Годишен преговор

Структура на програмата

```
- #include <библиотеки>

- // глобални декларации и функции

- int main()
- {
-    // локални декларации
-       // команди
-   return 0;
- }
```

Основни команди

- Деклариране на променлива:
  тип име;
  тип име=стойност;
- Деклариране на константа:
  const *тип име=стойност;*
- Команда за присвояване:
  променлива=израз;
- Команда за въвеждане:
  cin>>*променлива;*
- Команда за извеждане:
  cout<<*израз;*

Целочислени типове данни

Тип данни	**Пример**
char, signed char	-128...127
unsigned char	0...255
short, signed short	-32768...32767
unsigned short	0...65535
int, signed int	-32768...32767
unsigned int	0...65535
long, signed long	-2147483648...2147483647
unsigned long	0...4294967295

Реални типове данни

Тип данни	**Точност**	**Пример**
float	7 знака	$-3.4*10^{38}$.. $3.4*10^{38}$
double	14 знака	$-1.74*10^{308}$.. $1.74*10^{308}$

Аритметични операции

- унарен плюс: +
- унарен минус: -
- събиране: +
- изваждане: -
- умножение: *
- целочислено деление: /
- остатък от целочислено деление: %

Приоритет на операциите

- изразите в скоби: ( )
- унарни операции: + -
- умножение, деление, остатък : * / %
- събиране, изваждане: + -

Операции за сравнение

- по-малко: <
- по-малко или равно: <=
- по-голямо: >
- по-голямо или равно: >=
- равно: ==
- различно: !=

Логически тип данни

- деклариране на променливи:

  bool *име;*

- логически константи:

  true *(истина)* = 1

  false *(лъжа)* = 0

Логическо отрицание

- общ вид: **!** *операнд;*

- действие: връща вярно тогава и само тогава, когато операнда има стойност невярно

Логическо умножение, логическо "И"

- общ вид: *операнд1* **&&** *операнд2;*

- действие: връща вярно тогава и само тогава, когато и двата операнда имат стойност вярно

Логическо събиране, логическо "ИЛИ"

- общ вид: *операнд1* **||** *операнд2;*

- действие: връща вярно тогава и само тогава, когато поне единият от двата операнда има стойност вярно

Приоритет на операциите

- изразите в скоби: ( )
- логическото отрицание: !
- логическо умножение : &&
- логическо събиране: ||

Символен тип данни

- Деклариране на променливи:

  char *име;*

- ASCII таблица: таблица от 255 символа и техният пореден номер, наречен ASCII код

Символни константи

Символните константи се ограждат с апострофи. Биват два вида:

- графични: имат графично представяне. Това са буквите, цифрите и другите знаци. Например 'B', '4', '@', ' '
- управляващи: имат специално значение:
  - \n – нов ред
  - \t - табулация
  - \b – изтрива предния символ
  - \a – звуков сигнал
  - \\ - обратно наклонена черта
  - \" - кавичка
  - \0 – нулев символ

Операции над символни данни

- Намиране на ASCII кода на символ:

```
cout<<(int)'A';
```

- Намиране на символ по неговия ASCII код:

```
cout<<(char)66;
```

- Аритметични операции: допустими са, извършват се над ASCII кода на символите

```
cout<<'A'+5;
cout<<A+4;
```

- Логически операции: '\0' се преобразува до **false**, останалите символи – до **true**
- Операции за сравнение: извършват се над ASCII кода на символите

```
cout<<('A'<'B')<<endl<<('A'=='a')<<endl;
```

Неявно преобразуване

- Извършва се от компилатора при следните условия:

- към логически тип: ако се използват логически оператори. 0 и '\0' стават **false**, останалите - **true**

```
cout<<( ('0' && 0) || ('A' && 4) )<<endl;
```

- към числов тип: при аритметични операции. **false** става 0, **true** – 1, символите участват с ASCII кода си

```
cout<<( ('2' + 2) * false )<<endl;
```

Неявно преобразуване

- между числови типове: при аритметични операции между числа, заемащи различен обем памет – преобразуването е към по-големия тип

```
cout<<(10/4)<<' '<<(10.0/4)<<endl;
```

- при присвояване: прави се преобразуване към типа на променливата, при което може да се случи и загуба на точност

```
int a=2.5; int b=2e20;
```

Явно преобразуване

- общ вид:
  (*тип***)***израз***;**

- действие: преобразува стойността на *израз* до указаният в скобите *тип*

- примери:

```
cout<<(bool)'0'<<endl;          // извежда 1
cout<<(int)12.4<<endl;          // извежда 12
cout<<(char)65<<endl;           // извежда A
cout<<(double)10/4<<endl;       // извежда 2.5
cout<<(double)(10/4)<<endl;     // извежда 2.0
```

Команда за нарастване

- Общ вид:
- *променлива*++;

- Действие:

- Взима се текущата стойност на променливата и после се увеличава с 1

- Например:

```
a=10;
b=a++;
```

- Отговаря на:

```
a=10;
b=a;
a=a+1;
```

- ++*променлива*;

- Увеличава се стойността на променливата с 1 и се ползва новата стойност
-
-
```
a=10;
b=++a;
```
-

```
a=10;
a=a+1;
b=a;
```

Команда за намаляване

- Общ вид:
- *променлива--;*

- Действие:

Взима се текущата стойност на променливата и после се намалява с 1

- Например:

```
a=10;
b=a--;
```

- Отговаря на:

```
a=10;
b=a;
a=a-1;
```

- *--променлива;*

Намалява се стойността на променливата с 1 и се ползва новата стойност

-
-
```
a=10;
b=--a;
```

-
```
a=10;
a=a-1;
b=a;
```

Комбинирани команди за присвояване

Общ вид	Действие
a+=b	a=a+b
a-=b	a=a-b
a*=b	a=a*b
a/=b	a=a/b
a%=b	a=a%b

Пълна форма на условната команда

- общ вид:

  if(*условие***)**
  *команда1***;**
  else *команда2***;**

- действие: проверява се условието; ако има стойност **true** (вярно) се изпълнява *команда1*, ако има стойност **false** (невярно) – *команда2*

- особености: ако трябва да се изпълни повече от една команда, те се ограждат с { }

Кратка форма на условната команда

- общ вид:

  if(*условие***)**
  *команда1***;**

- действие: проверява се условието; ако има стойност **true** (вярно) се изпълнява *команда1*, ако има стойност **false** (невярно) – нищо не се изпълнява за тази команда

- особености: ако трябва да се изпълни повече от една команда, те се ограждат с { }

Условен израз

- общ вид:
  условие?израз1:*израз2*

- действие: проверява се условието; ако има стойност **true** (вярно), на израза се присвоява стойността на *израз1*, ако има стойност **false** (невярно) – стойността на *израз2*

- пример:

```
bool a;
cout << (a? "TRUE" : "FALSE") << endl;
```

Вложена условна команда

- Когато в една условна команда на мястото *команда1 или команда2* имаме друга условна команда, говорим за вложена условна команда

- условна команда:
  - **if(***условие***)** *команда1*; **else** *команда2*;
- действие: както при условната команда

вложена условна команда:

```
if(условие1)
    if(условие2)
        команда1;
    else команда2;
else
    if(условие3)
        команда3;
    else команда4;
```

Определяне на вложените команди

- правило: започва се отдолу нагоре и всеки else се комбинира с най-близкия if преди него, с който образуват валидна условна команда

- пример 1:

```
if(условие1)
    if(условие2)
        команда1;
    else команда2;
else
    if(условие3)
        команда3;
    else команда4;
```

пример 2:

```
if(условие1)
    if(условие2)
        команда1;

else
    if(условие3)
        команда3;
    else команда4;
```

Команда за избор на вариант - действие

- **switch (** *израз* **) {**
  case *константа1* **:** *команди1;* **break;**
  ...
  case *константаN* **:** *командиN;* **break;**
  default : *команди0;*

}

- изчислява се *израз* и стойността му се сравнява последователно с всяка от константите
- при съвпадение с някоя от тях се изпълняват командите след тази константа и всички следващи до срещане на **break**
- ако стойността на израза не бъде открита сред константите, се изпълнява частта с **default**, ако има такава

Команда за избор на вариант - пример

- **switch (** *израз* **) {**
  case *константа1* **:** *команди1;* **break;**
  case *константа2* **:**
  case *константа3* **:** *команди23;* **break;**
  case *константа4* **:** *команди4;*
  case *константа5* **:** *команди5;* **break;**
  default : *команди0;*

}

- ако *израз==константа1* ще изпълни *команди1*
- ако *израз==константа2* или *израз==константа3* ще изпълни *команди23*
- ако *израз=константа4* ще изпълни *команди4* и след това *команди5*
- ако *израз=константа5* ще изпълни *команди5*
- в противен случай ще изпълни *команди0*

Цикъл for

- **for (** *инициализация***;** *условие***;** *актуализация* **)**
  *команда***;**

- *инициализация* задава началните стойности; в нея може да се декларират променливи, които ще важат само за цикъла
- *условие* определя до кога ще се повтаря цикъла
- *актуализация* указва какво ще се променя при всяко завъртане на цикъла
- *команда* е произволна команда в C++; ако е повече от една, се използва съставен оператор
- нито една от частите не е задължителна

Действие на командата for

- **for (** *инициализация***;** *условие***;** *актуализация* **)**
  *команда***;**

1. изпълнява се *инициализацията*
2. проверява се *условие*; ако има стойност невярно, цикъла приключва
3. изпълнява се *команда* (тялото на цикъла)
4. изпълнява се *актуализация*
5. отново на стъпка 2 – проверява се *условие* и т.н.

Цикъл while - действие

while (*условие***)**
команда;

while (*условие***)**
{
команда 1;
...
команда N;
}

1. проверява се условието
2. ако върне стойност **true:** изпълнява се командата
3. цикъла се повтаря отново, докато върне стойност **false**
4. ако върне стойност **false:** цикъла се прекратява и се преминава на следващата команда

Цикъл с постусловие

```
do
  команда;
while (условие);

do
{
   команда 1;
   ...
   команда N;
}
while (условие);
```

- команда – произволна команда от езика C / C++.

  Ако са повече от една, се използва съставен оператор
- условие – логически израз определящ до кога ще се повтаря цикъла

Цикъл do-while - действие

```
do
  команда;
while (условие);

do
{
   команда 1;
   ...
   команда N;
}
while (условие);
```

1. изпълнява се командата
2. проверява се условието
3. ако върне стойност **true:** цикъла се повтаря отново
4. ако върне стойност **false:** цикъла се прекратява и се преминава на следващата команда

Вложени цикли

- Когато в тялото на един оператор за цикъл е вложен друг оператор за цикъл, говорим за вложени цикли. Единият се нарича външен, а другият – вътрешен.

Действие

- За всяко завъртане на външния цикъл вътрешния се извърта целия.

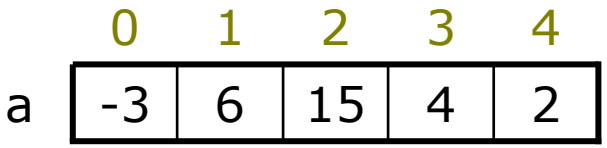
Пример за вложени цикли

Извеждане на таблицата за умножение до 10:

```
for(int i=1;i<=10;i++)
    for(int j=1;j<=10;j++)
        cout<<i<<'*'<<j<<'='<<i*j<<endl;
```

Масив

- структуриран тип данни, представляващ крайна редица от еднотипни елементи с пряк достъп до всеки елемент
- Достъпът до всеки елемент се осъществява посредством името на масива и номера на елемента, наречен индекс
- Масивът е статична структура от данни, защото по време на изпълнението на програмата не могат да се добавят или изтриват елементи



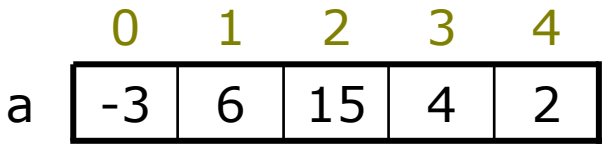
Деклариране на масив

тип име [брой елементи];

- тип – типа на елементите в масива; нарича се базов тип
- име – име на масива
- брой елементи - броят на елементите

- пример:

```
int a[5];
```



Инициализиране на масив

- директно

```
int a[5];
a[0]=2;
...
a[4]=-3;
```

- при декларацията

```
int a[5] = {2, 3, 8, 5, -3};
```

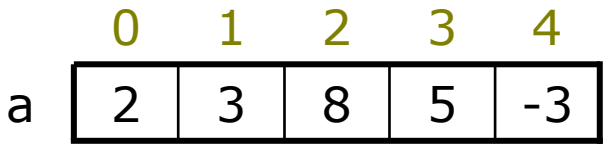
- при декларацията без указване на размер на масива

```
int a[] = {2, 3, 8, 5, -3};
```

- от клавиатурата

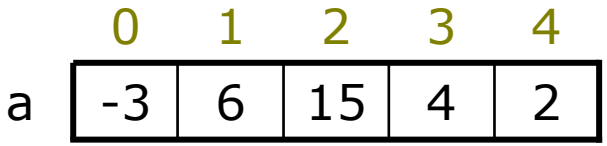
```
int a[5], i;

for(i=0; i<5; i++)
{
  cout<<"a["<<i<<"]=";
  cin>>a[i];
}
```



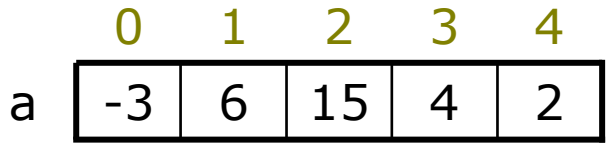
Сортиране чрез пряка селекция

1. Обхождаме целия масив и намираме най-малкия елемент
2. Записваме го на първа позиция, а първият елемент – на неговото място
3. Обхождаме целия останал масив и намираме най-малкия елемент
4. Записваме го на втора позиция, а вторият елемент – на неговото място
5. ...
6. Обхождаме останалият масив и намираме най-малкия елемент
7. Записваме го на позиция `n-1`, а елемент `n-1` – на неговото място



Сортиране на пряка размяна (метод на мехурчето)

1. Сравняваме първия и втория елемент
2. Ако първия е по-голям от втория, им разменяме местата
3. Повтаряме същото за втория и третия, третия и четвъртия и т.н.
4. Така най-големия елемент става последен
5. Повтаряме същото за всички елементи от 1-вия до n-1-вия, после до n-2-рия и т.н.



Низ

- последователност от краен брой елементи от символен тип се нарича символен низ (или просто низ)
- низът представлява всъщност масив от символи, така че за него са приложими всички операции и алгоритми за масиви
- в един низ могат да бъдат съхранявани низови константи с различна дължина
- за указване на край на низа се използва символа '\0'

Низови константи

- низовите константи се ограждат с кавички:
  - "Това е низ" - низ съдържащ текст
  - "125.6" - низ съдържащ цифри
  - "\r\n\t\a" - низ съдържащ управляващи символи
  - "" - празен низ

Деклариране на низ

char име [дължина];

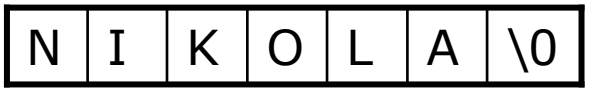
- име – указва името на низа
- дължина - указва дължината на низа. Трябва да е равна на максималният брой символи които ще записваме в низа + 1 (за да можем да запишем символа за край на низ)

- пример:

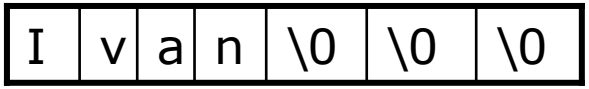
```
char str[20];
```

Инициализиране на низ

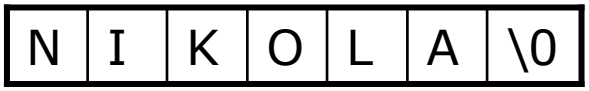
- чрез символни константи
  char ime[7] = {'N','I','K','O','L','A'};



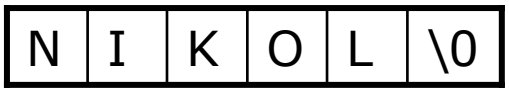
- char ime[7] = {'I','v','a','n'};



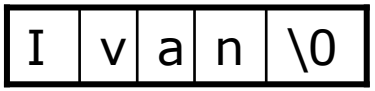
- чрез низови константи
  char ime[7] = "NIKOLA";



- без указване на дължината на низа
  char ime[] = {'N','I','K','O','L'};



- char ime[] = "Ivan";



Инициализация е допустима само при декларацията на низа!

Операции с низове

- въвеждане от клавиатурата - до натискане на клавиш за нов ред, интервал, табулация или до запълване на низа

```
cin >> ime;
```

- извеждане на екрана

```
cout << ime << endl;
cout << "Ivan\n";
```

- достъп до символите от низ

```
cout << ime[0];
ime[2] = 'o';
```

- Директни операции над цели низове не са допустими. Може да се извършват операции само над отделните символи от низа

Деклариране на функция

- *тип име*(*списък формални параметри*)

```
{
  команди;
}
```

- *тип* – типът на резултата който връща функцията. Не може да е масив или низ. За функции, които не връщат резултат се пише **void**
- име – името на функцията, чрез което ще я извикваме
- *списък формални параметри* – описват името и типа на параметрите, които са необходими на функцията за да работи. Ако няма параметри се оставя празно или се пише **void**
- *команди* – командите, които ще се изпълнят при извикването на функцията

Извикване на функцията

- име(*фактически параметри*);

- име – името на функцията която ще се изпълнява
- фактически параметри – набор от стойности, подавани към формалните параметри на функцията. Трябва да им съответстват по брой и тип.

- Действие:
- Изчисляват се фактическите параметри и се подават като стойности на съответните формални параметри на функцията. Тя се изпълнява. Връщаният от нея резултат се замества на мястото на извикването и.

- Пример: PrintSum(2, 18/6);

Видове параметри

- предавани по стойност – промяната им във функцията остава само там; при извикването на функцията могат да бъдат променливи, константи, изрази. Декларират се така:

  тип1 име1, тип2 име2, ...

- предавани по адрес – могат да бъдат само променливи. Промяната на стойността им във функцията води до промяна на стойността на съответните променливи във извикващата я функция. Декларират се така:

  тип1 ***&****име1, тип2* ***&****име2, ...*

Връщане на резултат

- **return** *израз*;

- израз – израз от същия тип като типа на функцията или тип който може да бъде преобразуван към него.

- действие:
- Изчислява се израза, функцията завършва изпълнението си и пресметнатата стойност се замества на мястото на извикването на функцията

Локални променливи

- декларирани някъде **във** функцията
- видими са от мястото на деклариране до **края на функцията или блока** в който се намират
- съществуват докато функцията завърши изпълнението и

Глобални променливи

- декларирани са **извън** всички функции
- видими са от мястото на деклариране до **края на програмата**
- съществуват докато завърши изпълнението на програмата

Край

за тази година...