Управљање улазно-излазним токовима

Изглед излаза је важан

- Приметимо да људима није све једно како изгледа излаз неког програма
  - То је често тако са врло добрим разлогом
  - Постоје неке очекиване конвенције
    - Шта значи 110?
    - Шта значи 123,456?
    - Шта значи (123)?

Излазни формати

- Целобројне вредности
  - **1234** (децимално)
  - **2322** (октално)
  - **4d2** (хексадецимално)
- Реални бројеви
  - **1234.57** (општи облик)
  - **1.2345678e+03** (научнички; scientific)
  - **1234.567890** (фиксирани; fixed)
- Прецизност (за бројеве са ограниченом прецизношћу)
  - **1234.57**
  - **1234.6**
- Поља
  - **|12|** (подразумевано за **|** праћено **12** праћено **|**)
  - **| 12|** (**12** у пољу од 4 знака)

Излаз бројева са различитом основом

- Помоћу манипулатора, можете променити основу
  - Основа 10 == децимално; цифре: 0 1 2 3 4 5 6 7 8 9
  - Основа 8 == октално; цифре: 0 1 2 3 4 5 6 7
  - Основа 16 == хексадецимално; цифре: 0 1 2 3 4 5 6 7 8 9 a b c d e f

```
   cout << dec << 1234 << "\t(decimal)\n"
        << hex << 1234 << "\t(hexadecimal)\n"
        << oct << 1234 << "\t(octal)\n";
// Знак '\t' табулатор, или „таб”

// резултат:
   1234 (decimal)
   4d2  (hexadecimal)
   2322 (octal)
```

Лепљиви манипулатори

- Помоћу манипулатора, можете променити основу
  - Основа 10 == децимално; цифре: 0 1 2 3 4 5 6 7 8 9
  - Основа 8 == октално; цифре: 0 1 2 3 4 5 6 7
  - Основа 16 == хексадецимално; цифре: 0 1 2 3 4 5 6 7 8 9 a b c d e f

```
cout << 1234 << '\t'
     << hex << 1234 << '\t'
     << oct << 1234 << '\n';
cout << 1234 << '\n';   // и даље важи oct манипулатор

// резултат:
1234  4d2  2322
2322
```

Остали манипулатори

- Можете променити основу
  - Основа 10 == децимално; цифре: 0 1 2 3 4 5 6 7 8 9
  - Основа 8 == октално; цифре: 0 1 2 3 4 5 6 7
  - Основа 16 == хексадецимално; цифре: 0 1 2 3 4 5 6 7 8 9 a b c d e f

```
cout << 1234 << '\t'
     << hex << 1234 << '\t'
     << oct << 1234 << endl;        // '\n'
cout << showbase << dec;   // прикажи основе кроз префикс
cout << 1234 << '\t'
     << hex << 1234 << '\t'
     << oct << 1234 << '\n';
// резултат:
1234  4d2  2322
1234  0x4d2    02322
```

Манипулатори за бројеве у покретном зарезу

- Може се променити формат исписа:
  - general – **iostream** најбољи формат за **n** цифара (ово је подразумевано)
  - **scientific** – једна цифра пре зареза (тачке) плус експонент; **n** цифара након **.**
  - **fixed** – без експонента; **n** цифара након зареза (тачке)

```
cout << 1234.56789 << "\t\t(general)\n"
    << fixed << 1234.56789 << "\t(fixed)\n"
    << scientific << 1234.56789 << "\t(scientific)\n";

// results:
1234.57         (general)
1234.567890     (fixed)
1.234568e+03    (scientific)
```

Манипулатори прецизности

- Прецизност (подразумевана је 6)
  - general – прецизност је број цифара
    - **general** није стандардни манипулатор, само постоји у std_lib_facilities.h
  - **scientific** – прецизност је број цифара иза зареза (тачке)
  - **fixed** – прецизност је број цифара иза зареза (тачке)

```
cout << 1234.56789 << '\t' << fixed << 1234.56789 << '\t'
     << scientific << 1234.56789 << '\n';
cout << general << setprecision(5)
     << 1234.56789 << '\t' << fixed << 1234.56789 << '\t'
     << scientific << 1234.56789 << '\n';
cout << general << setprecision(8)
     << 1234.56789 << '\t' << fixed << 1234.56789 << '\t'
     << scientific << 1234.56789 << '\n';

// резултат:
1234.57    1234.567890    1.234568e+03
1234.6     1234.56789     1.23457e+03
1234.5679  1234.56789000  1.23456789e+03
```

Ширина излазног поља

- Ширина је број знакова који треба да се употреби у наредној операцији исписа
  - **Опрез: није лепљиво!**
  - Излаз неће бити сасечен да се упакује у поље

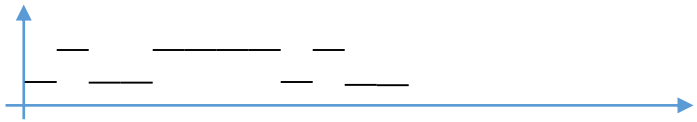
```
cout << 123456 <<'|'<< setw(4) << 123456 << '|'
    << setw(8) << 123456 << '|' << 123456 << "|\n";
cout << 1234.56 <<'|'<< setw(4) << 1234.56 << '|'
    << setw(8) << 1234.56 << '|' << 1234.56 << "|\n";
cout << "asdfgh" <<'|'<< setw(4) << "asdfgh" << '|'
    << setw(8) << "asdfgh" << '|' << "asdfgh" << "|\n";
// резултат:
123456|123456|  123456|123456|
1234.56|1234.56| 1234.56|1234.56|
asdfgh|asdfgh|  asdfgh|asdfgh|
```

Важно је приметити:

- за овакве детаље вам требају књиге, приручници, упутства, референце, Интернет садржај, хелп у развојном окружењу, итд.

Нивои апстракције

Напон



Нуле и јединице

```
01001111010011110101000001100100010000
```

Знакови

```
01001111 01001111 01010000 00110010 00100000
'O'      'O'      'P'      '2'      ' '
```

Објекти

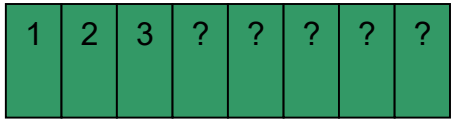
```
01001111010011110101000001100100010000
"OOP2"
0011001000100000
'2'+' ' или 2 или "2"
```

Модови у којима се може отворити датотека

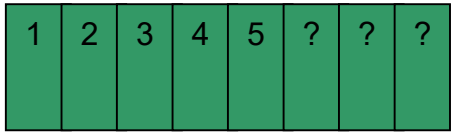
- Подразумевано, **ifstream** отвара за текстуално читање
- Подразумевано, **ofstream** отвара за текстуално писање
- Друге могућности:
  - **ios_base::app**
  - **ios_base::ate**
  - **ios_base::binary**
  - **ios_base::in**
  - **ios_base::out**
  - **ios_base::trunc**
- Примери:
  - **ofstream of1(name1);** // *подразумевано ios_base::out*
  - **ifstream if1(name2);** // *подразумевано ios_base::in*
  - **ofstream ofs(name, ios_base::app);**
  - **fstream fs("myfile", ios_base::in | ios_base::out);**

Текст наспрам бинарних вредности

123 као знакови:

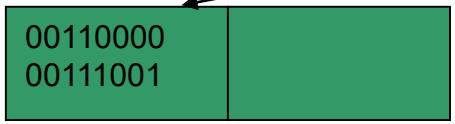


12345 као знакови:



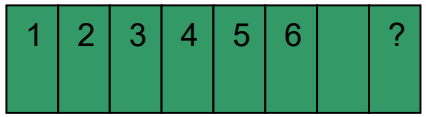
123 бинарно:

12345 бинарно:

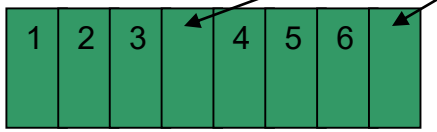


Код бинарних бројева, величина је важна

123456 као знакови:



123 456 као знакови:



Код текстуалних датотека имамо знакове који представљају крај уноса

Текстуално наспрам бинарног

- Кад год можете, користите текстуално
  - Лакше је за читање
  - Лакше је дебаговање
  - Текст је преносивији са платформе на платформу
  - Већина информација се може смислено представити у текстуалној форми
- Бинарно користите само када морате
  - Нпр. слике, звук...

Бинарне датотеке

```
int main()
{
  cout << "Please enter input file name\n";
  string name;
  cin >> name;
  ifstream ifs(name, ios_base::binary);

  if (!ifs) error("can't open input file ", name);

  cout << "Please enter output file name\n";
  cin >> name;
  ofstream ofs(name, ios_base::binary);

  if (!ofs) error("can't open output file ",name);
```

Бинарне датотеке

```
 vector<int> v;

 // читање из бинарне датотеке:
 int i;
 while (ifs.read(as_bytes(i), sizeof(int)))
     v.push_back(i);

 // …

 // писање у бинарну датотеку:
 for (int i = 0; i < v.size(); ++i)
     ofs.write(as_bytes(v[i]), sizeof(int));
 return 0;
}

// За сада третирајте as_bytes() као неку примитиву


// Опрез: на некој другој платформи ово не мора радити исто
```

Позиционирање у току

```
fstream fs(name);

// …
fs.seekg(5);
char ch;
fs >> ch;
cout << "sixth character is " << ch << '(' << int(ch) << ")\n";
fs.seekp(1);
fs << 'y';
```

Позиционирање у току

- Кад год можете
  - Једноставно употребљавајте токове
    - већина кода треба да користи једноставне **istream** и **ostream** објекте
  - Позиционирање је подложније грешкама

Стринг токови

stringstream ради са **string** променљивом.

У суштини, ово је рад са меморијом.

```
double str_to_double(string s)
 {
 istringstream is(s);
 double d;
 is >> d;
 if (!is) error("double format error");
 return d;
}

double d1 = str_to_double("12.4");
double d2 = str_to_double("1.34e-3");
double d3 = str_to_double("twelve point three");

// Мада постоји: std::stod у стандардној библиотеци.
```

Типско наспрам линијског

- Учитавање стринга

```
string name;
cin >> name;       // input: Mali Radojica
cout << name << '\n';  // output: Mali
```

- Учитавање линије

```
string name;
getline(cin, name);    // input: Mali Radojica
cout << name << '\n';  // output: Mali Radojica
// како сада добавити име и презиме?
istringstream ss(name);
ss >> first_name;
ss >> second_name;
```

Функције за класификацију знакова

- Из заглавља **<cctype>**:
  - **`isspace(c)`**
  - **`isalpha(c)`**
  - **`isdigit(c)`**
  - **`isupper(c)`**
  - **`islower(c)`**
  - **`isalnum(c)`**

  итд.

Линијски унос

- Дајте предност операцији **>>** у односу на **getline()**

- Људи обично користе **getline()** зато што не виде алтернативу
  - али то обично закомпликује код
  - када користите **getline()**, обично завршите са кодом који
    - користи **>>** да парсира текст из **stringstream**
    - користи **get()** да учита појединачни знак