به نام خدا

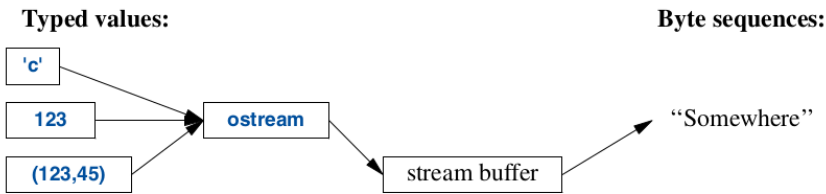
برنامه‌سازی پیشرفته

آرش شفیعی

ورودی و خروجی

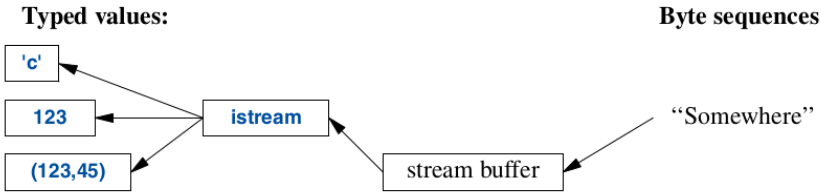
ورودی و خروجی

- کتابخانهٔ جریان ورودی و خروجی [1] امکاناتی برای خواندن یک ورودی و چاپ یک خروجی فراهم می‌کند.
- کلاس `ostream` اشیایی با نوع‌های داده‌ای متفاوت را به جریانی (استریمی) از کاراکترها تبدیل می‌کند.



[1] I/O stream library

- همچنین کلاس `istream` جریانی (استریمی) از کاراکترها را به اشیایی با نوع‌های داده‌ای متفاوت تبدیل می‌کند.



ورودی و خروجی

- شیء cout از کلاس ostream در کتابخانه <ostream> تعریف شده است. عملگر درج خروجی [1] << برای این کلاس سربارگذاری شده است.
- این عملگر یک عملگر دوتایی است که یک شیء از کلاس ostream را در طرف چپ خود و یک داده در طرف راست خود دریافت کرده و آن داده را به خروجی استاندارد برای چاپ می‌فرستد.
- همچنین cerr خروجی استاندارد برای ارسال پیام‌های خطاهاست.

```
۱ i = 42;
۲ cout << "the value of i is " << i << '\n';
```

[1] insert output

ورودی و خروجی

- به طور مشابه شیء `cin` از کلاس `istream` در کتابخانه `<istream>` تعریف شده است. عملگر استخراج ورودی [1] `>>` برای این کلاس سربارگذاری شده است.
- این عملگر یک عملگر دوتایی است که یک شیء از کلاس `istream` را در طرف چپ خود و یک متغیر در طرف راست خود دریافت کرده و مقدار آن متغیر را از ورودی استاندارد می‌خواند.
- حرف `c` در `cin` و `cout` مخفف کلمهٔ کاراکتر است.

```
۱ int i;
۲ double d;
۳ cin >> i >> d; // read into i and d
```

[1] extract input

ورودی و خروجی

- `cin` با استفاده از جداکنندهٔ خط فاصله تشخیص می‌دهد که ورودی‌ها را از هم جدا کرده، در متغیرهای مختلف قرار دهد.
- برای خواندن یک خط از ورودی که با کاراکتر `'\n'` پایان می‌یابد، از تابع `getline` استفاده می‌کنیم.

```
۱ cout << "Please enter your name\n";
۲ string str;
۳ getline(cin,str);
۴ cout << "Hello, " << str << "!\n";
```

ورودی و خروجی

- فرض کنید می‌خواهیم از یک جریان (استریم) ورودی، تعدادی عدد صحیح بخوانیم و به یک وکتور اضافه کنیم. این جریان ورودی می‌تواند یک فایل ورودی یا ورودی استاندارد باشد. بعدها خواهیم دید که علاوه بر `cin` یک فایل ورودی نیز شیئی از یک زیرکلاس `istream` است.

- با استفاده از عملگر استخراج ورودی در یک حلقهٔ تکرار به صورت زیر عمل می‌کنیم.

```
۱ vector<int> read_ints(istream& is) {
۲     vector<int> res;
۳     for (int i; is>>i; )
۴         res.push_back(i);
۵     return res;
۶ }
```

- عملگر استخراج بر روی یک استریم ورودی، یک استریم ورودی باز می‌گرداند که در صورتی که دادهٔ درستی نخواند مقدار پرچم داخلی [1] خود را برابر با `failbit` قرار می‌دهد که معادل صفر یا نادرست است، پس از حلقه خارج می‌شویم.

[1] internal flag

- همچنین می‌توانیم به صورت زیر با استفاده از تابع good() تا وقتی که جریان ورودی با اشکالی روبرو نشده است، ورودی را از cin بخوانیم.

```
cin >> x;
while(cin.good()) {
    cout << "x: " << x << endl;
    cin >> x;
}
```

- سپس برای استفاده مجدد از cin آن را توسط cin.clear() پاکسازی می‌کنیم و مقدار پرچم داخلی را برابر با goodbit قرار می‌دهیم و آنچه در بافر قرار دارد توسط cin.ignore() خالی می‌کنیم.

ورودی و خروجی

- عملگرهای درج و استخراج برای همهٔ نوع‌های داده‌ای اصلی سربارگذاری شده‌اند.
- همانطور که در سربارگذاری توابع گفتیم، دو عملگر استخراج و درج را می‌توانیم برای انواع داده‌ای تعریف‌شده توسط کاربر نیز تعریف کنیم.

```
۱ struct Entry {
۲     string name;
۳     int number;
۴ };
۵
۶ ostream& operator<<(ostream& os, const Entry& e) {
۷     return os << "{\"" << e.name << "\", " << e.number << "}";
۸ }
```

ورودی و خروجی

- برای سربارگذاری عملگر استخراج به صورت زیر عمل می‌کنیم.

```
۱  istream& operator>>(istream& is, Entry& e)
۲  // read { "name" , number } pair. Note: formatted with { " " , and }
۳  {
۴      char c, c2;
۵      if (is>>c && c=='{' && is>>c2 && c2=='"') { // star t with a { "
۶
۷          string name;
۸          // the default value of a string is the empty string: ""
۹          while (is.get(c) && c!='"')
۱۰             // anything before a " is part of the name
۱۱             name+=c;
۱۲
۱۳         // (to be continued)
```

ورودی و خروجی

- برای سربارگذاری عملگر استخراج به صورت زیر عمل می‌کنیم.

```
۱
۲           if (is>>c && c==',') {
۳               int number = 0;
۴               if (is>>number>>c && c=='}') { // read the number and a }
۵                   e = {name,number};
۶                   // assign to the entry
۷                   return is;
۸               }
۹           }
۱۰      }
۱۱      is.setstate(ios_base::failbit);
۱۲      return is;
۱۳      // register the failure in the stream
۱۴  }
```

ورودی و خروجی

- بعد از سربارگذاری این دو عملگر می‌توانیم به صورت زیر مقادیری را در ساختمان Entry بخوانیم و چاپ کنیم.

```
for (Entry ee; cin>>ee; ) // read from cin into ee
    cout << ee << '\n'; // write ee to cout
```

ورودی و خروجی

- کتابخانهٔ iostream تعدادی عملگر برای کنترل کردن فرمت خروجی و ورودی نیز ارائه می‌دهد.
- برای مثال می‌توانیم یک خروجی را در مبناهای مختلف چاپ کنیم.

```
cout << 1234 << ',' << hex << 1234 << ',' << oct << 1234 << '\n';
// print 1234,4d2,2322
```

- همچنین می‌توانیم اعدادی اعشاری به اشکال مختلف نمایش دهیم.

```
constexpr double d = 123.456;
cout << d << "; " // use the default format for d
<< scientific << d << "; " // use 1.123e2 style format for d
<< hexfloat << d << "; "  // use hexadecimal notation for d
<< fixed << d << "; " // use 123.456 style format for d
<< defaultfloat << d << '\n'; // use the default format for d

// output :123.456; 1.234560e+002;
// 0x1.edd2f2p+6; 123.456000; 123.456
```

- اعداد اعشاری را می‌توان توسط تابع precision گرد کرد.

```
۱ cout.precision(8);
۲ cout << 1234.56789 << '\n'; // output : 1234.5679
۳ cout.precision(4);
۴ cout << 1234.56789 << '\n'; // output : 1235
```

ورودی و خروجی

- از کلاس `iostream` که برای ورودی و خروجی استاندارد طراحی شده است، کلاس `fstream` ارث‌بری می‌کند که برای خواندن از فایل‌ها و نوشتن بر روی آنهاست.
- به طور خاص، کلاس `ifstream` برای خواندن از فایل‌ها و کلاس `ofstream` برای نوشتن بر روی فایل‌هاست.
- همچنین در کتابخانهٔ `sstream` کلاس `stringstream` برای خواندن از یک رشته و نوشتن بر روی یک رشته طراحی شده است.
- به طور خاص `istringstream` برای خواندن از روی یک رشته و `ostringstream` برای نوشتن بر روی یک رشته طراحی شده است.

- برای مثال می‌توانیم یک رشته را به صورت زیر با استفاده از ostringstream تولید کنیم.

```
۱ ostringstream oss;
۲ oss << "{temperature," << scientific << 123.4567890 << "}";
۳ cout << oss.str() << '\n';
```

فایل‌ها

- کتابخانهٔ <filesystem> دارای کلاس‌هایی است که می‌توان برای استفاده با فایل‌ها از آنها استفاده کرد.
- برای مثال می‌توانیم با استفاده از کلاس path آدرس یک فایل را دریافت کنیم.

```
۱  int main(int argc, char* argv[]) {
۲      if (argc < 2) {
۳          cerr << "arguments expected\n";
۴          return 1;
۵      }
۶
۷      path p {argv[1]}; // create a path from the command line
۸      cout << p << " " << exists(p) << '\n';
۹      // note: a path can be printed like a string
۱۰     // ...
۱۱ }
```

فایل‌ها

- با استفاده از کلاس ofstream می‌توانیم یک فایل را برای نوشتن و با استفاده از ifstream می‌توانیم یک فایل را برای خواندن باز کنیم.

```
۱ ofstream file;
۲ file.open ("example.txt");
۳ if (file.is_open()) {
۴     file << "Writing this to a file.\n";
۵     file.close();
۶ }
```

فایل‌ها

- با استفاده از کلاس ofstream می‌توانیم یک فایل را برای نوشتن و با استفاده از ifstream می‌توانیم یک فایل را برای خواندن باز کنیم.

```
1  string line;
2  ifstream file ("example.txt");
3  if (file.is_open()) {
4      while ( getline (file,line) ) {
5        cout << line << '\n';
6      }
7      file.close();
8  }
```

- بعد از خواندن ورودی یا نوشتن خروجی می‌توانیم بررسی کنیم که آیا خواندن و نوشتن موفقیت آمیز بوده یا با خطا برخورد کرده‌ایم و یا به انتهای فایل رسیده‌ایم. بدین منظور توابعی مانند good، bad، fail، eof در نظر گرفته شده است.

فایل‌ها

- با استفاده از تابع seekg می‌توانیم مکانی در فایل که خواندن از آنجا صورت می‌گیرد را تعیین کنیم. همچنین با استفاده از تابع seekp می‌توانیم مکانی در فایل که نوشتن در آنجا صورت می‌گیرد را تعیین کنیم.
- با استفاده از تابع tellg و tellp نیز مکانی که خواندن و نوشتن در آنجا صورت می‌گیرد را دریافت کنیم.

```
۱ seekg ( position );
۲ seekp ( position );
۳ seekg ( offset, direction );
۴ seekp ( offset, direction );
۵ tellg();
۶ tellp();
```

فایل‌ها

- برای مثال با استفاده از این توابع می‌توانیم اندازهٔ یک فایل را تعیین کنیم.

```
streampos begin,end;
ifstream myfile ("example.bin", ios::binary);
begin = myfile.tellg();
myfile.seekg (0, ios::end);
end = myfile.tellg();
myfile.close();
cout << "size is: " << (end-begin) << " bytes.\n";
```

فایل‌ها

- برای فایل‌های دودویی که متنی نیستند، می‌توانیم از توابع `read(memory_block, size)` و `write(memory_block, size)` برای خواندن و نوشتن استفاده کنیم.

```
streampos size;
char * memblock;
ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
if (file.is_open()) {
    size = file.tellg();
    memblock = new char [size];
    file.seekg (0, ios::beg);
    file.read (memblock, size);
    file.close();
    cout << "the entire file content is in memory";
    delete[] memblock;
} else
    cout << "Unable to open file";
```

فایل‌ها

- کلاس path توابع کاربردی زیادی دارد که می‌توان از آنها استفاده کرد.

```
۱  void test(path p) {
۲      if (is_directory(p)) {
۳          cout << p << ":\n";
۴          for (const directory_entry& x : directory_iterator(p)) {
۵              const path& f = x;
۶              string n = f.extension().string();
۷              if (n == ".cpp" || n == ".C" || n == ".cxx")
۸                  cout << f.stem() << " is a C++ source file\n";
۹          }
۱۰     }
۱۱ }
```

- توابع دیگری مانند copy برای کپی کردن فایل‌ها، copy_file برای کپی‌کردن محتوای فایل، create_directory برای ساختن پوشه، و remove برای حذف فایل وجود دارند که در صورت نیاز می‌توان از آنها استفاده کرد.