Лекция 8. Строки. Ввод/вывод.

Кодировка (набор символов)

- Определяет соответствие отображения символа некоторой последовательности байт
- Популярны: однобайтовые кодовые страницы (пример?) и юникод (сколько байт в utf-8 на букву?)
- Язык программирования не решает вопрос интерпретации символов
- Можно автоматически детектировать кодировку (неточно).
- Существуют также специальные последовательности для начала файла (BOM).

Строка std::basic_string

```
1.  template<
2.      class CharT,
3.      class Traits = std::char_traits<CharT>,
4.      class Allocator = std::allocator<CharT>>
5.  class basic_string;
```

- Оперирует последовательностью char-подобных символов.
- Хранит непрерывный буфер, с нулевым символом в конце (size его не учитывает).
- Есть typedefs: string, wstring, u16string, u32string.
- В пару полезен basic_string_view

Базовые операции над строками

- Многие операции схожи с операциями над вектором.
- Конструкторы: может неявно конструироваться от C-строки – экономия на операторах.
- Функция получения строки: c_str()
- Содержит функции поиска: find, find_first_of, find_last_not_of, …
- Содержит константу npos – обычно, индикатор конца строки.
- Может выводиться/читаться в/из потока.

boost string algorithms

- Дополняет STL библиотеку полезными функциями (<boost/algorithm/string.hpp>)
- Содержат: triming, преобразование регистра, предикаты, find/replace, split, классификаторы (e.g. isspace).

```
1.  vector<string> strings;
2.  split(strings, "one; two, , three",
3.  is_any_of(",;"), token_compress_on);
```

Text vs Binary files format

- Текстовый формат хранит обычный печатный текст. Специальные символы могут иметь особую интерпретацию (0-31): конец строки, конец файла, звуковой сигнал и т.д.
- Конец строки в text зависит от OS: \r, \r\n, \n
- В текстовом файле важна установленная локаль
- Работа с бинарным файлом использует все байты в нем, как они есть.

printf, fprintf, snprintf

- Формат описывает строку вывода со вставками значений переменных. Каждая вставка начинается с %. Можно описать выравнивание, мин. ширину вывода, дополнение до мин. ширины, точность (для floating point), наличие обязательного знака у чисел и т.д.
- Отдельно указывается тип выводимого значения. Надо быть осторожным в соответствии типа в строке формата и типа переменной. GCC поможет - выдаст предупреждение.

```
1.  int printf( const char* format, ... );
2.  int fprintf( std::FILE* stream, const char* format, ... );
3.  int sprintf( char* buffer, const char* format, ... );
4.  int snprintf( char* buffer, int buf_size, const char* format, ... );
5.
6.
7.  double res = 3.1415;
8.  fprintf(some_file, "result is %.2lf", res);
9.  // result is 3.14
10.
11. int value = 42;
12. printf("answer is %04d or %04x\n", value, value);
13. // answer is 0042 or 002a
```

scanf, fscanf, sscanf, scanf_s

- Считывает из строки пробельные символы, точно соответствующие формату символы и переменные, обозначенные в формате символом %.
- Случайно можно получить code injection (как?)

```
1.  int scanf( const char* format, ... );
2.  int fscanf( std::FILE* stream, const char* format, ... );
3.  int sscanf( const char* buffer, const char* format, ... );
4.
5.  //example:
6.  int   i, j; float x, y; char  str1[10];
7.
8.  char input[] = u8"25 54.32E-1 Thompson 56789 12";
9.
10. int ret = std::sscanf(
11. input, "%d%f%9s%2d%f%*d", &i, &x, str1, &j, &y);
12.
13. //result:
14. i    = 25
15. x    = 5.432
16. str1 = Thompson
17. j    = 56
18. y    = 789
```

Потоки ввода/вывода

- Универсальное средство ввода/вывода, обеспечивают:
  - Ввод/вывод
  - Форматирование
  - Буферизацию
  - Национальные особенности
  - Стандартный ввод/вывод (cin, cout, cerr)
  - Файловый [i|o]fstream
  - Строковый [i|o]stringstream
  - RAII
  - Threadsafe (*)

Вывод

- Выводиться могут по умолчанию встроенные типы. Для любого пользовательского типа также можно поддержать вывод (и ввод).

```
1.  template<class Ch, class Tr = std::char_traits<Ch>>
2.  class basic_ostream : virtual public std::basic_ios<Ch, Tr>{};
3.
4.  cout << 2 << "sqrt is " << 1.41;
5.  cout << (a ^ b);
6.  cout << int('A') << " " << char(65); // 65 A
7.
8.  // output by interface
9.  struct base_t
10. {
11. virtual ostream& out(ostream&) const;
12. /*...*/
13. };
14.
15. ostream& operator<<(ostream& os, base_t const& obj)
16. {return obj.out(os); }
```

Ввод

- Пропускает все разделители.
- Читает, пока может.

```
1.  template<class Ch, class Tr = std::char_traits<Ch>>
2.  class basic_istream : virtual public std::basic_ios<Ch, Tr>{};
3.  //..
4.
5.  vector<int> v(size);
6.  while (i != v.size() && cin >> v[i])
7.      ++i;
8.
9.  // reading of 10 9 8. 7 6 ends on 8
```

Состояния потока

```
1.  good       // checks if no error has occurred i.e.
2.             // I/O operations are available
3.  eof        // checks if end-of-file has been reached
4.  fail       // checks if a recoverable error has occurred
5.  bad        // checks if a non-recoverable error has occurred
6.  operator!      // checks if an error has occurred (syn of fail()
7.  operator bool // checks if no error has occurred (syn !fail())
8.  rdstate    // returns state flags
9.  setstate   // sets state flags
10. clear      // clears error and eof flags
```

Форматированный ввод/вывод

- Оперирует флагами, определенными внутри ios_base: right..left, boolalpha, oct..hex, showpos, и т.д.

```
1.  fmtflags flags() const;
2.  fmtflags flags(fmtflags flags);
3.
4.  fmtflags setf(fmtflags flags);
5.  fmtflags setf(fmtflags flags, fmtflags mask);
6.
7.  cout << 42; // 42
8.  cout.setf(ios_base::oct, ios_base::basefield);
9.  cout.setf(ios_base::showbase);
10. cout << 42; // 052
```

Манипуляторы

- Позволяют избежать low-level интерфейса управления флагами. Все благодаря небольшому ухищрению:

```
1.  basic_ostream& operator<<
2.      (basic_ostream& (*f)(basic_ostream&))
3.  {
4.      return f(*this);
5.  }
6.
7.  cout << 42 << endl << oct << showbase << endl << 42;// 42 \n 052
8.  cout << setprecision(2) << 3.1415;                  // 3.14
```

Файловые потоки

- RAII потоки ввода/вывода в файл.

```
1.  ifstream ifs("numbers"); // app, ate, in, out,
2.  ofstream ofs("squares"); // trunc, binary flags
3.
4.  double number;
5.  while (ifs >> number)
6.      ofs << sqr(number);
7.
8.  //..
9.  ifstream ifs("input" , ios_base::binary);
10. Ofstream ofs("output", ios_base::binary);
11.
12. #pragma pack(push, 1)
13. struct data{/*...*/};
14. data d = {/*...*/};
15. #pragma pack(pop)
16.
17. ifs.read (&d, sizeof(d));
18. ofs.write(&d, sizeof(d));
```

boost lexical_cast

- Упрощают перевод между строковыми и числовыми типами .
- Неуспешный перевод приводит к генерации исключения boost::bad_lexical_cast

```
1.  string str_num = "3.1415";
2.  double value = std::strtof(str_num, 0); // C++11
3.  str_num = to_string (value);            // C++11
4.
5.  value = boost::lexical_cast<double>(str_num);
```

boost format

- Похожа на sprintf, но обеспечивает:
  - строгую проверку типов
  - гарантию непереполнения
  - можно не указывать тип в формате – будет выведен автоматически
- Теряет в скорости. Но это можно исправить. Как?

```
    string person = "Baggins";
1.  size_t coins  = 7;
2.  string phrase =
3.    str(boost::format("Mr. %s needs %2d coins") % person % coins);
4.
    cout << format ("Mr. %1% needs %2$2d coins") % person % coins;
```

memory mapped files*

- Наиболее быстрый способ чтения/записи из/в файл. И один из наиболее удобных, благодаря boost.

```
1.  using boost::interprocess;
2.
3.  file_mapping  file  ("/usr/home/file", read_write);
4.  mapped_region region(file, read_write, size/2, size - size/2);
5.
7.  char* where = region.get_address();
8.  size_t size = region.size();
9.
10. //..
11. region.flush(offset, size);
```

Вопросы?