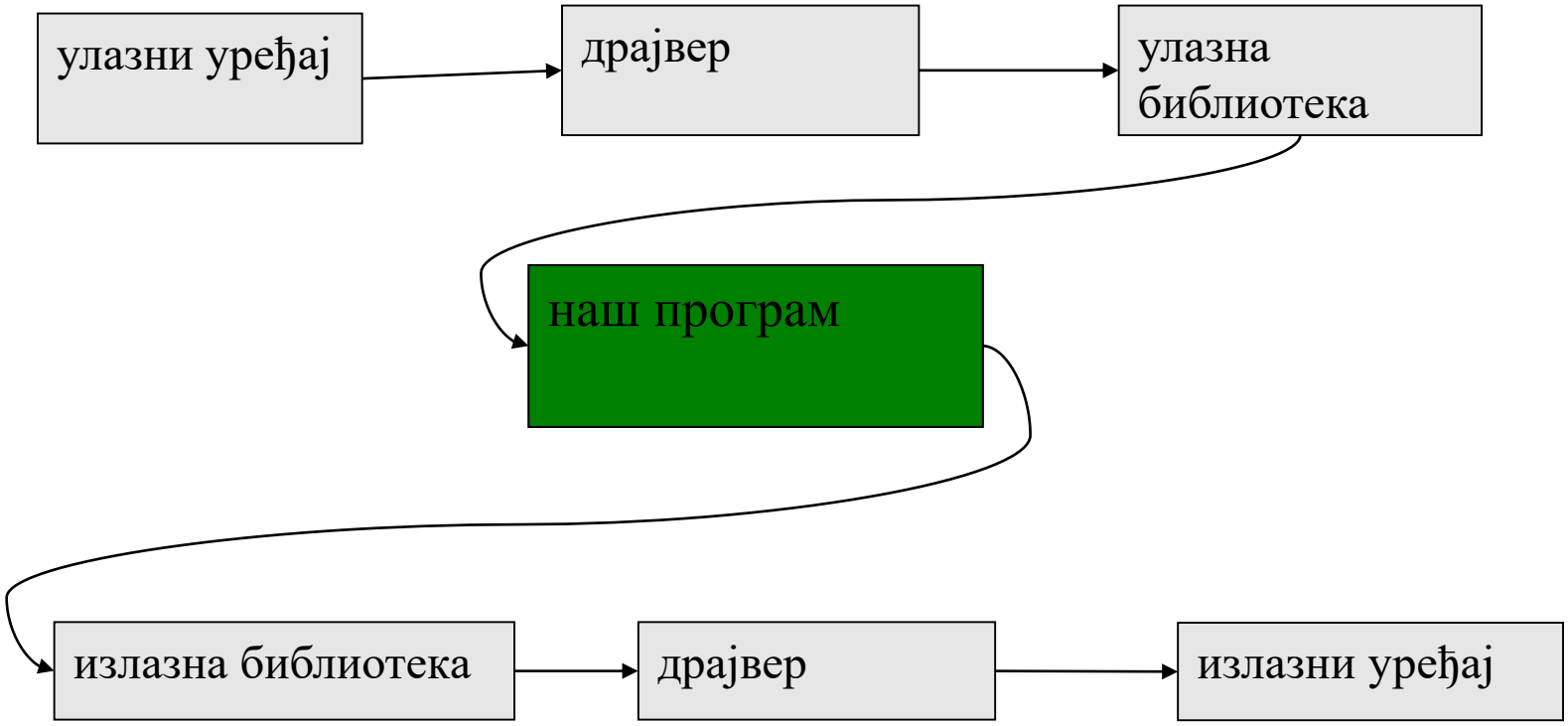
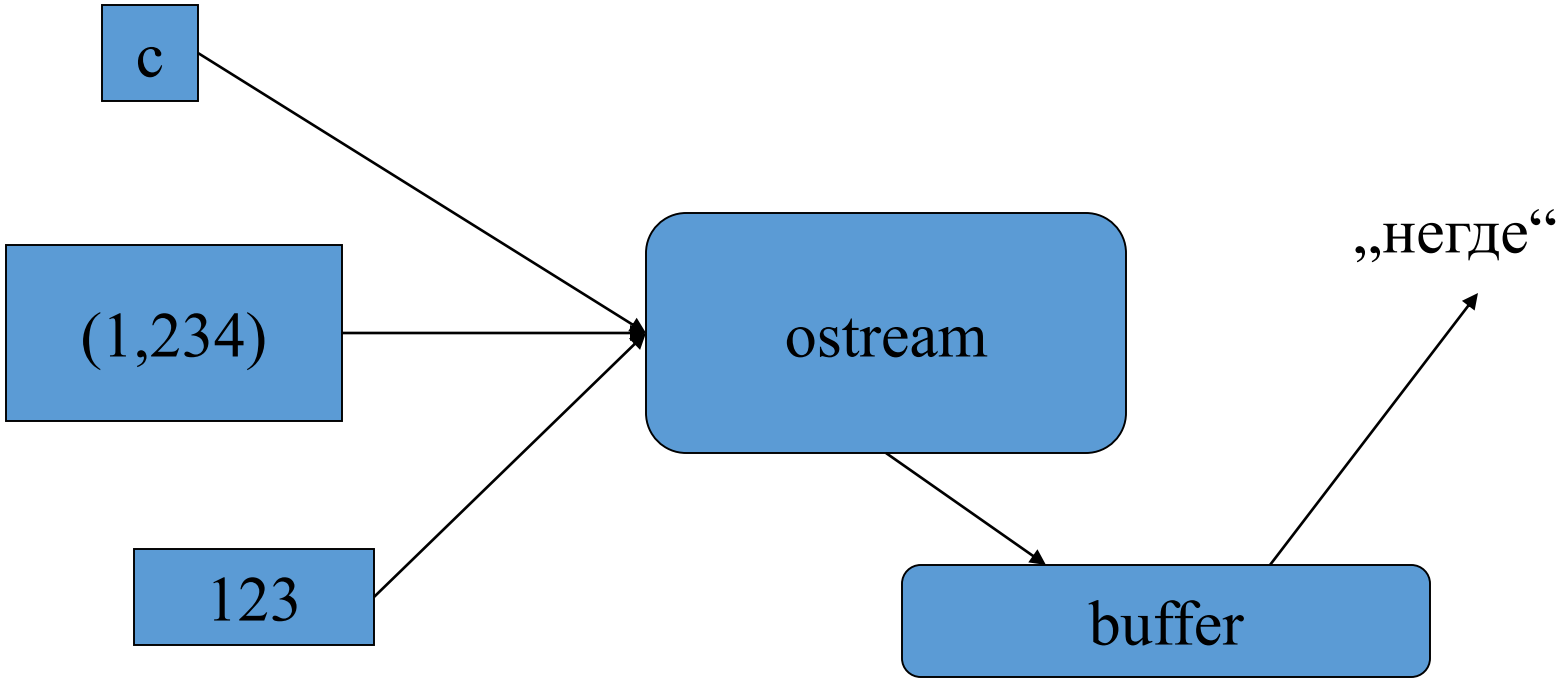
Улазно-излазни токови

Улаз и излаз

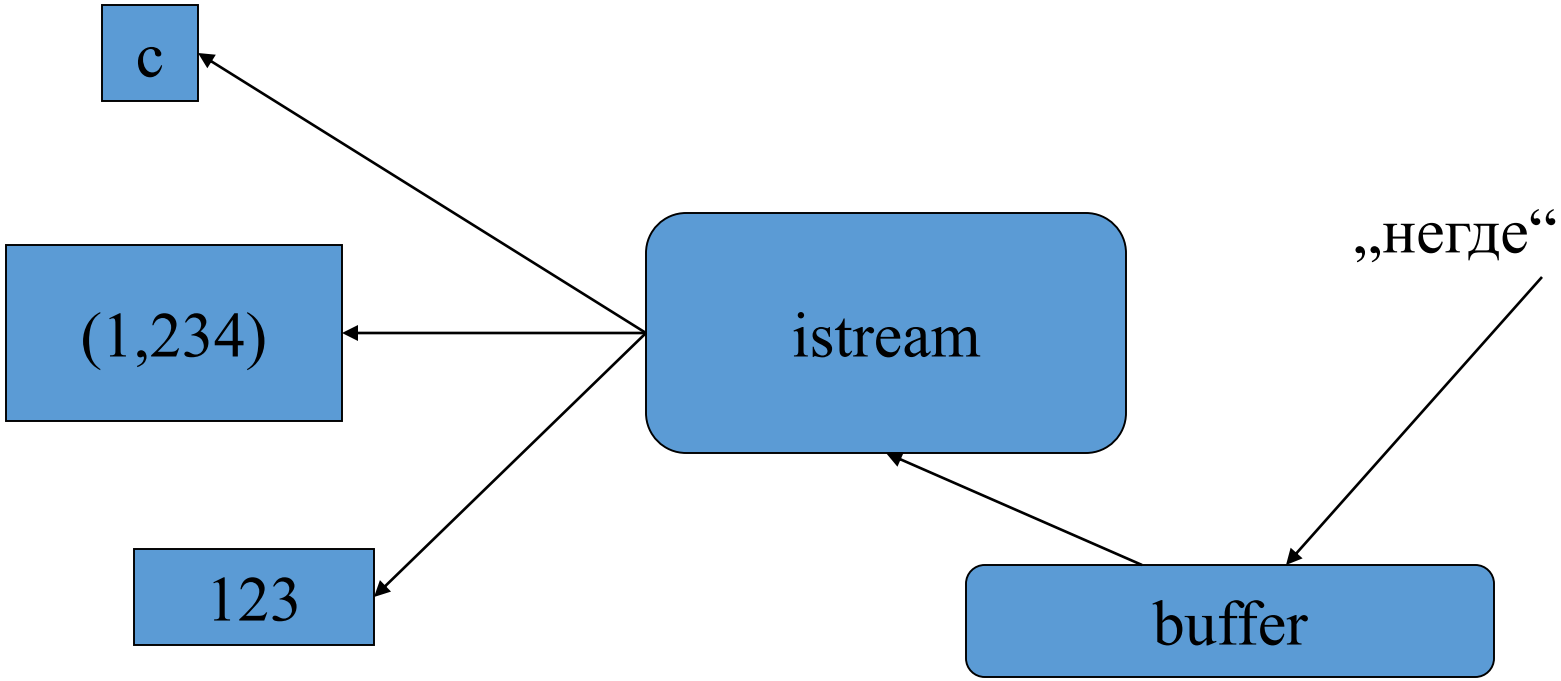

улазни уређај
драјвер
улазна библиотека
наш програм
излазна библиотека
драјвер
излазни уређај

Модел тока



- **ostream (o**utput **stream)**
  - претвара вредности разних типова у поворке знакова
  - шаље те знакове негде
    - конзола, датотека, меморија, други рачунар

Модел тока



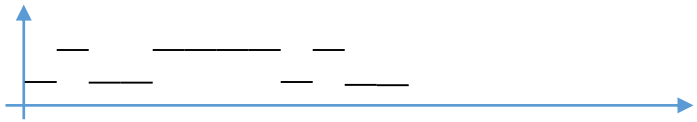
- **istream (i**nput **stream)**
  - претвара поворке знакова у вредности различитих типова
  - добавља знакове од негде
    - конзола, датотека, меморија, други рачунар

Модел тока

- Читање и писање
  - Типизираних објеката
    - << (улаз) и >> (излаз), плус још неке операције
    - Типски безбедно
    - Форматирано
  - Обично су подаци у форми текста (поворке знакова)
    - Али не морају бити (тзв. бинарни токови)
  - Проширuво
    - Могу се додати сопствене операције за сопствене типове (операције << и >> се могу дефинисати као што смо спомињали на претходном часу)
  - Ток може бити наслоњен на било који узлазно-излазни уређај

Нивои апстракције

Напон



Нуле и јединице

```
01001111010011110101000001100100010000
```

Знакови

```
01001111 01001111 01010000 00110010 00100000
‘O’      ‘O’      ‘P’      ‘2’      ‘ ‘
```

Објекти

```
01001111010011110101000001100100010000
“OOP2“
0011001000100000
‘2’+‘ ‘ или 2 (интецер) или “2” (стринг)
```

Датотеке

- За читање датотеке
  - Морамо знати њено име
  - Морамо је отворити (за читање)
  - Онда можемо из ње читати
  - Морамо је затворити
    - Ово се обично имплицитно дешава
- За писање у датотеку
  - Морамо знати њено име
  - Морамо је отворити (за писање)
    - Или направити нову датотеку тог имена
  - Онда можемо у њу писати
  - Морамо је затворити
    - Ово се обично имплицитно дешава

Отварање датотеке за читање

```
// …
int main()
{
  cout << "Please enter input file name: ";
  string name;
  cin >> name;
  ifstream ist(name);

  if (!ist) error("can’t open input file ", name);
  // …
```

Отварање датотеке за писање

```
  // …
  cout << "Please enter name of output file: ";
  cin >> name;
  ofstream ofs(name);

  if (!ofs) error("can’t open output file ", name);
  // …
}
```

Читање из датотеке

- Нека датотека садржи низ уређених парова који представљају редом сат и температуру која је тада измерена

  0 60.7
  1 60.6
  2 60.3
  3 59.22

- Сати су нумерисани од 0 до 23
- Завршетак
  - Долазак до краја датотеке
  - Наилазак на било шта неочекивано у садржају датотеке
    - Нпр. **q**

Читање из датотеке

```
struct Reading { // мерења температуре
    int hour;
    double temperature;
    Reading(int h, double t) : hour(h), temperature(t) { }
};

vector<Reading> temps; // вектор који садржи мерења

int hour;
double temperature;
while (ist >> hour >> temperature) {
    if (hour < 0 || 23 < hour) error("hour out of range");
    temps.push_back(Reading(hour, temperature));
}
```

Руковање грешкама при раду са улазима и излазима

- Извори грешака
  - Људске грешке
  - Датотеке које не одговарају спецификацији
  - Спецификација која не одговара стварности
  - Програмерске грешке
  - итд.

- Код iostream-а резултат операције доводи до четири могућа стања:
  - `good()` // *операција је била успешна*
  - `eof()` // *дошло је до краја датотеке (End Of File)*
  - `fail()` // *десило се нешто неочекивано*
  - `bad()` // *десило се нешто неочекивано и озбиљно*

Пример читања целог броја

- Неочекивани знак у улазу, **или очекивани терминатор**
  - 1 2 3 4 5 *
  - Стање је **fail()**
- Лош формат броја
  - 1 2 3 4 5.6
  - Стање је **fail()**
- Дошло је до краја датотеке
  - 1 2 3 4 5 end of file
  - 1 2 3 4 5 Control-Z (Windows)
  - 1 2 3 4 5 Control-D (Unix)
  - Стање је **eof()**
- Нешто озбиљно лоше се десило
  - Нпр. лоше форматиран диск
  - Стање је **bad()**

Руковање грешкама при раду са улазима и излазима

```
void fill_vector(istream& ist, vector<int>& v, char terminator)
{ // читај интеџере из ist и убацуј у v док не дођемо до eof() или terminator знака
  int i = 0;
  while (ist >> i) v.push_back(i); // учитавај и смештај у v до неке „грешке“
  if (ist.eof()) return; // ако је наишао крај датотеке – то је у реду
  if (ist.bad()) error("ist is bad"); // врло лоше - напуштај брод!

  if (ist.fail()) { // опорави се од грешке или је пријави
     ist.clear(); // очисти стање тока да би смо видели шта је узрок грешке
     char c;
     ist >> c; // сад смо се свели на читање знак по знак
     if (c != terminator) { // terminator је очекивани знак за крај, рецимо *
       ist.unget(); // врати знак назад; може и овако ist.putback(c)
       ist.clear(ios_base::failbit); // постави стање назад на fail()
       // пријави грешку, ако треба
     }
   }
}
```

Бацање изузетка за **bad()** случај

```
// Токови подразумевано не бацају изузетке.
// Овако се саопштава току да баци изузетак у стању bad():
ist.exceptions(ist.exceptions() | ios_base::badbit);

// Може се тумачити као
//   "Постави маску за изузетке тока ist на оно што је
//    до сада била, плус badbit."
// или као
//   "Баци изузетак ако се деси нешто озбиљно лоше."
```

Сада код у претходном примеру не мора да проверава **bad** стање

```
  01101
| 00110
  01111
```

„или“ над битима

Поједностављена улазна петља

```
void fill_vector(istream& ist, vector<int>& v, char terminator)
{ // read integers from ist into v until we reach eof() or terminator
  int i = 0;
  while (ist >> i) v.push_back(i);
  if (ist.eof()) return; // ако је наишао крај датотеке – то је у реду

  // у овој тачки стање није good(), није eof() и није bad() (јер смо ставили да за
  // то баци изузетак и биће ухваћено у неком спољном коду)
  // значи, мора да је fail()
  ist.clear(); // очисти стање тока да би смо видели шта је узрок грешке
  char c;
  ist >> c; // сад смо се свели на читање знак по знак
  if (c != terminator) { // terminator је очекивани знак за крај, рецимо *
    ist.unget(); // врати знак назад; може и овако ist.putback(c)
    ist.clear(ios_base::failbit); // постави стање назад на fail()
    // пријави грешку, ако треба
  }
}
```

Читање једне вредности

```
// Први покушај:

cout << "Please enter an integer in the range 1 to 10 (inclusive):\n";
int n = 0;
while (cin >> n) {
    if (1 <= n && n <= 10) break;
    cout << "Sorry, "
       << n
       << " is not in the [1:10] range; please try again\n";
}
```

- Могуће су три врсте проблема:
  - корисник је унео вредност ван опсега
  - не учитава се никаква вредност (EOF)
  - корисник је унео нешто погрешног типа (у овом случају нешто што није интеџер)

Читање једне вредности

- Шта желимо да урадимо у та три случаја?
  - решити проблем у коду који обавља читање?
  - бацити изузетак, да би неко други решавао проблем (можда и прекинуо програм)?
  - игнорисати проблем?

  - Читање једне вредности је нешто што нам често треба и зато желимо решење које је једноставно за коришћење

Обрадити све случајеве: много кода

```
cout << "Please enter an integer in the range 1 to 10 (inclusive):\n";
int n = 0;
while (n == 0) { // Шта је овде проблем? (прво прочитај цео код)
  cin >> n;
  if (cin) {  // број је добро учитан – провери опсег
       if (1 <= n && n <= 10) break;
       cout << "Sorry, " << n << " is not in the [1:10] range; please try again\n";
  }
  else if (cin.fail()) {   // наишли смо на нешто што није интеџер
       cin.clear();
       cout << "Sorry, that was not a number; please try again\n";
       char ch;
       while (cin >> ch && !isdigit(ch)) ; // прескочи знакове који нису цифре
       if (!cin) error("no input"); // шта могу бити стања у овом тренутку?
       cin.unget(); // врати цифру да би после прочитали број
  }
  else
       error("no input");  // eof или bad – дигни руке
}
// у овој тачки n је у опсегу [1:10]
```

Зашто је код толико компликован? Покушавамо да урадимо све одједном

- Помешали смо следеће ствари
  - учитавање интеџер вредности
  - комуникацију са корисником
  - исписивање порука о грешци
  - прескакање неодговарајућих знакова
  - проверу опсега учитане вредности
- Пробаћемо поделити код на засебне целине

Подела кода

- Које делове желимо?
  - **int get_int(int low, int high); //** учитај **int** у опсегу [**low**..**high**] са **cin**
  - **int get_int();** **//** учитај int са **cin**
  - **void skip_to_int()**; **//** прескочи неодговарајуће знаке

Прескакање неодговарајућих знакова

```
void skip_to_int()
{
  if (cin.fail()) {
    cin.clear();
    char ch;
    while (cin>>ch) {
      if (isdigit(ch)) {
        cin.unget();
        return;
      }
    }
  }
  error("no input");
}
```

Учитавање целог броја

```
int get_int()
{
  int n = 0;
  while (true) {
    if (cin >> n) return n;
    cout << "Sorry, that was not a number; please try again\n";
    skip_to_int();
  }
}
```

Учитај цео број који упада у неки опсег

```
int get_int(int low, int high)
{
  cout << "Please enter an integer in the range "
    << low << " to " << high << " (inclusive):\n";
  while (true) {
    int n = get_int();
    if (low <= n && n <= high) return n;
    cout << "Sorry, "
      << n << " is not in the [" << low << ':' << high
      << "] range; please try again\n";
  }
}
```

Употреба

```
int n = get_int(1, 10);
cout << "n: " << n << endl;

int m = get_int(2, 300);
cout << "m: " << m << endl;
```

- Проблемчић:
  - Комуникација са корисником је и даље уграђена у функције

Једно решење

```
// функције параметризоване опсегом и порукама

int strength = get_int(1, 10,
                "enter strength",
                "Not in range, try again");
cout << "strength: " << strength << endl;

int altitude = get_int(0, 50000,
                "please enter altitude in feet",
                "Not in range, please try again");
cout << "altitude: " << altitude  << "m above sea level\n";
```

- Да ли нам је ово стварно потребно?
- Мора се одговорити за сваки конкретан случај.
- То су важна и честа питања у развоју програма.
- Што више знамо о проблему и његовом решењу, одговори на оваква питања су нам све бољи и бољи.

Једно решење

```
int get_int(int low, int high, const string& greeting, const string& sorry)
{
  cout << greeting << ": [" << low << ':' << high << "]\n";
  while (true) {
    int n = get_int();
    if (low <= n && n <= high) return n;
    cout << sorry  << ": [" << low << ':' << high << "]\n";
  }
}
```

- Непотпуна параметризација: **get_int()** није параметризована
  - помоћне функције не би требало да генеришу своје поруке о грешци
  - заиста, све озбиљне библиотечке функције не генеришу поруку о грешци
    - бацају изузетке (који могу садржати поруку о грешци)

Дефинисање операције << за кориснички тип

- Обично је врло једноставно:

```
ostream& operator<<(ostream& os, const Date& d)
{
  return os << '(' << d.year()
            << ',' << d.month()
            << ',' << d.day() << ')';
}
```

Употреба

```
void do_some_printing(Date d1, Date d2)
{
  cout << d1; // значи: operator<<(cout, d1)

  cout << d1 << d2;
            // (cout << d1)  <<  d2
            // (operator<<(cout, d1)) <<  d2
            // operator<<((operator<<(cout, d1)), d2)
}
```

Дефинисање операције >> за кориснички тип

```
istream& operator>>(istream& is, Date& dd)
  // Учитај датум у формату: ( година , месец , дан )
{
  int y, d, m;
  char ch1, ch2, ch3, ch4;
  is >> ch1 >> y >> ch2 >> m >> ch3 >> d >> ch4;
  if (!is) return is; // нешто се десило, стање је већ постављено у eof() или bad()
  if (ch1!='(' || ch2!=',' || ch3!=',' || ch4!=')') { // грешка у формату
       is.clear(ios_base::failbit); // довођење у стање fail()
       return is;
  }
  dd = Date(y, Month(m), d);
  return is;
}
```