第12章「値のように振舞うクラス」

主なトピック

- 型変換のコンストラクタ
- 演算子の定義
- friend関数

Strクラスのコンストラクタ

```
class Str  {
public:
        typedef  Vec<char>::size_type  size_type;
        //  4種類のコンストラクタ
        Str()  {}
        Str(size_type n, char c) : data(n, c)  {}
        Str(const char* cp)  {
             std::copy(cp, cp + std::strlen(cp), std::back_inserter(data));
        }
        template <class In> Str(In b, In e)          {
             std::copy(b, e, std::back_inserter(data) );
        }
private:
    Vec<char>  data;
}
```

自動の型変換

- Strクラスは，以下のような使い方ができると便利
  - Str s(“hello”); // オブジェクトの生成
  - Str t = “Hello”; // 初期化
  - s = “Hello!”; // 代入
- “hello”はconst char*型なので，型変換
  - これは，3番目のコンストラクタで対応済み

自動の型変換

- 1番目の「=」は，初期化なのでconst Str&を引数に取るコピーコンストラクタが使われる
  - そのようなコンストラクタは定義されていない！
- 2番目の「=」は，代入
  - const char*を引数にとる代入演算子は定義されていない！
- でも，これで大丈夫
  - ユーザ定義の型変換

ユーザ定義の型変換

- Str(const char*)というコンストラクタが定義されている
- Strが必要な場所でconst char*が使われると，自動的にこのコンストラクタが呼び出され，Str型への型変換が行われる

Strの演算子

```
class Str  {
public:
    //  以下を追加
    char& operator[ ]  (size_type i)  { return data[ i ]; }
    const char& operator[ ]  (size_type i) const { return data[ i ]; }
}
```

入出力演算子

- Str s; cin >> s; のように使いたい
- しかし，クラスの演算子として定義しようとすると，cin.operator >> (s) のようにcinクラスの演算子になってしまう
- そこで，関数として実装

std::istream& operator >> (std::istream&, Str&);

std::ostream& operator << (std::ostream&, const Str&);

入出力演算子

```
std::ostream&  operator << (ostream& os, const Str& s)
{
    for(Str::size_type i = 0; i != s.size(); ++i)
            os << s[ i ];
    return os;
}
```

Strクラス

```
class Str  {
public:
    //  以下を追加
    size_type  size()  const  { return data.size(); }
}
```

入出力演算子

```
std::istream&  operator >> (istream& is, Str& s)
{
    //  現在のデータを破棄, Vecクラスにclear()を実装する必要
    s.data.clear();
    //  空白を読んで破棄
    char c;
    while( is.get(c) && isspace(c) )
            //  空白だったら何もしない
    if(is)  {
            //  このままでは，ここでコンパイルエラー
            // この演算子にprivateのメンバにアクセスさせる必要
            do s.data.push_back(c);
            while( is.get(c) && !isspace(c) );
            //  もし，空白を読み込んだら，ストリームに戻す
            if(is)
                is.unget();
    }
    return is;
}
```

フレンド

```
class Str  {
    //  以下を追加
    friend  std::istream& operator >> (std::istream&, Str&);
public:
}
```

- friendは，この演算子（関数）がStrクラスのprivateメンバにアクセスすることを許可する

他の2項演算子

- s = s + s1; や s += s1; のような演算子があると便利

```
class Str  {
public:
    //  以下を追加
    Str& operator += (const Str& s)   {
            std::copy( s.data.begin(), s.data.end(), std::back_inserter(data) );
            return *this;
    }
    Str& operator + (const Str& s, const Str& t)            {
            Str r = s;
            r += t;
            return r;
    }
}
```

```
char* begin(){
    return( data.begin() );
}
char* end(){
    return( data.end() );
}
bool operator == (const Str& s) const {
    if( size() != s.size() )
        return false;
    for( Str::size_type i = 0; i != size(); ++i ){
        if(data[i] != s[i])
            return false;
    }
    return( true );
}
```