CURSUL 8

- **`type_info` şi `cast`**
- Biblioteci standard
  - Biblioteca **`string`**
  - Ierarhia de clase ios, Ierarhia de clase streambuf;
    - Clasa de baza ios: date membru(flags), functii de acces, operatori.
    - Clasele istream, ostream: Functii de intrare, Functii de iesire, Operatori.
    - Clasele ifstream, ofstream, fstream: Constructori destructori;
    - Manipulatori;

Operatorul **`typeid`**

- Operatorul **`typeid`** permite determinarea tipului unui obiect la momentul execuției
- Syntaxa

  `typeid(` ***`type-id`*** **`)`**

  `typeid(` ***`expresie`*** **`)`**
- Rezultatul este referință la un obiect al clasei **`type_info`** (fişierul **`<typeinfo>`**)
- Dacă se aplică la un ***`type-id`*** atunci returnează o referință la **`type_info`** ce reprezintă numele tipului(***`type-id`***)
- Dacă se aplică la o expresie atunci rezultatul este referință la **`type_info`** ce reprezintă tipul obiectului denotat de expresie

Operatorul **typeid**

- Clasa type_info descrie informații relative la tip; aceste sunt generate de compilator. Obiectele clasei conțin pointer la numele tipului:

```
class type_info {
public:
   virtual ~type_info();
   int operator==(const type_info& rhs) const;
   int operator!=(const type_info& rhs) const;
   int before(const type_info& rhs) const;
   const char* name() const;
   const char* raw_name() const;
private:
   ...
};
```

Operatorul **typeid**

```
void f(Figura& r, Figura *p){
     typeid(r)    //tipul obiectului referit de r
     typeid(*p)   //tipul obiectului pointat de p
     typeid(p)    //tipul pointerului: Figura*
     typeid(p) == typeid(Figura*)
     typeid(*p) == typeid(Figura)
}

class Base { ... };

class Derived : public Base { ... };

void f()
{
   Derived* pd = new Derived;
   Base* pb = pd;
   ...
   const type_info& t = typeid(pb);     // tipul pointerului Base*
   const type_info& t1 = typeid(*pb);   // tipul obiectului pointat de pb
   ...
}
```

Operatorul **typeid**

```
#include <typeinfo>
// utilizarea  metodei type_info::typeid()
//si static_cast<T>(exp) in loc de virtual

double f(RxR *pa)
{
    if (typeid(*pa) == typeid(RxR))
       return static_cast<RxR *>(pa)->modul();
    if (typeid(*pa) == typeid(RxRxR))
       return static_cast<RxRxR *>(pa)->modul();
    return 0.0;
}
```

Ierarhia cont - Operatorul **typeid**

```
class Cont
{
public:
   ~Cont();
   Cont(char* = NULL, char* = NULL, double = 0);
   void afiseaza() const;
   virtual void calculeazaDob() = 0;
protected:
   Persoana titular;
   double sold;
};
```

Ierarhia cont - Operatorul **typeid**

```
class ContDepozit : public Cont
{
public:
    ContDepozit(char* = NULL, char* = NULL, double = 0);
    virtual void calculeazaDob();
};

class ContCredit : public Cont
{
public:
    ContCredit(char* = NULL, char* = NULL, double = 0);
    virtual void calculeazaDob();
};
```

Ierarhia cont - Operatorul **typeid**

```
// in cazul virtual calculeazaDob
void actualizeazaSold(Cont* pc)
{
    pc->calculeazaDob();
}

// calculeazaDob() nu este virtuala
void actualizeazaSold(Cont* pc)
{
    if (typeid(*pc) == typeid(ContDepozit))
        static_cast<ContDepozit*>(pc)->calculeazaDob();
    if (typeid(*pc) == typeid(ContCredit))
        static_cast<ContCredit*>(pc)->calculeazaDob();
}
```

Static cast

```
int cmp1(void *p, void *q)
{
   return strcmp(static_cast<Pers1*>(p)->nume,
   static_cast<Pers1*>(q)->nume);
}
int cmp2(void *p, void *q)
{
   return static_cast<Pers1*>(p)->id ==
   static_cast<Pers1*>(q)->id;
}
```

Static cast

```
int cmp1(const void *p, const void *q)
{
return strcmp(const_cast<const Pers2*>(p)->nume,
const_cast<const Pers2*>(q)->nume);
}
int cmp2(const void *p, const void *q)
{
return const_cast<const Pers2*>(p)->id ==
const_cast<const Pers2*>(q)->id;
}
```

Dynamic cast

- Se utilizează când corectitudinea unei conversii nu poate fi determinată de compilator
- Sintaxa:

  `dynamic_cast<T*>(p)`
- Semantica:
  - Dacă obiectul pointat de p este de clasă T sau de clasa de bază T atunci returnează un pointer de tip T* la acest obiect; în caz contrar returnează 0

Dynamic cast

```
std::vector<Figura*> v[20];
//…
int nrCercuri = 0;
int nrDrept = 0;
for (i=0; i<v.size(); i++)
{
   if (dynamic_cast<Cerc*>(v[i])){
      v[i]->arie();
      nrCercuri++;
   }
   if (dynamic_cast<Dreptunghi*>(v[i])){
      v[i]->arie();
      nrDrept++;
   }
}
```

Dynamic cast

```
class A { virtual void f(); };
class B { virtual void g(); };
class D : public virtual A, private B {};
void g()
{
D d;
B* bp = (B*)&d;// cast necesar datorita protectiei
A* ap = &d;// derivare publica, nu e nevoie de cast
D& dr = dynamic_cast<D&>(*bp); // !OK
ap = dynamic_cast<A*>(bp); // !OK
bp = dynamic_cast<B*>(ap); // !OK
ap = dynamic_cast<A*>(&d); // OK
bp = dynamic_cast<B*>(&d); // !OK
}
```

Biblioteca **`string`**

- Un string este o secventa de caractere
- Biblioteca string contine :
  - Operatii pentru manipularea sirurilor
  - Asignare
  - Comparari
  - Adaugare caractere la sfarsit (append)
  - Concatenare
  - Cautare subsiruri
- Se poate folosi si stilul C: sirurile sunt tablouri de **`char`** iar bibliotecile corespunzatoare au prefixul c: **`cstring`**, **`cctype`**, **`cwtype`**, **`cstdlib`**

Structura char_traits

- **char_traits** este o specializare pentru E = **char** a template-ului:

```
template <class E>
struct char_traits{
   typedef E char_type;
   typedef T1 int_type;
   //…
   static int compare(const E *x,
                      const E *y, size_t n);
   static size_t length(const E *x);
   static E *copy(E *x, const E *y, size_t n);
   static int_type eof();
};
```

Clasa basic_string

```
template<class E,
   class T = char_traits<E>,
   class A = allocator<T> >
class basic_string { };

typedef basic_string<char> string;
typedef basic_string<wchar_t> wstring;
```

Clasa basic_string - constructori

```
explicit basic_string();
basic_string(const basic_string& rhs);
basic_string(const basic_string& rhs,
      size_type pos, size_type n);
basic_string(const E *s, size_type n);
basic_string(const E *s);
basic_string(size_type n, E c);
basic_string(const_iterator first, const_iterator
   last);
```

Clasa basic_string - constructori

```
string s0
string s1 = "";
string s2("Facultatea de Informatica");
string s3(s2);
string s4(s2, 2, 3);
string s5(20, 'a');
string s6 = s2;
string s7(p+7, 3); // char* p;
string s8(v.begin(), v.end());
string s9 = 'a'; // eroare
string s10(5); // eroare
```

Clasa basic_string - iteratori

- Iterator: pointer "smart" la un element al unei secvențe, în stare să:
  - ofere elementul la care pointează (operatori * şi ->)
  - pointeze la următorul element (operatori ++)
  - verifce egalitatea a 2 iteratori (operatori ==)

```
iterator begin();
const_iterator begin() const;
iterator end();
const_iterator end() const;
reverse_iterator rbegin();
const_reverse_iterator rbegin()const;
reverse_iterator rend();
const_reverse_iterator rend() const;
```

Clasa basic_string - operatori

```
basic_string& operator=(const basic_string& rhs);
basic_string& operator=(const E *s);
basic_string& operator=(E c);
basic_string& operator+=(const basic_string& rhs);
basic_string& operator+=(const E *s);
basic_string& operator+=(E c);
const_reference operator[](size_type pos) const;
reference operator[](size_type pos);
```

- Acces cu verificarea domeniului de valori:

```
const_reference at(size_type pos) const;

reference at(size_type pos);
```

Clasa basic_string – Alte functii

```
append() asign() insert()
compare() find() rfind()
copy()     swap()
find_first_of() find_first_not_of()
find_last_of() find_last_not_of()
replace() erase() substr()
size()     max_size() resize()
length() empty() capacity()
reserve()
```

Clasa basic_string – Alte functii

```
s.copy(p, n, m); // p = s[m]…
s.insert(s.begin(), '');
s.insert(0, s1);
string s = "accdcde";
int i1 = s.find("cd");              //i1=2
int i2 = s.rfind("cd");             //i2=4
int i3 = s.find_first_of("cd");     //i3=1
int i4 = s.find_last_of("cd");      //i4=5
int i5 = s.find_first_not_of("cd");//i5=0
```

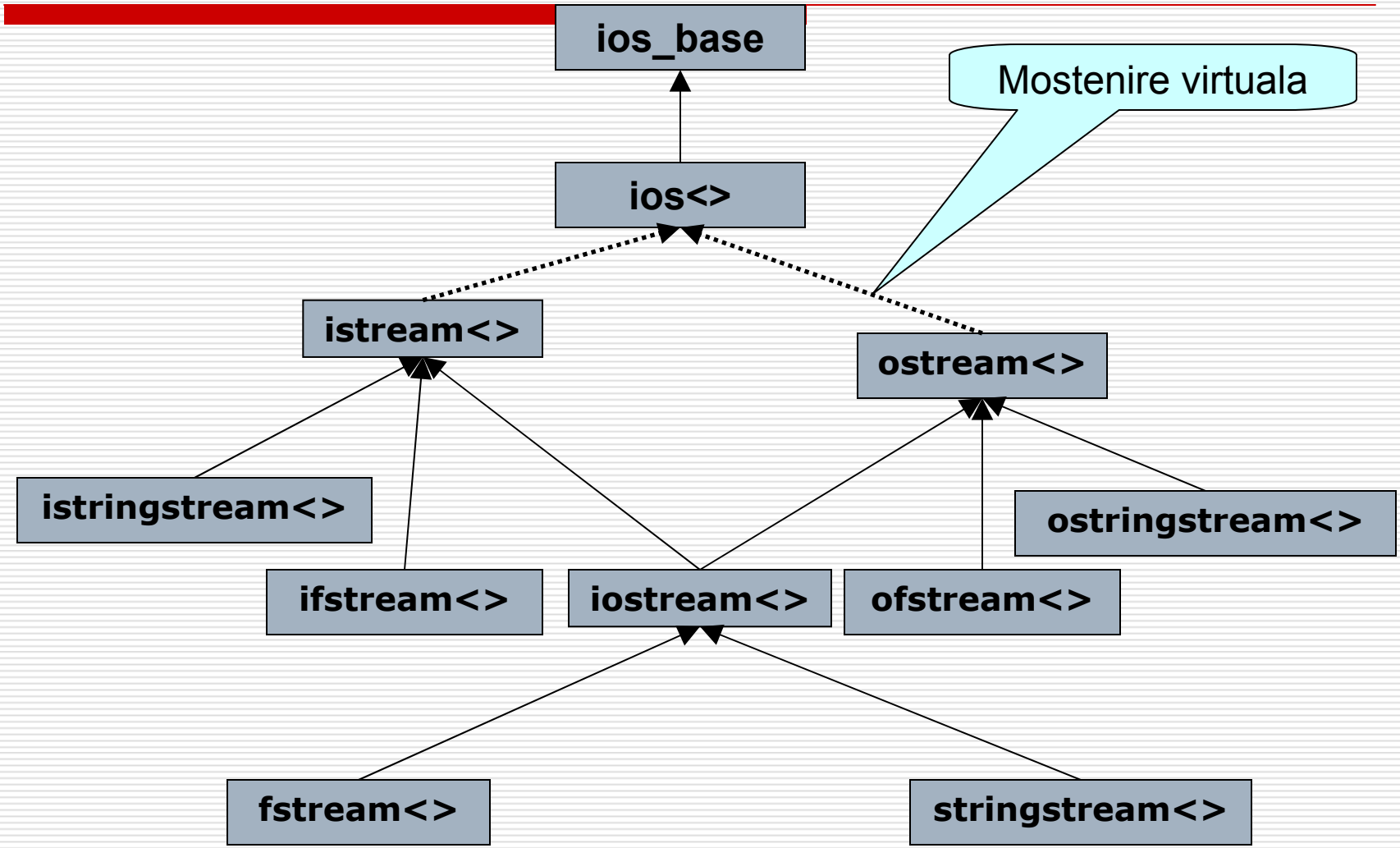
Operatii I/O cu stringuri

```
template<class E, class T, class A>
basic_ostream<E, T>& operator>>( basic_ostream
   <E, T>& os, basic_string<E, T, A>& str);

template<class E, class T, class A>
basic_ostream<E, T>& operator<<( basic_ostream
   <E, T>& os,const basic_string<E, T, A>& str);
```

Operatii I/O cu stringuri

```
template<class E, class T, class A>
basic_istream<E, T>& getline( basic_istream <E,
   T>& is, basic_string<E, T, A>& str);

template<class E, class T, class A>
basic_istream<E, T>& getline( basic_istream <E,
   T>& is, basic_string<E, T, A>& str, E delim);
```

Ierarhia de clase iostream



Ierarhia de clase iostream

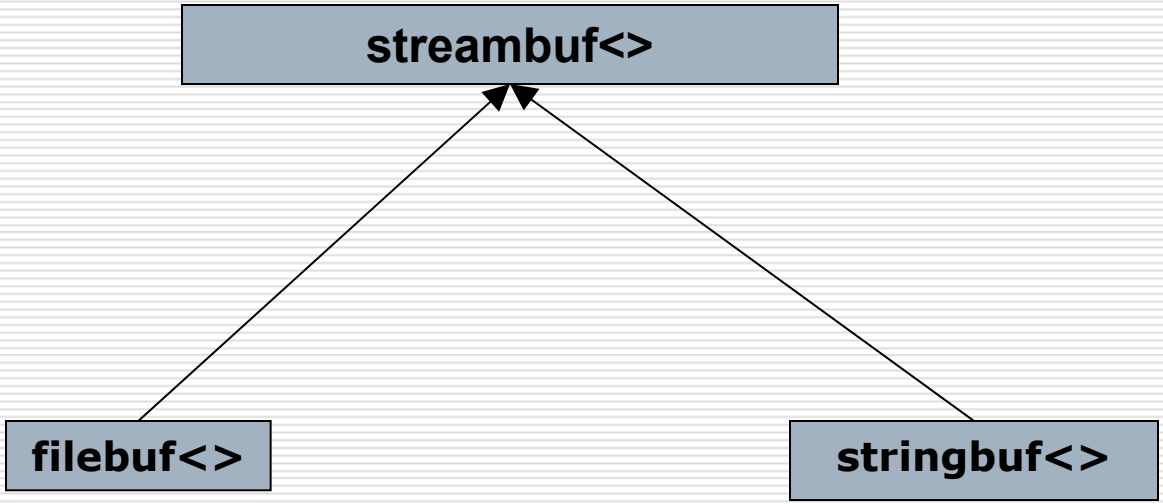
- Clasele cu sufixul <> sunt template- uri parametrizate cu un tip caracter iar numele lor au prefixul *basic_ :*

```
template <class E, class T = char_traits<E> >
class basic_ios : public ios_base {
//…
};
```

- **`char_traits<E>`** contine informatii pentru manipularea elementelor de tip E

```
template <class E, class T = char_traits<E> >
class basic_istream : virtual public basic_ios<E, T> {
   //…
  };
```

Ierarhia de clase streambuf



Clasa basic_iostream

```
template <class E, class T = char_traits<E>>
class basic_iostream :
    public basic_istream<E, T>,
    public basic_ostream<E, T> {
public:
    explicit basic_iostream(basic_streambuf<E, T>* sb);
    virtual ~basic_iostream();
};
```

Fisierul header <iosfwd>

```
typedef basic_ios<char, char_traits<char> > ios;
typedef basic_istream<char, char_traits<char> >
                        istream;
typedef basic_ostream<char, char_traits<char> >
                        ostream;

typedef basic_iostream<char, char_traits<char> >
                        iostream;
typedef basic_ifstream<char, char_traits<char> >
                        ifstream;
typedef basic_ofstream<char, char_traits<char> >
                        ofstream;
typedef basic_fstream<char, char_traits<char> >
                        fstream;
```

Clasa ostream

- Un obiect din ostream (flux de iesire) este un mecanism pentru conversia valorilor de diverse tipuri in secvente de caractere
- In <iostream>:

```
ostream cout; //fluxul standard de iesire
ostream cerr; // mesaje eroare,fara buffer
ostream clog; // mesaje eroare,cu buffer
```

`operator<<` (în ostream)

```
basic_ostream& operator<< (const char *s);
basic_ostream& operator<< (char c);
basic_ostream& operator<< (bool n);
basic_ostream& operator<< (short n);
basic_ostream& operator<< (unsigned short n);
basic_ostream& operator<< (int n);
basic_ostream& operator<< (unsigned int n);
basic_ostream& operator<< (long n);
basic_ostream& operator<< (unsigned long n);
basic_ostream& operator<< (float n);
basic_ostream& operator<< (double n);
basic_ostream& operator<< (long double n);
basic_ostream& operator<< (void * n);
basic_ostream& put(E c);       // scrie c
basic_ostream& write(E *p, streamsize n); // scrie p[0],…,p[n-1]
basic_ostream& flush(); // goleste bufferul
pos_type tellp(); // pozitia curenta
basic_ostream& seekp(pos_type pos); // noua positie pos
basic_ostream& seekp(off_type off, ios_base::seek_dir way);
        // pozitia way (= beg, cur, end) + off
```

Formatare, <iomanip>

```
void main(){
      int i = 10, j = 16, k = 24;
      cout << i << '\t' << j '\t' << k << endl;
      cout << oct << i << '\t' << j << '\t' << k << endl;
      cout << hex << i << '\t' << j << '\t' << k << endl;
      cout << "Introdu 3 intregi: " << endl;
      cin >> i >> hex >> j >> k;
      cout << dec << i << '\t' << j << '\t' << k << endl;
}
/*
10    16    24
12    20    30
A     10    18
Introdu 3 intregi: 11 11 12a
11    17    298
```

Manipulatori

MANIPULATOR	EFECTUL	FISIERUL
`endl`	**Scrie newline**	**iostream**
`ends`	**Scrie NULL in string**	**iostream**
`flush`	**Goleste**	**iostream**
`dec`	**Baza 10**	**iostream**
`hex`	**Baza 16**	**iostream**
`oct`	**Baza 8**	**iostream**
`ws`	**Ignoră spațiile la intrare**	**iostream**
`skipws`	**Ignoră spațiile**	**iostream**
`noskipws`	**Nu ignoră spațiile**	**iostream**
`showpoint`	**Se scrie punctul zecimal şi zerourile**	**iostream**
`noshowpoint`	**Nu se scriu zerourile, nici punctul**	**iostream**
`showpos`	**Scrie + la numerele nenegative**	**iostream**
`noshowpos`	**Nu scrie + la numerele nenegative**	**iostream**
`boolalpha`	**Scrie true si false pentru bool**	**iostream**
`noboolalpha`	**Scrie 1 si 0 pentru bool**	**iostream**

Manipulatori

MANIPULATOR	EFECTUL	FISIERUL
`scientific`	**Notația ştiințifică pentru numere reale**	**iostream**
`fixed`	**Notația punct fix pentru numere reale**	**iostream**
`left`	**Aliniere stânga**	**iostream**
`right`	**Aliniere dreapta**	**iostream**
`internal`	**Caract. de umplere intre semn si valoare**	**iostream**
`setw(int)`	**Seteaza dimensiunea câmpului**	**iomanip**
`setfill(char c)`	**Caracterul c înlocueşte blancurile**	**iomanip**
`setbase(int)`	**Setarea bazei**	**iomanip**
`setprecision(int)`	**Seteaza precizia in flotant**	**iomanip**
`setiosflags(long)`	**Seteaza bitii pentru format**	**iomanip**
`resetiosflags(long)`	**Reseteaza bitii pentru format**	**iomanip**

Clasa istream

- Un obiect din **`istream`** (flux de intrare) este un mecanism pentru conversia caracterelor in valori de diverse tipuri
- In <iostream>:

  `istream cin; //fluxul standard de intrare`

- In **`basic_istream`** este definit **`operator>>`** pentru tipurile fundamentale

Functii utile in istream

```
streamsize gcount() const;
int_type get();
basic_istream& get(E& c);
basic_istream& get(E *s, streamsize n);
basic_istream& get(E *s, streamsize n, E delim);
basic_istream& get(basic_streambuf<E, T> *sb);
basic_istream& get(basic_streambuf<E, T> *sb, E delim);
basic_istream& getline(E *s, streamsize n);
basic_istream& getline(E *s, streamsize n, E delim);
basic_istream& ignore(streamsize n = 1,
                      int_type delim = T::eof());
int_type peek(); // citeste fara a-l extrage
basic_istream& read(E *s, streamsize n);
streamsize readsome(E *s, streamsize n); // peek cel mult n
basic_istream& putback(E c); // pune c in buferul de intrare
basic_istream& unget(); // pune inapoi ultimul citit
pos_type tellg();
basic_istream& seekg(pos_type pos);
basic_istream& seekg(off_type off, ios_base::seek_dir way);
int sync();  // flush input
```

Fisiere

- Intr-un program C++ fluxurile standard **`cin`**, **`cout`**, **`cerr`**, **`clog`** sunt disponibile; corespondenta lor cu dispozitivele (fisierele) standard este facuta de sistem

- Programatorul poate crea fluxuri proprii – obiecte ale clasei **`fstream`** (**`ifstream`**, **`ofstream`**). Atasarea fluxului unui fisier sau unui string se face de programator:
  - La declarare cu un constructor cu parametri
  - Dupa declarare prin mesajul **`open()`**

Fisiere

- Constructorii clasei fstream:

```
explicit basic_fstream();
explicit basic_fstream(const char *s,
   ios_base::openmode mode =
   ios_base::in | ios_base::out);
```

- Metode:

```
bool is_open() const;
void open(const char *s,
   ios_base::openmode mode =
   ios_base::in | ios_base::out);
void close();
```

Fisiere de intrare

- Constructorii clasei ifstream:

```
explicit basic_ifstream();
explicit basic_ifstream(const char *s,
   ios_base::openmode mode =ios_base::in);
```

- Metode:

```
bool is_open() const;
void open(const char *s,
   ios_base::openmode mode =
   ios_base::in);
void close();
```

Fisiere de iesire

- Constructorii clasei ofstream:

```
explicit basic_ofstream();
explicit basic_ofstream(const char *s,
   ios_base::openmode which =
   ios_base::out | ios_base::trunc);
```

- Metode:

```
bool is_open() const;
void open(const char *s,
   ios_base::openmode mode =
   ios_base::out | ios_base::trunc);
void close();
```

ios_base::openmode

- **app**, pozitionare la sfarsit inainte de fiecare insertie
- **ate**, pozitionare la sfarsit la prima creare a fluxului
- **binary**, citirea fisierului in mod binar si nu in mod text
- **in**, permite extragerea (fisier de intrare)
- **out**, permite insertia (fisier de iesire)
- **trunc**, truncheaza un fisier existent la prima creare a fluxului ce-l controleaza

Clasa stringstream

- Permite ca stringurile sa fie tratate ca fisiere:

```
explicit basic_stringstream(ios_base::openmode mode
   = ios_base::in | ios_base::out);

explicit basic_stringstream(const
   basic_string<E,T,A>& x, ios_base::openmode mode =
            ios_base::in | ios_base::out);
```

Exemple

```
string comp1(int n, const string& cs){
   ostringstream outs;
   outs << "Eroare(" << n << "):" << cs;
   return outs.str();
}
void scrie_cate_unul(const string& s){
   istringstream ins(s);
   string w;
   while(ins>>w) cout << w << '\n';
}
cout << comp1(3, "test");
scrie_cate_unul("Facultatea de Informatica");
```