствата за изход**, се извежда информацията от RAM паметта към външен носител. Такъв вид устройства са мониторите. Те се използват заедно с видеоконтролери (видео карти) с видео памет, която извежда информацията от оперативната памет. Други устройства за изход са принтерите. Те биват три вида – матрични, мастилено-струйни и лазерни. Тенденцията е матричните да се използват във финансовата сфера, тъй като при тях може да се извежда повече от едно копие с един

цикъл на работа на принтера. Другите два типа обаче са доста по качествени и тихи. При лазерните принтери има барабан, върху който има светлинно изображение на това което трябва да се принтира. Според светлината по листа се полепва прах от тонер касетата, след което този прах се изпича и остава на листа. При мастилено-струйните, мастилото се излива на капчици върху листа от т. нар. глави.

Устройствата за вход и изход са:

- ✓ Твърд диск – представлява две плочи една над друга които се въртят
- ✓ FDD – floppy disk device – 3,5'' дискети, които в момента не се развиват и с течение на времето излизат от употреба
- ✓ CD Device (compact disk) – биват три вида ROM (read-only), R (за еднократно записване и многократно четене) и RW (за многократно записване и четене)
- ✓ DVD Device – аналогични на CD, но с по голям капацитет. Има съвместимост на DVD устройствата със CD, но не и обратното.

Начини на представяне на целите числа: Има три начина на представяне на целите отрицателни числа във двоичен вид.

- ✓ Пряк код: старшия бит има стойност 1 когато числото е отрицателно и 0 когатое положително. Например: числото -5 : 10000101
- ✓ Обратен код: старшия бит се запазва, а всички останалите приемат обратната си стойност. Числото -5: 11111010
- ✓ Допълнителен код: всичко е както и при обратния, но когато числото е отрицателно, към младшия бит се прибавя 1

Използването на различни начини за представяне е свързано с работата на централната процесорна единица. Целта е алгоритмите за събиране и изваждане да са възможно най-прости. При обратния код, изваждането се свежда до събиране на две двоични числа и когато резултата е отрицателен се прибавя една циклична единица. Допълнителния код прибавянето на цикличната единица се избягва, а всичко останало е както и при обратния код.

В C/C++ се използва допълнителен код!!!

Представяне на реални числа: Използва се един бит за знак на мантисата, един за знак на порядъка, няколко бита за записване на порядъка и няколко бита за мантиса като числото се получава по формулата – **мантиса * 2** $^{\text{порядъка}}$

Мантисата трябва да бъде нормализирана, т.е. старшия бит да е 1. Тя представлява частта след десетичната запетая, понякога старшия бит може да се изпусне, тъй като за него се знае че е винаги 1. В някой компютри се използва шестнайсетично представяне на мантисата.

Порядъка може да има, а може и да няма знак. Като когато няма знак, числата се смятат по формулата: **мантиса * 2** $^{\text{порядъка – константа}}$

Константи. Видове. Вътрешно представяне.

1. **Видове константи –**
    a. **Числови:** това включва цели (десетични, осмични и шестнайсетични) и дробно десетични
    b. Символни
    c. Низови

При целите десетични константи, числото не може да започва с 0, тъй като това е признак че числото е в осмична бройна система: 0256. Когато числото е в шестнайсетична система, то е във вида: 0x и число: 0xFF

2. **Вътрешно представяне на целите числа** – в зависимост от средата, данните могат да варират. Когато се задава адрес, се използва адреса на младшия байт. Представянето на константите става във възможно най-малкия тип. Насилствена промяна на типа става със суфикси: L за long и U за unsigned. Те се поставят след константата, например: 37L. За Borland C++:

    a. **Int** – 2B – като старшия бит е знаков
    b. **Unsigned int** – 2B – като всички битове са за стойността на числото
    c. **Long int** – 4B – със знаков старши бит
    d. **Unsigned long int** – 4B – без знаков старши бит

3. **Представяне на дробните десетични числа** – за десетичен знак е прието да се използва точка. Ако точка присъства в записа на число дори и след нея да няма нищо, числото се приема като тип double. Например: 27. ; -157.23 и т.н. Съществува и следния начин на запис: знак-мантиса-E-порядък. Например: $-25E07 = -25*10^7$.

    a. **Float** – 4B – един бит за знак на мантисата, 1B за порядъка и 23 бита за мантисата, диапазона е от $|10^{-38};10^{38}|$ и съхранява числа с точност 7 цифри след десетичната точка
    b. **Double** – 8B – знаков бит, 11 бита за порядъка и 52 за мантисата, диапазона е $|10^{-308};10^{308}|$ - съхранява числа с точност 15 цифри. Числата по подразбиране се възприемат като тип double.
    c. **Long double** – 10B – знаков бит, 15 бита за порядъка и 64 за мантисата, диапазона е $|10^{-4900};10^{4900}|$ и съхранява числа с точност до 19 цифри.

При резултат $< 10^{-4900}$ се получава машинна нула, а при $> 10^{4900}$ резултата е неопределеност. Суфиксите за промяна на типа са: L за Long double и f за float.

4. **Символни константи –** единичен символ, заграден в апострофи: ‘A’. Един символ се съхранява в един байт, като се представя чрез някакъв код. Известни са различни системи за кодиране на символите. Представя се като тип char.

ASCII код – American standart code for information interchange.

5. **Низови константи –** последователност от символи, заградени в кавички: „абс”, „А” или празен низ „”. Разглежда се като символен едномерен масив или като указател към символен низ: char*

Низовите константи се съхраняват в паметта като едномерни масиви от символи завършващи с нулев байт. Този нулев байт е признак за край на низа. Броят на елементите в масива е с едно по-голям от елементите в низа.

Променливи и типове данни.

Задача. Изчислете периметъра на окръжност и лицето на кръг, по въведен радиус.

```
#include <stdio.h>
void main()
{
     float r,p,s;
     printf("Vuvedete r="):
     scanf("%f",&r);
     p=2*3.14*r;
     s=3.14*r*r;
     printf("Perimetura e %f,a liceto e %f",p,s);
}
```

1. **Групи символи:**
   - ✓ букви – всички малки и големи, от латинеца или кирилица букви и символа „долно тире” – „_”;
   - ✓ цифри – всички цифри 0,1,...,9;
   - ✓ празни символи – интервал, табулация и нов ред (enter);
   - ✓ специални символи – всички символи от клавиатурата, които не влизат в горните три групи;

Идентификатор – последователност от букви или цифри, която започва с буква. В езика C/C++, както и в повечето езици за програмиране съществуват резервирани (запазени) идентификатори. Те обикновено се използват за означаване на синтактичните единици на езика, наричат се още ключови думи. Дължината на идентификатора, не е определена от стандартите на езика, но всеки компилатор има зададена като стойност някаква дължина за идентификаторите. По-дълъг идентификатор също ще бъде приет, но сравняването на различните идентификатори, става само до посочения символ. По-нататък те се възприемат като еднакви.

2. **Променливи –** всяка променлива, съдържа име (идентификатор) и стойност. Стойността се съхранява в един или няколко последователни байта, разпределени за съответната променлива. Всяка променлива, преди да се използва, трябва да се декларира. С дефинирането и за нея се разпределя памет в съответствие със зададения и тип/ Дефиницията на една променлива има следния вид:

<тип-на-променливата> <списък-с-имена>; **Например**:

```
float r,p,s;
```

3. **Типове на променливите:** в C/C++ те могат да бъдат следните:

signed char – 1B	unsigned char – 1B	char – 1B	
int – 2B	long int – 4B	unsigned int – 2B	unsigned long int – 4B
float – 4B	double – 8B	long double -10B	

Вътрешното представяне в паметта е както при константите. Запазва се и диапазона им. При задаването на дадена променлива и типа и, следват следните неща: размер памет и начин на кодировка. Според това се получава и диапазона. Като тип може да се използва и short, това се налага при компилатори които използват повече от байтовете за дадена променлива, отколкото са ни необходими. Unsigned и signed при тип char имат значение, когато променливите от този тип се интерпретират като числа, т.е. когато се налага да се смята с променливи от този тип. В повечето случаи по подразбиране е unsigned.

Коментари и структура на програмата.

1. **Коментар –** поясняващ текст към програмите. Може да се включи навсякъде където можем да поставим интервал, без това да промени смисъла на програмата. Синтаксиса е следния:

   /* И тук се въвежда коментара ***/**

В стила на C++ може да се използва едноредов коментар, който игнорира всичко след себе си до край на текущия ред. Синтаксиса е „ // коментар тук „.
В софтуерната промишленост коментарите са задължителни. Те дават цялостна представа за работата на програмата, алгоритъма, авторите т.н. Те трябва да включват всички необходими бележки и упътвания, за да може програмата да се разбере без допълнителни източници. Съществуват методи на програмиране, при които първо се изписват само коментарите и след това съдържанието на самата програма.
При проверка на програмата за грешки и отстраняването, понякога се налага част от кода да се изолира и това става най-лесно с поставянето му в коментар. В такъв случай трябва да се внимава да не се получат т.нат. вложени коментари (т.е. един коментар в друг) и да се провери дали работната среда (в частност компилатора) поддържат това. Коментарите са обект на обработка от препроцесора. Той ги премахва, за да не се отразят на кода на програмата.

Схема на работа на работната среда при компилирането на дадена програма:

изходен файл → препроцесор (коментари, заглавни файлове, макроси) → транслатор (обектен файл) → свързващ редактор → изпълним файл (програма) с разширение *.exe

2. Етапи на програмирането:
   a. **въвеждане и коригиране на програмата**
   b. **съхранение на програмата** в съответния файл с разширение *.c или *.cpp
   c. **компилиране на програмата** и получаването на обектен код, оформен като файл *.obj
   d. **свързващо редактиране** – в резултат от свързването на всички обектни файлове в програмата, както и всички библиотечни функции се получава изпълним файл - *.exe
   e. **изпълнение на програмата**

3. **Структура на програмата –** В езика С програмата представлява съвкупност от функции. Между тях или преди тях може да има заглавни файлове или външни променливи. Точно една от тези функции има име main. Тя се смята за главна функция в програмата и когато файла се изпълнява, операционната система предава управлението именно на тази функция. Края на функцията main, означава и край на програмата.

Операции и изрази

Задача. Даден е правоъгълен триъгълник с хипотенуза z и разлика на катетите a. Намерете катетите x и y.
Решение:

```
#include <stdio.h>
#include <math.h>

void main(){
double z,a,d,x,y;
printf("Vuvedete z I a:");
scanf("%lf %lf",&z,&a);
d=sqrt(2*z*z-a*a);
x=(a+d)/2;
y=(a-d)/2;
printf("X I Y sa: %lf I %lf",x,y);
}
```

1. **Формули в програмирането –** записването на формули в математиката графично е многоетажно, докато в програмирането, то е линейно. Не се допуска изпускане на знака за умножение(*).
2. **Израз –** всяка валидна за езика комбинация от операции, операнди и кръгли скоби. Под валидна комбинация се разбира комбинация, която не противоречи на стандартите на езика. Операндите могат да бъдат променливи, константи или обръщения към функции. Операциите в зависимост от броя операнди биват:
    i. **унарни –** унарен плюс (+) и унарен минус (-)
    ii. **бинарни –** (+), (-), (*), (/), (%). Тези операции запазват типовете. Т.е. при делене на цяло число на цяло число резултата е отново цяло число.
    iii. тернарни.

Всяка стойност различна от 0 може да се интерпретира като true, а всяка стойност която е 0 като false. Възприето е, стойността на резултата при релации и логически операции да е винаги или 1 за true, или 0 за false. Само в новите версии на C++ имаме дефиниран логически тип (т.е. тип който може да приема само стойности 1 или 0)

Релации: (<), (<=), (>), (>=), (= =), (!=)
Логически операции: && (логическо умножение), | | (логическо събиране), ! (логическо отрицание)

В някои компилатори, ако имаме поредица от логически операции (оп1 && оп2 && ...) израза се изчислява само докато стане ясно каква е стойността на целия израз. При такава оптимизация на компилатора, може да се получи грешка в програмата, ако в тези операнди са поставени изрази, то които зависи реализацията на кода, по-нататък.

Присвояване: общ вид – променлива = израз. Тази операция връща резултат. Пресмята се израза, след което типа на стойността му се преобразува към типа на променливата. Резултата, който се връща е стойността на целия израз. Например: `x=y=100` е валиден израз. Операцията се изпълнява от дясно на ляво. Ако след израза поставим точка и запетая (;), операцията се превръща в оператор.

Съкращения за често използвани комбинации от операции:

a=a+b → a+=b a=a-b → a-=b
a=a*b → a*=b a=a/b → a/=b
a=a%b → a%=b a=a+1 → ++a → a++
a=a-1 → --a → --a

Разликата между a++ и ++a е че при ++a първо се изпълнява операцията и след това променливата се използва с новата си стойност. Когато тези две операции се използват самостоятелно те са еквивалентни.

```
int x,y;
x=15;     // x=16
y=++x;    // y=16 , докато

int x,y;
x=15;     // x=16
y=x++;    // y=15.
```

Операция , - общ вид: операнд1,операнд2. Операция за последователно изпълнение. Резултата от тази операция е стойността на последния израз.

Явно преобразуване на тип – общ вид: (тип) операнд; Например:

```
int suma, broi;
float sr_uspeh;
sr_uspeh = suma/(float)broi;
```

Размер на областта – sizeof(operand) – връща размера на операнда в байтове.

3. **Ред за изпълнение на операциите в израз –** определя се от приоритета и асоциативността им. Операциите с равен приоритет се изпълняват в съответствие с тяхната асоциативност.

Таблица на приоритетите:

Приоритет	Операция	Асоциативност
1. (най-нисък)	( ) – извикване на функция [ ] – достъп на елементите от масив	→
2.	Отрицание – (!), унарен плюс (+), унарен минус (-), инкрементиране (++), декрементиране (--)	←
3.	(*), (/), (%)	→
4.	Бинарни събиране и изваждане (+) и (-)	→
5.	**<< >> - побитово отместване**	
6.	(<), (<=), (>), (>=)	→
7.	Равенство и неравенство (= =), (!=)	→
8.	**& - побитово и**	
9.	**^ - побитово изключващо или**	
10.	**| - побитово или**	
11.	Логическо „и” (&&)	→
12.	Логическо „или” (| |)	→
13.	Условен оператор (? :)	←
14.	Присвояване и производните му (=, += ...)	←
15. (най-нисък)	**Последователно изпълнение**	→

Преобразуване на типовете

При новите версии, при преобразуванията при които може да се получат грешки, се изисква то да се прави в явен вид.

(операнд1) операция (операнд2)

Операция може да се извърши само ако и два операнда са от един тип. В случай че те не са, по-младшия тип се преобразува в по-старшия:

подредба на типовете по старшинство:
long double → double → float → unsigned long int → long int → unsigned int → int →
→ unsigned char → signed char

Подреждането на типовете се извършва по обема памет, която се заделя за всеки тип. За тези които заделят равен обем памет, решаващо е кой от двата може да съхранява по-големи стойности.

В случай, че някой от операндите или и двата операнда са char (без значение от кой тип), те се преобразуват в int.
Преобразуване **int → unsigned int** кода не се променя, но се променя интерпретацията на числото. Тоест една и съща последователност от 1 и 0 ще се интерпретира като две различни числа, което води до грешка в алгоритъма.

При преобразуване **int → (unsigned) long int** се добавят два старши байта към операнда от тип int за да се изравнят. Тези старши 2 байта се запълват със знаковия бит на числото. Тоест ако се преминава от int → long int числото се запазва.

При преобразуване **unsigned int → long int** 2-та добавени байта се запълват с 0, при което числото не се променя.

При преобразуване **int → [float, double, long double]** числото се представя с мантиса и порядък и стойността се запазва.

При извършване на операция присвояване (общ вид: променлива = израз), когато типовете са различни, стойността на израза без значение дали по-старши или по-младши се преобразува към типа на променливата, а новата стойност се запазва в променливата. При това, възможно е да се получи грешка ако се преминава от старши към младши тип и числото не може да се запише в младшия тип. Такава грешка няма да се изведе от компилатора и може да остане незабелязана, но резултата е ще е грешен.

При преминаване от long int → int се изхвърлят двата старши байт, без значение каква е стойността им.

Условни изрази и условни оператори

задача. Съставете програма за пресмятане на заплащането за 1 седмица. Приемаме че за извънреден труд се смятат часовете над 40 седмично.

suma = 1) h*norma, h<=40 2) 40*norma + (h-40)*1.5*norma

```
#include <stdio.h>
void main() {
int cod; // zapazwa li4niq kod na wseki rabotnik
float h,norma,suma; //suotvetno za 4asovete trud, zaplata na
4as I kraen rezultat
printf("Vuvedete cod, h I norma:");
scanf("%d %f %f ", &cod, &h,&norma);

if (h<=40) suma = h*norma;
else suma = 40*norma + (h-40)*1.5*norma;
printf("%6d  %10.2f", cod, suma);
}
```

%6d – означава че променливата от тип инт е се изпише минимум в шест позиции и ще бъде подравнена от дясно.
%10.2f – означава че променливата от тип float е се изпише в минимум 10 позиции като 2 от тях ще са за цифрите зад десетичната запетая.

1. **Операция за условен израз**
   общ вид: операнд1 ? операнд2 : операнд3

Това е тернарна операция, която пресмята операнд1, ако стойността му е различна от 0 резултата от операцията е операнд2, стойността на операнд1 е 0, то резултата на цялата конструкция е операнд3. Ако операнд2 и операнд3 са от различен тип, резултата се преобразува към по-старшия.

2. **Условен оператор If**
   общ вид: if (израз1) операнд1; else операнд2; В кратка си форма else операнд2 отсъстват.

Условния израз е оптимизация на условния оператор. Препоръчва се при по-прости изрази, където е възможно, да се използва условния израз.

След if и else може да следва само един оператор. Ако е необходимо да се изпълнят няколко оператора в тялото на if конструкцията се използва, т.нар. „съставен оператор"

3. **Съставен оператор**
   общ вид: { няколко оператора }

пример:
```
if ((x==y) && (x<10)) {x=2*y; y++;}
```

Допуска се и влагането на няколко условни оператора един в друг:

if (expression1) if (expression2) statment1 else statement2;

В такава ситуация не е ясно към кой if е else. За това важи следното правило:
Всеки else се асоциира с най-близкия стоящ от ляво на него if, който не е асоцииран с друг else. Препоръчва се влагането на if след else, което изклчва възможността за получаване на логически грешки.

Масиви

пример:
```
char str[10];
double mat[3][2];
int tabl[4][3][2];
```

[ ] – определя, че се дефинира масив. Числото в него трябва да бъде константен израз, който да може да се изчисли по време на компилацията. Стойността представлява броя на елементите на масива.
В общия случай, индекса на масива е израз от цял тип. Първия елемент има индекс 0, т.е. максималния индекс в един масив е с 1 по-малък от съответната зададена граница.Елементите на масива се съхраняват последователно в паметта. Ако масива е много размерен, индексите се изменят от дясно на ляво, т.е. по-бързо се изменят индексите, които са по-вдясно.

В C/C++ не съществува променлива от тип низ от символи, затова всеки едномерен масив от тип char се възприема като променлива от такъв тип.

Оператор за цикъл for

Общ вид на оператора: for (израз1; израз2;израз3) оператор;
израз1 – начална стойност на променливите в израз 2
израз 2 – условие, управляващо продължителността на цикъла
израз3 – някакво правило, по което променливата от израз 2 се променя

изрази 1 и 3 обикновено са присвояване, а израз 2 е сравнение.

Най-често се използват циклите с предварително известен брой итерации.
пример:

```
int i,n1,n2,result;
result=0;
n1=3;
n2=5;
for (i=n1;i<=n2;i++)
     result+=I;
```

В синтаксиса на for израз 1,2 и 3 не са задължителни. Всеки от тях поотделно или в комбинация с останалите, може да бъде пропуснат. В тези случаи ; отнасящи се към пропуснатия израз, трябва задължително да присъства. Ако е пропуснат израз 2 се смята че той винаги е true и така се получава безкраен цикъл.

търсене на края на низ от символи:

```
int I;
char str[50];
for (i=0;str[i++]!="\0";) ;
printf("%c","a[i-2]);
```

Оператор "goto"

Оператор за безусловен преход. Общ вид:

```
goto етикет;
.....................
етикет: оператор;
```

Етикетите представляват идентификатори. Използването на оператора за влизане в цикъл е опасно и води до неопределен резултат. Управлението може да се предава само в рамките на една функция. Прескачането между различни функции е забранено в стандарта на С.

Оператор за избор на вариант "switch"

Общ вид: switch (израз_селектор)

{case израз1: оператори; break;
 case израз2: оператори; break;
****
default: оператори;
}

Чрез оператор **switch** се избира една от няколко взаимно изключващи се алтернативни възможности. Израз-селектор е израз от тип char или int. Представлява условието за избор на вариант. Израз1 израз2 ... са константни изрази от тип char или int.Те могат да се разглеждат като етикети към следващите ги оператори. **Case, switch и default** са ключови думи в синтаксиса на C/C++.

Оператора действа по следния начин:

- ✓ пресмята се израза селектор
- ✓ тази стойност се сравнява с израз1, израз2 ...
- ✓ ако стойността на селектора съвпада със стойността на някой константен израз се изпълняват операторите след този израз.
- ✓ в случай че има break управлението се предава на оператора, който се намира след switch конструкцията.
- ✓ ако няма никакво съвпадение и присъства конструкция default се изпълняват операцията към нея.

В switch в израз1, израз2 .. не може да има повтарящи се стойности. В какъв ред се пресмятат изразите и в какъв ред се проверява дали те съвпадат с израза селектор, зависи от компилатора и реда може да бъде произволен. Ако имаме следния фрагмент от код:

```
case izraz1:
case izraz2:
case izraz3:
-----------
case izrazK:
            operatori;
```

тогава ще се изпълни групата от оператори ако израза-селектор съвпадне с кой да е от изразите от 1 до К. Ако оператор break отсъства след изпълнението на операторите към дадения етикет, програмата продължава с изпълнението на операторите от следващия вариант. Вместо break може да се използват и още: go to, return, exit();

Операторите switch могат да се влагат един в друг. Switch може да бъде заместен от серия вложени if – else конструкции. Това се препоръчва, тъй като е ясен реда на проверката на изразите и пресмятането им. Може да се направи и оценка на възможностите и най-често случващия се вариант да се постави в началото, като така се спестява от времето за работа на програмата. От друга страна switch се използва за прегледност и яснота на работа на алгоритъма.

Като аргумент на функцията exit() се подава число. В зависимост от реализацията на програмната среда, различните стойности, могат да означават различни неща за операционната система. За стандартизиране на работата се използват макросите:

EXIT_SUCCESS или EXIT_FAILURE

например: exit(EXIT_SUCCESS);

Това означава, че програмата приключва и връща на операционната система команда, че програмата е приключила успешно.

Оператор break

Break може да се разположи на всяко място, където може да се постави кой да е друг оператор в тялото на даден цикъл. Изпълнението му, предизвиква излизане от цикъла и предава управлението на оператора след цикъла. След излизането от цикъла, стойностите на променливите от цикъла не се променят, като се запазват такива каквито са били преди изпълнението на break

Оператор continue

Този оператор се използва за завършване на итерацията на цикъла. Може да бъде записан на произволно място в тялото на цикъла, разрешено за оператор. След изпълнението на оператора, цикъла продължава със следващата итерация.

Форматиран изход. Функцията printf()

Тези функции се използват за форматиран вход и изход, т.е. възможността за преобразуване на входно-изходната информация, която е последователност от символи на външен носител в определен тип данни в оперативната памет. Тази информация по-късно може да бде изведена на външен носител.. Такъв вход и изход дава възможност за форматиране на в/и информация.

printf(“форматиращи параметри”,списък с аргументи);

Форматиращите параметри са два вида:

- ✓ последователност от символи, които се извеждат графично, а управляващите се изпълняват
- ✓ последователност от форматиращи символи, които не се извеждат, но управляват показването на съответния аргумент от подадения списък.

Възможно е и смесване на двата вида форматиращи параметри.

общ вид на модификаторите: %[модификатор]форматиращ спецификатор

Модификатори	Форм. специфик.	Интерпретация	Показване
(1) минимална дължина на полето, форматиране от дясно - %10d (2) форматиране то ляво - %-10d	d	int	Цяло число със знак
(1) и (2)	u	unsigned int	Цяло число без знак
# - включва водещата 0	o	unsigned int	Осмично число без водещата 0
# - включва водещата 0x	x или X	unsigned int	Шестнайсетично число без водещите 0x
l	d, u, o, x, X	long	

точност – задава явно точността: %.2f , използват се и (1) и (2)	f	float	Дробно десетично число с 6 цифри в дробната част
l	f	double	
L	f	long double	
използват се „точност” (1), (2) и l и L	e или E	float	Реално число във вид с мантиса и порядък
	c	char	единичен символ
(1) (2) “точност” – броя на символите от низа които ще бъдат извеждани	s	char*	низ от символи
	%%		%

Със знака * може да се задава минимална дължина на поле, която се задава като израз точно преди променливата за която се отнася.
Функцията printf() е от тип int и връща като резултат броя на изведените символи или макроса EOF, ако има грешка в изпълнението.

Форматиран вход. Функцията scanf()

Общ вид на функцията: scanf(“форматиращи параметри”, списък от аргументи);
Подобно на printf(), scanf() също връща резултат. Това са броя въведени аргументи или макроса EOF.
Форматиращите параметри са аналогични на тези на printf(). Scanf() присвоява въвежданата информация на определени променливи. Затова аргументите на функцията са указатели към тези променливи. Адресите се получават чрез операцията &. Изключение правят низовите променливи, тъй като при тях, името на самия низ е указател към първия му елемент.
Въвежданата информация се нарича входен поток от данни. В този поток, всеки елемент от данните се отделя за да може да се съответства на зададения аргумент в списъка на scanf, с помощта на т.нар. празни символи.

Обръщение към функции

Общ вид: име-на-функция(списък с аргументи);

В общия случай, аргументите са изрази. Ако след обръщението няма ; - функцията задължително трябва връща резултат, в такъв случай - тя се възприема като операнд.
Ако след обръщението има ;, тя се разглежда като оператор, в такъв случай не е задължително тя да връща резултат.

Заместване на формални параметри с фактически параметри.
Между параметрите трябва да има съответствие по тип и брой. Вградения механизъм в C/C++ е заместване по стойност на параметрите. Пресмятат се изразите, които са подадени като аргументи, проверява се типа на фактическите параметри и при нужда се правят съответните преобразувания, като типа на формалните се преобразува към типа на фактическите. Накрая получената стойност се присвоява на фактическия параметър.

/* Съставете програма, която намира максимума от редица от числа*/

```
#include <stdio.h>

double maxi(double c[],int n)
{
     int i;
     double m;
     m=c[0];
     for (i=1;i<n;i++)
          if (m<c[i]) m=c[i];
     return m;
}

int main()
{
     int br,i;
     double mas[100];
     printf("Vuvedete broi elementi:");
     scanf("%d",&br);
     for (i=0;i<br;i++)
          scanf("%d",&mas[i]);

     printf("Max e %lf", maxi(mas, br));
     return 0;
}
```

Параметрите се пресмятат от дясно на ляво и записват в програмен стек. Стойността на променливите се записват в стека, от горе на долу. Записва се стойността на името на масива(т.е. адреса към първия му елемент). Управлението се предава на извиканата функция. Следва да се разпредели място в паметта за локалните параметри. Паметта се разпределя от дясно на ляво. Когато се ползва масив, ние използваме подадения масив и се извършва заместване по име. Затова не се декларира броя на елементите, прие едномерните масиви.

При двумерните масиви, името му е адреса към адрес на първия елемент и така за всеки следва ред. Затова при повече от едно-мерен масив, се изисква да се предават броя на елементите му, тъй като те са необходими при пресмятането и достъпването на елементите по адреси.

Прототипи на функции

Ако извикваната функция е разположена след извикващата, тя трябва да бъде декларирана предварително. Това деклариране се нарича прототип на функцията, който информира компилатора за типа на връщаната стойност, името на функцията и списъка и с аргументи. Използвайки прототипи компилатора извършва контрол по съответствието на броя и типа на параметрите и при необходимост извършва преобразувания. Точката със запетая накрая на прототипа означава, че съответната конструкция е прототип, а не дефиниция на функцията.

Класове памет

Спецификаторите за съхраняване extern,auto, register, static и mutable се използват за променяне на начина по който C/C++ съхранява променливите. Тези спецификатори предхождат името на типа който модифицират.

Extern
Ако спецификатора extern е поставен преди име на променливата , компилатора ще знае че променливата има вътрешно свързване. Външното свързване значи че обекта е видим извън неговия файл. Всъщност ,extern казва на компилатора типа на променливата без в действителност да заделя съхранител за нея. Модификатора extern е най-често използван когато имаме два или повече файла споделящи еднакви глобални променливи.

Auto
Спецификатора auto казва на компилатора че локалната променлива която следва е създадена при влизането в блок и се разрушава при излизането от блок. Откакто всички променливи дефинирани вътре във функциите са auto по подразбиране , ключовата дума auto е рядко (ако дори) използвана.

Register
Когато беше създаден C ,register можеше само да бъде използван върху локални целочислени илисимволни променливи понеже той беше причина компилатора да направи опит да пази променливата в регистрите на CPU вместо да ги поставя в паметта . Това правеше всички извиквания на променливата извънредно бързи. Дефиницията на register оттогава беше разширена .Сега всяка променлива може да бъде определена като register и е работа на компилатора да оптимизира достъпа до нея. За символите и целочислените , това още означава използване на кеш паметта ,например.Запомнете че register е само заявка. Компилатора е свободен да я отхвърли. Причината за това е че ограничен брой променливи могат да бъдат оптимизирани за скорост. Когато този лимит бъде надвишен, компилатора просто игнорира по нататъшните register заявки.

Static
Спецификатора static инструктира компилатора да пази променлива в съществуване през цялото действие на програмата вместо да я създава и разрушава всеки път когато влиза и излиза от обсега. По този начин правейки локални променливи static позволяваме да поддържат техните стойности между извикванията на функциите .Модификатора static може да бъде приложен към глобални променливи .Когато това е направено , то е причина за ограничаване

обхвата на променлива във файла в който е декларирана .Това значи че тя ще има вътрешно свързване . Вътрешно свързване значи че идентификатора е известен само във неговия собствен файл. В C++ ,когато static е използвана върху членовете-данни на клас причинява само едно копие от този член да бъде разделен между всички обекти от този клас.

Mutable

Спецификатора mutable е приложен само в C++. Той позволява на член на обект да предефинира константността. Така mutable член на const обект не е const и може да бъде изменян.

Разделно компилиране

По подразбиране, имената на функциите са външни, т.е. в един модул може да се обръщаме към функция компилирана в друг модул. В извикващия модул трябва да се опише прототипа на извикваната функция, а в началото да се постави extern. Аналогична е ситуацията и при обръщение към стандартни функции.

Обикновено описанията на прототипите на тези стандартни функции се оформят в заглавни файлове, които се включват с директивата #include. Тъй като прототипите на функциите действат в областта на един модул, в който са описани, то е удобно те да се оформят в отделни файлове и да се включва навсякъде където са необходими. Ако една функция е дефинирана като static, към нея може да има обръщение само в модула, в който е дефинирана. Променлива, която в някой модул е дефинирана като extern, може да се инициализира само във файла, където е дефинирана като външна променлива.

Инициализиране на променливите

Ако за една променлива се разпределя памет в областта за статични данни, тя се нулира. В противен случай, началната и стойност е неопределена. Всяка променлива може да се инициализира по време на декларацията и по следния начин:

```
тип променлива=израз;
```

При това важи следното правило: ако за една променлива се разпределя памет в областта за статични данни, израза трябва да е константен, тъй като се изчислява по време на компилация. Иначе израза може да е, произволен. Ако за вътрешно статична променлива се разпределя памет в областта за статични данни, тя се инициализира само един път, преди първото изпълнение на съответната функция, където е дефинирана и запазва последната текуща стойност при всяко следващо влизане във функцията. Ако е необходимо, да се инициализира отново при всяко влизане, това се прави чрез оператор за присвояване =.

```
void f1(int x, int y)
{ int i=1;
   int j=x*y;
   } (предпочита се)
void f1(int x, int y)
{ int i,j;
   i=1;
   j=x*y;
   }
```

Масивите могат да се инициализират по време на тяхното дефиниране, чрез списък от начални стойности на елементите им. Той се поставя в {}.

```
int list[5]={0,1,2,3,4};
```

Подреждането на списък, съответства на реда на разпределянето на елементите в RAM-а. Ако няма съответствие между броя и началните стойности са по-малко от броя на елементите, то тези които са в повече се нулират. Ако началните стойности са повече от елементите, обикновено компилатора връща съобщение за грешка. Възможно е и вложено инициализиране, като се използват вложени {}, с кото елементите се инициализират по редове.