Introduction to programming 2014 - 2015

Corina Forăscu

corinfor@info.uaic.ro

http://profs.info.uaic.ro/~introp/

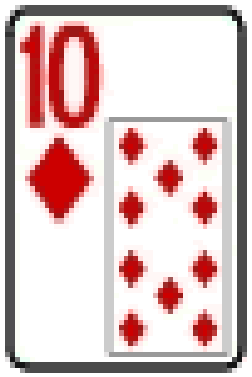
Course 5: agenda

- Struct & union
- File I/O

Data structure

- Data structure: a group of data elements grouped together under one name; the data elements (members), can have different types and different lengths.
- The allocated memory is contiguous; elements are stored in order of declaration of structure.

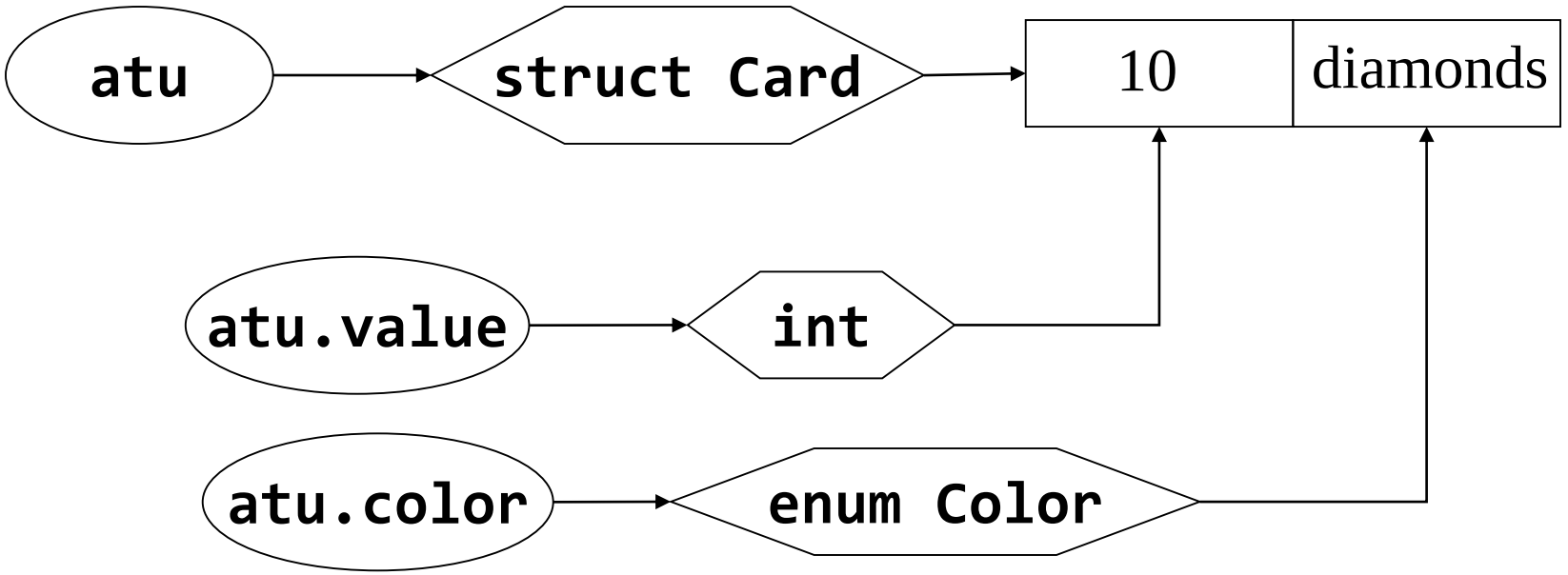
Simple data structure



```
enum Color { hearts, spades, diamonds, clubs};
typedef enum Color Color;
struct Card
{
    int value;
    Culoare cul;
};
```

Simple data structures

```
struct Card trump;
trump.value = 10;
trump.color = diamonds;
```



Simple data structures

```
cout << “The trump is: “ << trump.value;
switch (trump.color)
{
case clubs:
     cout << “ clubs\n”;
     break;
case diamonds:
     cout << “ diamonds\n”);
     break;
// ...
}
```

Synonyms for data structures

- The name **`struct Card`** is too long
- Let's associate a synonym:

```
typedef struct Card
{
    int value;
    Color color;
} Card;
```

- We can declare a variable easier:

```
Card trump;
```

- Now **`Card`** and **`struct Card`** are synonyms

Synonyms for data structures

- with **`typedef`** the structure can be anonymous

```
typedef struct
{
      int value;
      Color color;
} Card;
```

- When you declare a variable, you will write only:

`Card card;`

Complex Data Structures

- A player has a name, a hand of cards and money

```
typedef struct Player
{
        char* name;
        Card hand[4];
        long money;
} Player;
```

- A table has a number and 4 players

```
typedef struct Table
{
        int no;
        Player player[4];
} Table;
```

Complex data structures

- The player **j** gets an 8 of clubs as second card

```
j.hand[1].value = 8;
j.hand[1].color = clubs;
```

- The third player from the **m** table gets a 9 of diamonds as first card

```
m.jucator[2].mana[0].val = 9;
m.jucator[2].mana[0].cul = caro;
```

Alternative data structures

- A geometrical figure is:
  - A point
  - Or a segment
  - Or a circle
  - Or …
- Question: Can we define the **“figure”** type?
- Answer: Yes
  - Using structures: … it’s not economic (*why*?)
  - Using alternative data structures.

Union types
(alternative data structures)

- The **sintax** for union type is the same as **struct** type.
- Operations are the same
- Allow one portion of memory to be accessed as different data types

```
union int_or_float{ int i; float x; };
```

- The size(**sizeof**) of this type is the one of the largest member element
- Each of these members is of a **different** data type.
- all of them are referring to the same location in memory =>the modification of one of the members will affect the value of all of them.
- It is not possible to store different values in them in a way that each is independent of the others.

Alternative data structures

```
typedef struct {
        int x;
        int y;
} point;

typedef struct {
        point A;
        point B;
} segment;

typedef struct {
        point center;
        int radius;
} circle;
```

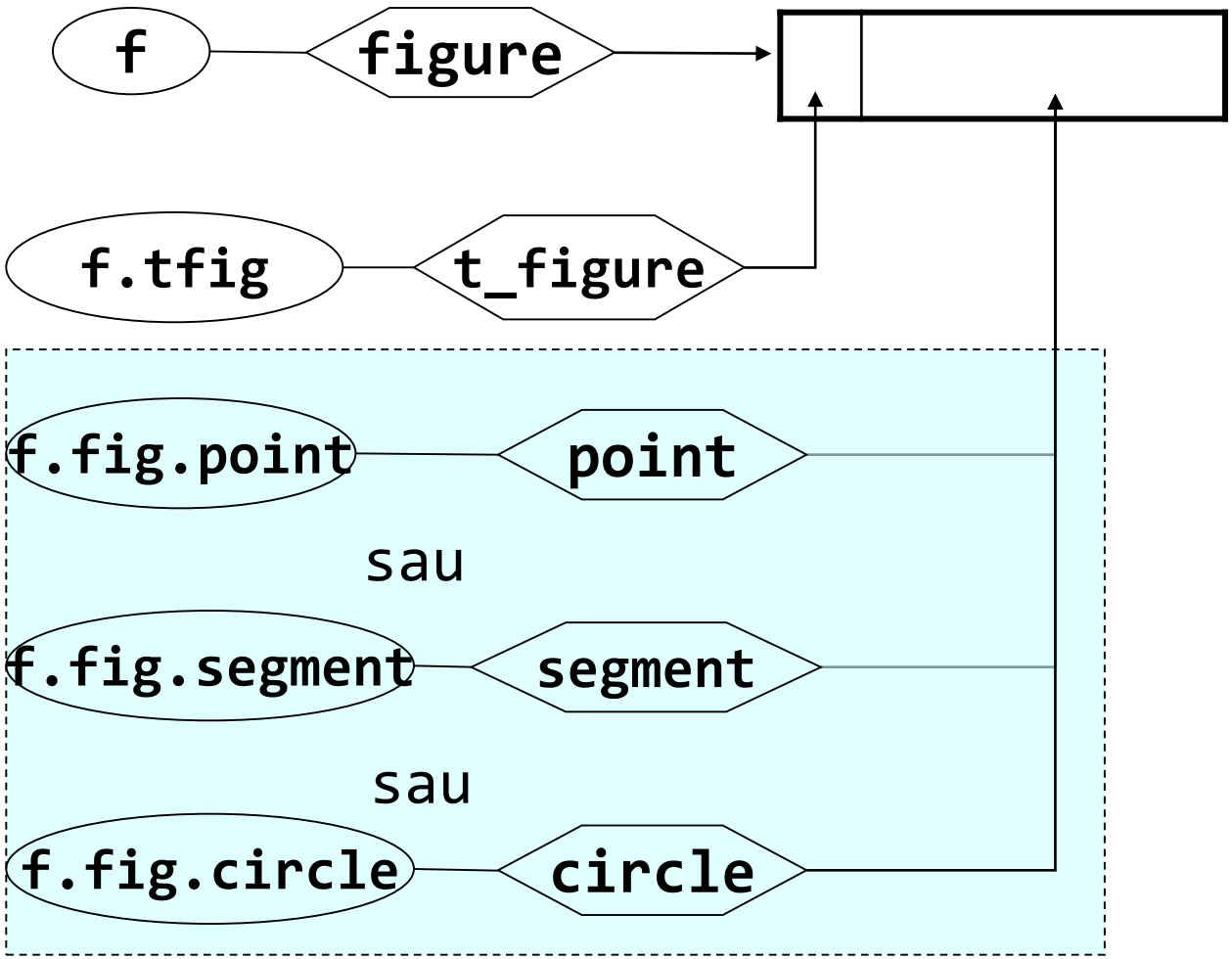
Alternative data structures

```
typedef enum { t_point, t_segment, t_circle }
t_figure;

typedef struct {
      t_figure tfig;
      union {
            point point;
            segment segment;
            circle circle;
      } fig;
} figure;
```

Alternative data structures - variabiles

```
figure f;
```



Alternative data structures– using variables

- Perimeter:

```
double perim(figura f) {
      switch (f.tfig)
      {
      case t_point:
            return 0;
            break;
      case t_segment:
            return segm_length(f.fig.segment);
            break;
      case t_cerc:
            return perim_circle(f.fig.circle);
            break;
      }
}
```

```
int main(){
    figure a_figure;

    circle a_circle = { 5, 5, 100 }; (why?)

    a_figure.tfig = t_circle;
    a_figure.fig.circle = a_circle;  //circle

    cout << “The figure has the perimeter: ";
    cout << perim(a_figure) << “\n”;

    //     segment a_segment = {0,0,3,4};
    //     a_figure.tfig = t_segment;
    //     a_figura.tfig.segment = a_segment;
    // ...
    return 0;
}
```

Anonymous union vs Regular union

Structure with regular union

```
struct book1_t {
  char title[50];
  char author[50];
  union {
    float dollars;
    int yen;
  } price;
} book1;
```

book1.price.dollars
book1.price.yen

Structure with anonymous union

```
struct book2_t {
  char title[50];
  char author[50];
  union {
    float dollars;
    int yen;
  };
} book2;
```

book2.dollars
book2.yen

- *What are the differences? What are the similarities?*

I/O File

- A file can be seen as a “stream” of characters.
- A file has a name
- To access a file, it has to be “opened”
- The system has to know which operations can be made within a file (the programmer send the instructions)
  - Open the file for reading– the file **must** exist
  - Open the file for writing – the file will be created
  - Open the file for adding information – the file exists and will be updated
- After processing, the file will be closed

fstream.h

- Define the classes:
  - `ifstream`: class input streams
  - `ofstream`: class output streams
  - `fstream`: class input/output streams
- Declaration of a stream:

```
ifstream input;
```

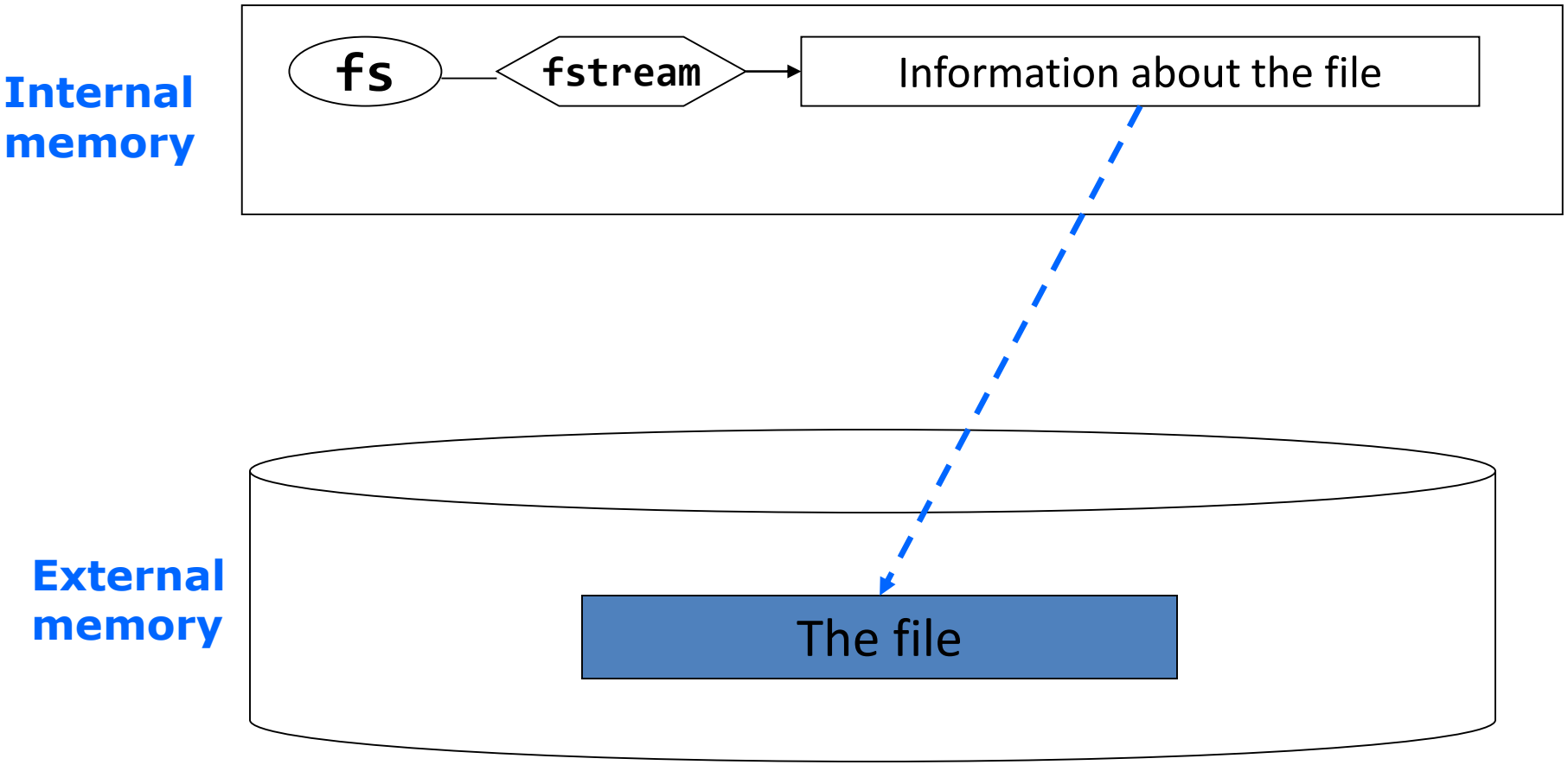
- Link the **stream** with a physical file (on disk):

```
void open(const char *filename, int mode);
```

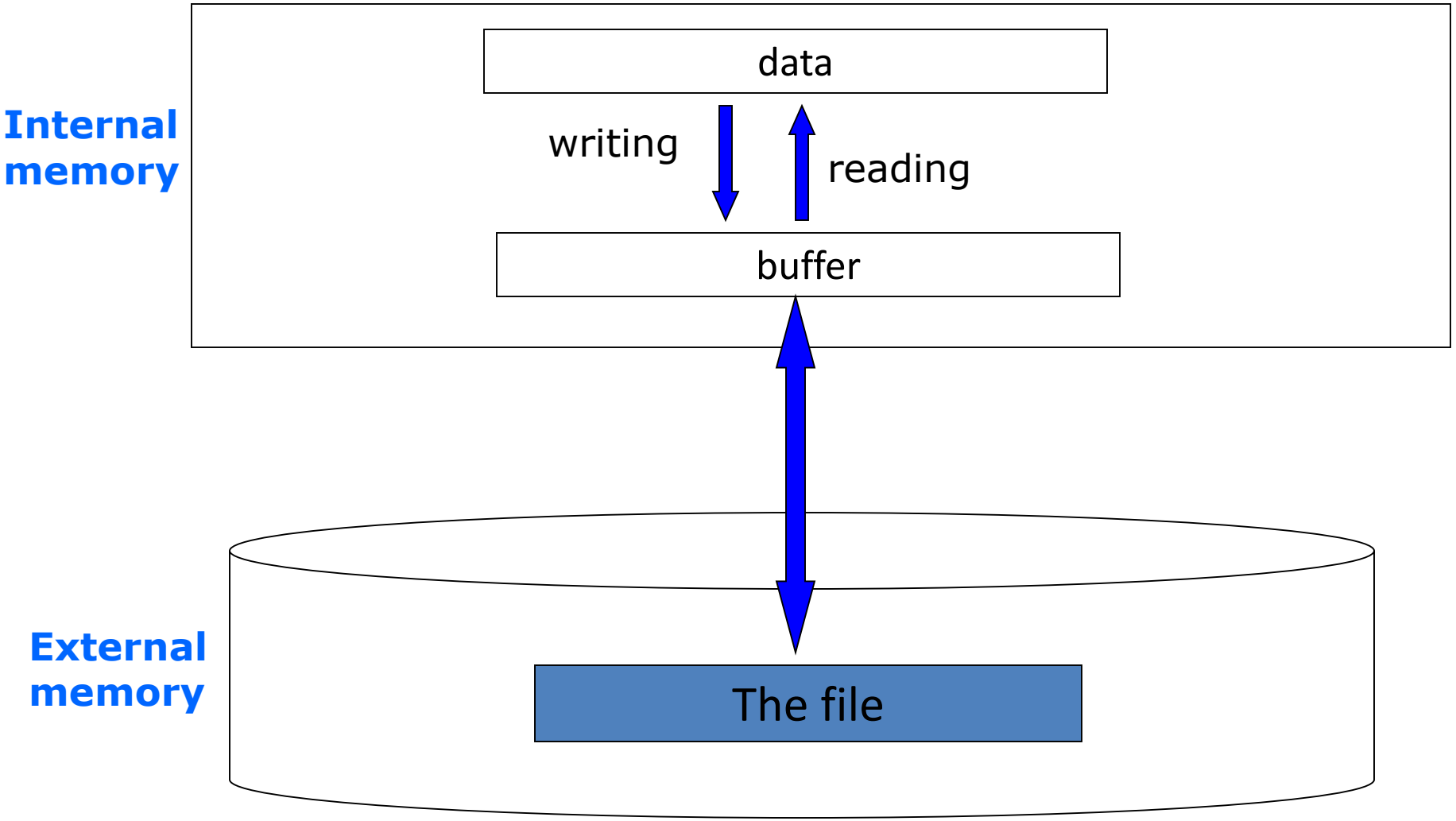
  - *ios::app* – All output operations happen at the end of the file, appending to its existing contents.
  - *ios::ate* – The *output position* starts at the end of the file.
  - *ios::binnary* – Operations are performed in binary mode rather than text.
  - *ios::in* – File open for reading
  - *ios::out* – File open for writing
  - *ios::nocreate, ios::noreplace, ios::trunc*

These flags can be combined with the bitwise OR operator |

I/O File


Internal memory
fs
fstream
Information about the file
External memory
The file

Files– reading/writing



Functions `close()`, `flush()`

```
mystream.close();
```

- This member function takes flushes the associated buffers and closes the file by interrupting every connection between the **file** and the **stream mystream**

```
mystream.flush()
```

- Flushes the buffer: the data from buffer is written in the file

Example 1

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

int main(){
    ofstream  out;
    out.open("d:\\results.txt"); //which is the mode?
    if (!out) {
        cout << «file error!\n";  return 1;
    }
    out << "Ionescu " << 7.50 << endl;
    out << "Popescu " << 9.75 << endl;
    out << "Georgescu " << 8.25 << endl;

    out.close();
    return 0;
}
```

Example 2

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

int main(){
    ifstream  in;
    in.open("d:\\resuts.txt");
    if (!in) {
        cout << «file error!\n";  return 1;
    }
    char nume[20]; float nota;
    for (int i = 1; i <= 3; i++){
        in >> nume >> nota;
        cout << nume << " " << nota << '\n';
    }
    in.close();return 0;
}
```

Binary I/O

```
istream & get(char &ch);
ostream & put(char ch);
```

- **get()** reads a single character from the stream and memorize the value in *ch*
- **put()** write *ch* in the stream

Example 3

```
/*Copy file with transforming for letters from lower case to upper case*/
#include <iostream>
#include <fstream>
using namespace std;

int main(){
    char   c, file_name[128];
    ifstream  ifs;
    ofstream  ofs;
    cout << "\nWrite the name of the file:  "; cin >> file_name;
    ifs.open(file_name);
    if (!ifs) {
          cout << "Opening the file has failed\n";
          return 1;
    }
    ofs.open("d:\\copy.out");
    while (ifs.get(c)) {
          if (islower(c)) c = toupper(c);
          ofs.put(c);
    }
    ifs.close(); ofs.close(); return 0;
}
```

Reading/writing blocks of data

```
istream& read(char * buf, int no);
```

- There are read at most *no* octets from the stream and added in the buffer *buf.*

```
istream& write(const char * buf, int
  no);
```

- Write in stream *no* octets read from buffer *buf*

Example 4

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

int main(){
      float tab[3] = { 7.50, 9.75, 8.25 };
      int i;

      ofstream  out;
      out.open("d:\\marks.dat", ios::out | ios::binary);
      if (!out) { cout << “file error!\n";  return 1; }

      out.write((const char *)&tab, sizeof(tab));
      out.close();
      for (i = 0; i<3; i++) tab[i] = 0.0;

      ifstream  in;
      in.open("d:\\marks.dat", ios::in | ios::binary);
      in.read((char *)&tab, sizeof(tab));

      for (i = 0; i<3; i++) cout << tab[i] << " ";

      in.close();
      return 0;
}
```

Another functions

```
istream &get(char *buf, int no,
                        char delim ='\n');
```

- Read in buffer *buf* at most *no* octets or it can be found character *delim*

```
int get();
```

- Return the next character from stream or EOF if the end of file

```
istream &getline(char *buf, int nr,
                        char delim ='\n');
```

- Beside **get( ),** extract the delimiter from stream

Another functions

```
int eof();
```

- A non-zero value is returned in the case that the end-of-file indicator associated with the stream is set.
- Otherwise, zero is returned.

```
int peek();
```

- Returns the next character in the input sequence, without extracting it.
- The character is left as the next character to be extracted from the stream.

```
istream &putback(char ch);
```

- Return in stream the character *ch.*

Random access function

```
istream &seekg(streamoff offset, seek_dir org);
istream &seekp(streamoff offset, seek_dir org);
```

- Move the pointers *get* şi *put* for the next I/O operation with *offset* octets by *org* .

- *Org* value might be :
  - `ios::beg` – start of file
  - `ios::cur` – current location
  - `ios::end` – end of fie

- Examples(for *get* pointer):
  - move to end of file
    ```
    mystream.seekg(0, ios::end)
    ```
  - Move to previous character
    ```
    mystream.seekg(-1, ios::cur)
    ```
  - Move to beginning of file
    ```
    mystream.seekg(0, ios::beg)
    ```

Other functions

```
streampos tellg();
streampos tellp();
```

- Tell the current position of *get* and *put* pointers.

- I/O state:
  - **eofbit, failbit, badbit**
  - `int eof(); int fail(); int bad(); int good();`

```
void clear(int indicatori=0);
```

- reset error indicators and end-of-file

Exemple 5

```
#include <iostream>
#include <fstream>
using namespace std;
//Write a file by end to begging

int main(){
     char   c, file_name[128];
     ifstream  ifs;

     cout << "\nName of file:  ";  cin >> file_name;
     ifs.open(file_name, ios::in | ios::binary);
     if (!ifs) { cout << “file error!\n";  return 1; }

     ifs.seekg(0, ios::end); // positioning at the end
     ifs.seekg(-1, ios::cur); // positioning at the last octet
     while (ifs.tellg() >= ios::beg) {
          ifs.get(c); cout << c;
          ifs.seekg(-2, ios::cur); /previous octet
     }
     ifs.close();
     return 0;
}
```