

Лекция 4.

Обработка на данните. Елементарна обработка

*Как променливите в
програмите получават
стойности и как можем
да разберем какви са те?*

Компютърен модел на обработката на данните

Моделирането на обработката на данните е вторият основен елемент при създаване на компютърни алгоритми

Модели на обработката на данните:

- абстрактен модел
- знаков модел
- физически модел

Абстрактен модел на обработката на данните

Абстрактният модел (АМ) се основава на съответния алгоритъм, за който се изгражда компютърният модел и точно на тези стъпки от описанието на алгоритъма, които определят правилата за обработка на данните

- **Предпоставки:**

Съществуват безброй много практически задачи, които могат да бъдат решени с компютър → безброй много съответни алгоритми

- **Цел:**

Да се определят такива базови алгоритми, които са достатъчно универсални, че чрез тях да може да се опише всеки алгоритъм

- **Изисквания към базовите алгоритми:**

- да позволяват задаване на параметрите-правила на компютърните алгоритми (правилата за определяне на множествата от входните и изходните данни, както и на междинните резултати)
- да дават възможност за определяне реда на изпълнение на стъпките на алгоритъма

ПРИМЕР: Алгоритъм на Евклид за определяне на НОД

Вход: Числа a, b ; **Изход:** НОД на a, b

Стъпка 1. Пригответе се за работа.

Стъпка 2. Въведете и запомнете числата a и b .

Стъпка 3. Ако $a \neq b$, изпълнете ст. 4, в противен случай – ст. 6.

Стъпка 4. Ако $a > b$, то изчислете $a - b$ и го запомнете като a ,
в противен случай изчислете $b - a$ и го запомнете като b .

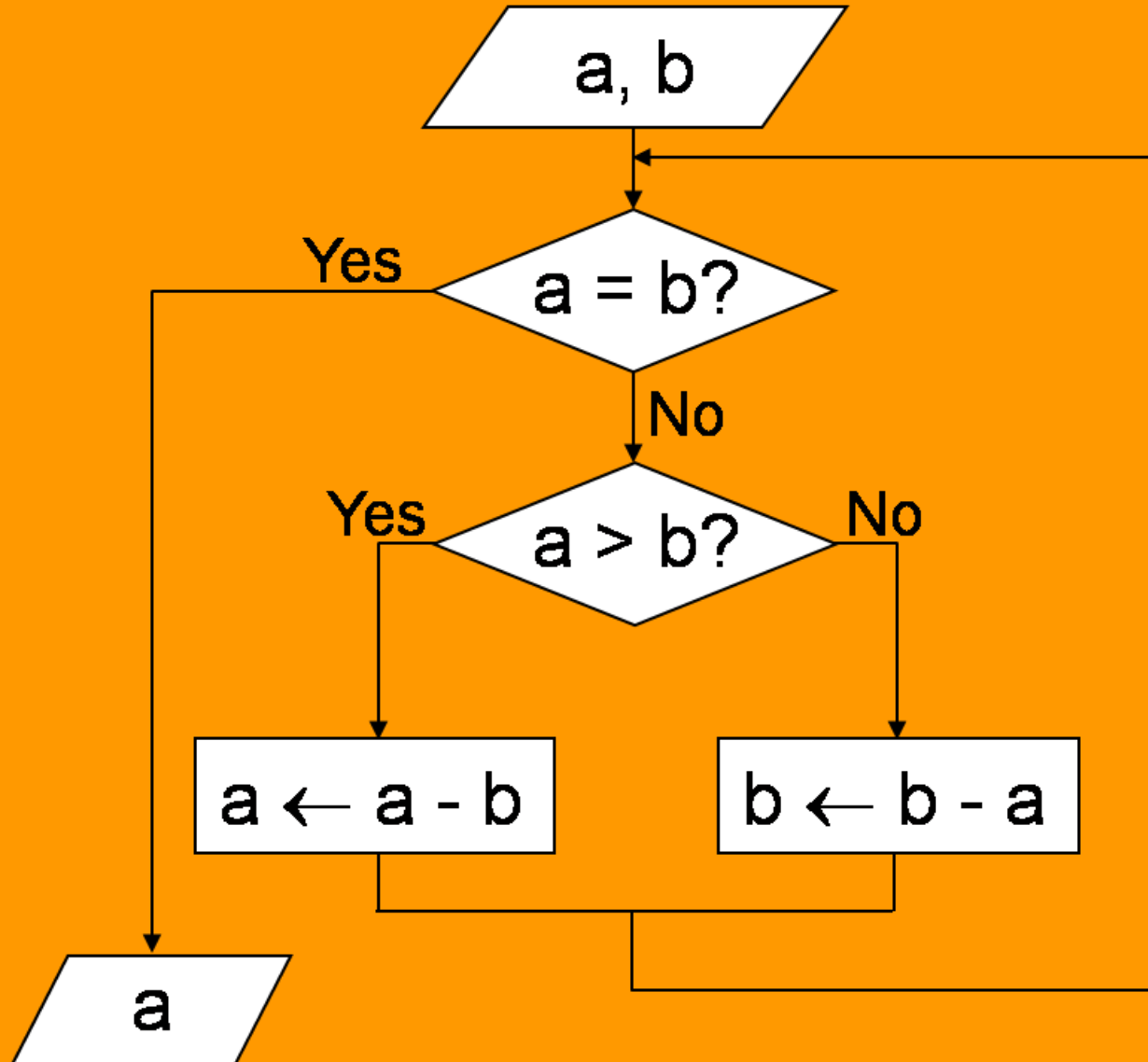
Стъпка 5. Изпълнете ст. 3.

Стъпка 6. Съобщете стойността на a (като резултат).

Стъпка 7. Прекратете работа.

Видове стъпки според тяхното предназначение:

- стъпки 2, 6 и 4 задават **входните и изходните данни** и **междинните резултати**
- стъпки 3 и 4 описват **условно изпълнение** на едно от две различни действия
- указанието от стъпка 5 изисква **повторение на изпълнението** на стъпки от 3 до 5
- стъпки 1, 2 и 3 се **изпълняват последователно** по реда на тяхното срещане



Основни абстрактни модели на обработката на данните

Видове базови алгоритми (на които се основава описанието на всеки алгоритъм) според своето предназначение:

- за определяне на входните данни, изходните данни и междинните резултати
- за последователно изпълнението на група от стъпки
- за избор при изпълнението на група от стъпки
- за повторение на изпълнението на група от стъпки



Абстракцията на обработката на данните се нарича още **абстракция за контрол**, защото служи за определяне на потока на управление на една програма

Знаков и физически модел на обработката на данните

Знаков модел (ЗМ)

Обработката на данните в ЕП се задава с т.нар. **оператори** (команди, *statements*)

- **Особености:** В някои езици на базата на един абстрактен модел за удобство може да са въведени два или повече знакови модела (оператора), осигуряващи различни варианти за задаване на действия от един и същи тип. В такъв случай се казва, че имаме "**натоварване**" в средствата на езика
- **Видове:**
 - **Простите оператори** изискват изпълнението на някакво действие, обикновено над елементи на данните и не включват в себе си, като съставни части, други оператори
 - **Структурните оператори** могат да включват в себе си други оператори.
- **Определяне:** начинът на неговото записване (синтаксис) и начинът на неговото изпълнение (семантика)

Физически модел

Определена последователност от машинни инструкции, а всяка машинна инструкция е реализирана чрез някаква електронна схема в компютъра



Метод на изучаване на абстракциите за контрол (оператори в C++) :

- *АМ*: неговата същност, характеристики и предназначение при обработката на данните
- *ЗМ в C++*: синтаксис и семантика на оператора

Оператори в езика C++. Прости оператори

C++ операторите са програмни елементи, които контролират как и в какъв ред се обработват различните програмни обекти (данни, функции и др.)

Видове прости оператори в C++:

- **Оператори-изрази** — **изчисляват стойността на израз с цел да се използва неговата стойност** (🔊 това означава, че всеки израз е оператор) . Най-често използваните оператори-изрази са:
 - присвояване
 - оператори за въвеждане и извеждане на данни
 - условен оператор-израз (ще бъде разгледан по-късно)
 - активиране на функция (ще бъде разгледан по-късно)
- **Празен оператор** (*Null statement*) — **използва се, когато синтаксиса на C++ изисква оператор, а никакви действия не е необходимо да се предприемат:**

; // Null statement

🔊 В допълнение C++ има прости оператори `goto`, `break`, и `continue` които са изрази, които могат да бъдат използвани за неструктурирани прекъсвания на контролния поток (ще бъдат разгледани по-късно)

Прости оператори в езика C++: Оператор за присвояване – АМ

Абстрактен модел

- **Същност:** Операторът за присвояване (*assignment*) изисква изчислената стойност на някакъв израз да се съхрани на определено място в паметта, обикновено указано чрез име на дадена променлива (т.е. да се присвои на променливата)
- **Приложение:** При задаване на правилата за определяне на стойностите на множеството на входните данни и междинните резултати, а понякога участва и в определянето на изходните резултати

Знаков модел в C++

- **Синтаксис:**
<л-израз> <операция за присвояване> <д-израз>
, където <операция за присвояване> може да бъде:
 - прост (*simple assignment*) =
 - съставен (*compound assignment*):
 - *= /= %= += -= с аритметична (*arithmetic*) операция
 - <<= >>= с операция за изместване (*shift*)
 - &= |= ^= с побитова (*bitwise*) операция

Прости оператори в езика C++: Оператор за присвояване – ЗМ

Знаков модел в C++ (продължение)

- **Семантика:** Изчислява се изразът отлясно (<д-израз>, дясна стойност, *R-value*, *Righthand-value*) на операцията за присвояване и получената стойност се присвоява на израза (променливата), намиращ се отляво (<л-израз>, лява стойност, *L-value*, *Lefthand-value*)

При **съставните оператори за присвояване** трябва да се знае, че формата

$e1 \text{ op} = e2$, е еквивалентна на $e1 = e1 \text{ op} e2$ (op е $+$, $-$, $*$, $...$)

- **Примери:**

(1)

```
int a = 25, b;  
b = a; // прост оператор за присвояване
```

(2)

```
// оператор за присвояване с операция събиране  
int nNumber=1;  
nNumber += 3; // nNumber вече има стойност 4
```

(3)

```
b*=2; // е еквивалентна на b=b*2;
```

Прости оператори в езика C++: Оператор за присвояване – особености

- **Особености:**

- (<л-израз> трябва да бъде израз, който може да се променя (обикновено променлива, която не е `const`)
- ако отляво и отдясно на знака за присвояване се среща една и съща променлива, то при изчисляване на израза отдясно се използва старата стойност на променливата (дясна стойност), а след изпълнение на оператора, тя получава вече новата изчислена стойност (лява стойност)
- (C++) операторът за присвояване връща като резултат самия <л-израз> (с новоприсвоената му от оператора за присвояване стойност). Ето защо е възможно:

```
int xcoord, ycoord, zcoord;  
xcoord = ycoord = zcoord = 1.0/dist;
```
- операторите за присвояване са дясно асоциативни. Поради това и поради горното:

```
int a=1, b=2, d=4;  
a += b += d; // a=7, b=6, d=4
```

 – дясно асоциативни
- **съставните оператори за присвояване** не могат да се използват с изброени типове (съобщение за грешка)
- ако <д-израз> и <л-израз> са от тип указател, то в **съставните оператори за присвояване** могат да участват само операции + или -

Прости оператори в езика C++: Оператори за въвеждане и извеждане на данни – АМ

Абстрактен модел

- **Същност:** Твърде рядко програмите реализират “затворени” алгоритми. Те обикновено обменят данни с външната среда – данните, от които те се нуждаят и данните, които трябва да изведат като резултат

Операторът за въвеждане на данни (*input*) позволява въвеждане на данни от външната за компютъра среда към изпълняваната програма

Операторът за извеждане на данни (*output*) позволява извеждане на данни от изпълняваната програма към външната за компютъра среда

- **Приложение:**
 - **Операторът за въвеждане** се използва при задаване на правилото за определяне на множеството от входните данни на алгоритъма. По този начин неговото главно предназначение е осигуряване на свойството масовост на алгоритъма, моделиран със съответната програма. Именно благодарение на него програмата може да се изпълнява за различни множества входни данни
 - **Операторът за извеждане** се прилага за задаване на правилото за определяне на множеството на изходните резултати

Прости оператори в езика C++: Оператори за въвеждане и извеждане на данни

Някои езици имат специфични оператори за вход/изход (*input/output, I/O*). Други езици като C++ нямат такива средства, а вместо това те разчитат на външните за езика библиотеки

Различни възможности за вход/изход в C++:

- (1) C стандартна библиотека (run-time) за директен небуфериран вход/изход (*getchar* и *putchar*)
- (2) ANSI C стандартна библиотека (run-time) за потоков (*stream*) вход/изход (стандартни функции *scanf* и *printf*)
- (3) Директен вход/изход към/от конзолата и портовете (*fread, fwrite*)
- (4) Библиотеката Microsoft Foundation Class Library (MFC)
- (5) Библиотеката от класове Microsoft *iostream* (обекти *cin* и *cout*)



Ще се спрем само на:

- вход/изход към/от конзолата
- стандартните функции *scanf* и *printf* (с примери) и обектите *cin* и *cout* (задълбочено)

Оператори за въвеждане и извеждане на данни

Функциите *getchar* и *putchar* са дефинирани в стандартната библиотека *stdio.h*

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    int c;
    while((c = getchar()) != EOF )
        putchar(toupper(c));
    fflush(stdout);
    return 0;
}
```

Оператори за въвеждане и извеждане на данни – `printf`

Функциите `scanf` и `printf` са дефинирани в стандартната библиотека `stdio.h`

- **Пример:**

```
#include <stdio.h>
main() {
    int sum;
    sum = 500 + 15;
    printf("The sum of 500 and 15 is %d\n", sum);
}
```

- **Резултат от изпълнение на програмата:**

The sum of 500 and 15 is 515

- **Особености:**

- `%` – специален символ в C++, използва се за извеждане стойност на променлива `printf` предизвиква извеждане на текста, зададен като първи аргумент, докато срещне `%`, тогава извлича втория аргумент (в примера `sum`), извежда неговата стойност и продължава
- `d` – записан след `%` указва, че се очаква съответният аргумент да има целочислена стойност (в примера стойността на `sum` е 515 – тя се извежда)
- `\n` – извежда нов ред

Оператори за въвеждане и извеждане на данни – scanf

- **Пример:**

```
#include < stdio.h >
main()
{
    int sum;
    char letter;
    float money;
    printf("Please enter an integer value ");
    scanf("%d", &sum );
    printf("Please enter a character ");
    /* the leading space before the %c ignores space characters
    in the input */
    scanf(" %c", &letter );
    printf("Please enter a float variable ");
    scanf("%f", &money );
    printf("\nThe variables you entered were\n");
    printf("value of sum = %d\n", sum );
    printf("value of letter = %c\n", letter );
    printf("value of money = %f\n", money );
}
```

- **Особености** – *scanf* има два аргумента:

- (1) низ, който указва какъв тип данни се очаква да бъде въведен (char, int, или float)
- (2) променлива от този тип, която ще получи като стойност въведената такава

Оператори за въвеждане и извеждане на данни – форматиращи символи за `printf` и `scanf`

- Форматиращи символи за **курсора**:

<code>\n</code>	нов ред (<i>newline</i>)
<code>\t</code>	табулация (<i>tab</i>)
<code>\r</code>	край на ред (<i>carriage return</i>)
<code>\f</code>	нов ред (<i>form feed</i>)
<code>\v</code>	вертикална табулация (<i>vertical tab</i>)
<code>\a</code>	звънец (<i>bell</i>)
<code>\p</code>	нова страница (<i>new page</i>)
<code>\\</code>	“ \ ”

- Форматиращи символи при **въвеждане и извеждане на стойности**:

<code>%d</code>	цяло в десетична бройна система
<code>%c</code>	СИМВОЛ
<code>%s</code>	НИЗ
<code>%f</code>	float
<code>%e</code>	double

fread

- `int fread (void * buffer, size_t size, size_t count, FILE * stream);`
- Параметри.
- `buffer`
указател към структурата цел с обем най-малко (`size*count`) bytes.
- `size`
размер в байтове на всеки прочетен елемент.
- `count`
брой елементи за четене.
- `stream`
указател към отворен файл.

```
#include <stdio.h>
#include <stdlib.h>
main () {
    FILE * pFile;
    long lSize;
    char * buffer;
    pFile = fopen ( "myfile.txt" , "rb" ); отворяне в режим четене
    двоично
    if (pFile==NULL) exit (1);
    fseek (pFile , 0 , SEEK_END);
    lSize = ftell (pFile); // функцията ftell определя размера на файла
    rewind (pFile);
    // заделяне на памет динамично за целия файл
    buffer = (char*) malloc (lSize);
    if (buffer == NULL) exit (2);
    // копиране на файла в буфера
    fread (buffer,1,lSize,pFile);
    fclose (pFile);
    free (buffer);
    return 0;
}
```

fwrite

```
size_t fwrite ( const void * buffer, size_t size,  
size_t count, FILE * stream );
```

- Параметри.
- buffer
указател към структурата цел с обем най-малко (size*count) bytes.
- size
размер в байтове на всеки прочетен елемент.
- count
брой елементи за четене.
- stream
указател към отворен файл с права за запис.

```
#include <stdio.h>

main ()
{
    FILE * pFile;
    char buffer[] = "This buffer contains 34
characters";
    pFile = fopen ("myfile.txt" , "w");
    fwrite (buffer , 1 , 34 , pFile);
    fclose (pFile);
    return 0;
}
```

Прости оператори в езика C++: Оператор за извеждане на данни – 3М на *cout*

Обектите `cin` и `cout` са дефинирани в стандартната библиотека `iostream.h`, като обекти на т.нар. `iostream` класове

Знаков модел на `cout` в C++

- **Синтаксис:**

`cout << <израз1> << <израз2> << ... << <изразk>;`

- **Семантика:** Извежда последователно стойностите на изброените изрази на стандартното изходно устройство

- **Примери:**

(1)

`cout << Days << "\n";` //отпечата стойността на `Days` и нов ред

(2)

`cout<<"The coords: x "<<xcoord<<" , y "<<ycoord<<" , z "<<zcoord;`

- **Особености:**



Обектите `cin`, `cout` **са несъвместими с Windows ГПИ**



`endl` е специален клас, който поставен като аргумент на `cout` извежда нов ред

Прости оператори в езика C++: Оператори за въвеждане на данни – ЗМ за *cin*

Знаков модел в C++

- **Синтаксис:**

```
cin >> <variable1> >> <variable2> >> ... >> <variablek>;
```

- **Семантика:** Въвежда последователно стойностите на изброените променливи от стандартното входно устройство, като всички водещи “бели полета” се игнорират (“бели полета”, "white space" са интервал, нов ред, табулатор и т.н.)

- **Примери:**

(1)

```
cout << Days << "\n"; //отпечата стойността на Days и нов ред
```

(2)

```
double xcoord, ycoord, zcoord;  
cin >> xcoord;  
cin >> ycoord;  
cin >> zcoord;
```

(3)

```
cin >> xcoord >> ycoord >> zcoord;
```

 Стойностите, които се въвеждат се отделят с “бели полета”, въвеждането завършва с *ENTER*

Прости оператори в езика C++: Форматиране при въвеждане/извеждане на данни

За форматиране на данните, които се въвеждат/извеждат с помощта на `cin/cout` са предвидени т.нар. **манипулатори**, които са дефинирани в стандартния заглавен файл `<iomanip>`

М а н и п у л а т о р	П р и м е р	П р е д н а з н а ч е н и е
<code>setbase(int base);</code>	<code>int b=8;</code> <code>cout<<setbase(8)<<b;</code> или просто <code>cout<<oct<<b; //може и hex, dec</code>	О с н о в а н а Б С п р и в ъ в е ж д а н е и и з в е ж - д а н е н а ц е л о ч и с л е н и д а н н и
<code>setw(int n);</code>	<code>int a=2;</code> <code>cout<< setw(7)<<a;</code>	Ш и р и н а н а п о л е т о з а и з в е ж д а н е н а с т о й н о с т т а
<code>setfill(Е c);</code>	<code>cout<<setfill('0')<<setw(7)<<a;</code>	С и м в о л з а з а п ъ л в а н е н а п о л е т о з а и з в е ж - д а н е н а с т о й н о с т т а
<code>setprecision(int n);</code>	<code>float ff=100.634;</code> <code>cout<<setprecision(3)<< ff;</code>	Т о ч н о с т п р и и з в е ж - д а н е н а с т о й н о с т с п л а в а щ а т о ч к а



За да използваме в програмата `<iomanip>`:

```
#include <iostream>
#include <iomanip>
using namespace std;
```



```
#include <iostream>
#include <iomanip>

int main () {
    double f =3.14159;
    std::cout << std::setprecision(5) << f << '\n';
    std::cout << std::setprecision(9) << f << '\n';
    std::cout << std::fixed;
    std::cout << std::setprecision(5) << f << '\n';
    std::cout << std::setprecision(9) << f << '\n';
    return 0;
}
```

Результат

3.1416

3.14159

3.14159

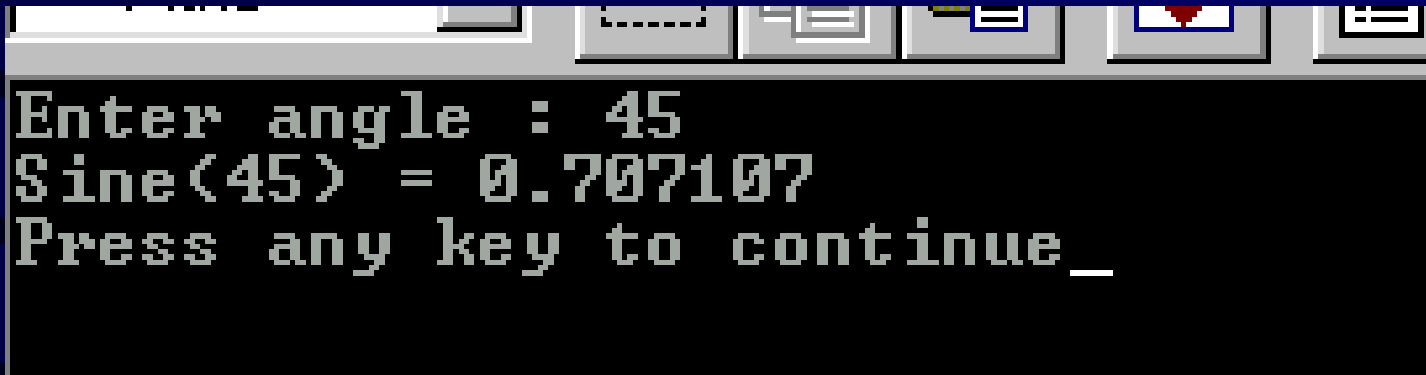
3.14159000

Пример

Програма, която преобразува градусовата мярка на ъгъл в радиани и намира неговия синус:

```
//тестваща програма за простите оператори
#include <iostream.h>
#include <math.h>
void main()
{
    double angle;
    double sine;
    cout << "Enter angle : ";
    cin >> angle;
    sine = sin(angle*3.141592/180.0);
    cout << "Sine(" << angle << ") = " << sine << endl;
}
```

Изпълнение на програмата

A screenshot of a DOS-style command window. The window has a title bar with several icons on the right. The text inside the window is displayed in a monospaced font. It shows the user entering the angle 45, the program calculating the sine of 45 degrees as 0.707107, and then prompting the user to press any key to continue.

```
Enter angle : 45  
Sine(45) = 0.707107  
Press any key to continue_
```

Входно изходни операции върху файлове

Работата с файлове се реализира, чрез потоци.

Потоците (streams) са важна част от всяка входно-изходна библиотека. Те намират своето приложение, когато програмата трябва да "прочете" или "запише" данни от или във външен източник на данни – файл, други компютри, сървъри и т.н. Терминът вход (input) се асоциира с четенето на информация, а терминът изход (output) – със записването на информация.

В C++ работата с файлове може да се използват следните класове:

- ofstream: за запис във файл
- ifstream: за четене от файл
- fstream: и запис и четене

Потоци

Двоични потоци

Двоичните потоци работят с двоични (бинарни) данни. Това ги прави универсални и тях може да ползваме за четене на информация от всякакви файлове (картинки, музикални и мултимедийни файлове, текстови файлове и т.н.).

Текстови потоци

Текстовите потоци са много подобни на двоичните, но работят само с текстови данни или по-точно с поредици от символи (`char`) и стрингове (`string`). Използват се за работа с текстови файлове.

Запис в текстов файл

```
#include <iostream>
#include <fstream>
using namespace std;

int main () {
    ofstream myfile;
    myfile.open ("example.txt");
    myfile << "Writing this to a file.\n";
    myfile.close();
    return 0;
}
```

Отваряне на файл

Използва се метод `open` (име на файл, режим) ;

```
ofstream myfile;  
myfile.open ("example.bin", ios::out | ios::app |  
ios::binary);
```

`ios::in` – Отваря файла за четене.

`ios::out` – Отваря файла за запис, като изтрива съдържанието му, ако той съществува.

`ios::app` – Отваря файла за запис (добавяне) в края на файла.

`ios::ate` – Отваря файла за запис и установява `put` указателя в края на файла, като предишното съдържание се запазва. Указателят може да се премества на произволни места.

`ios::trunc` – Отваря файл, ако той съществува изтрива съдържанието му.

`ios::nocreate` – Ако файлът съществува той се отваря. Не се създава файл с посоченото име.

`ios::noreplace` – Създава и отваря файл с посоченото име, само ако такъв файл не съществува.

`ios::binary` – Превключва режима за достъп до файл от текстов в двоичен.

Четене от текстов файл

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
    string line;
    ifstream myfile ("example.txt");
    if (myfile.is_open())
    {
        while ( getline (myfile,line) )
        {
            cout << line << '\n';
        }
        myfile.close();
    }
    else cout << "Unable to open file";
    return 0;
}
```