Контрольные вопросы. Задание 17.

12 марта 2023 г.

1 Почему класс std::string имеет много перегруженных функций-членов?

Класс std::string имеет много перегруженных функций-членов, т.к. является специализацией шаблонного класса std::basic_string < charT, traits, Allocator=std::Allocator < T > >, а именно:

```
using std::string = basic_string<
        char,
        std::char_traits < char >,
        std::Allocator< char > >
```

std::string обеспечивает поддержку таких объектов с интерфейсом, аналогичным интерфейсу стандартного контейнера байтов, но с добавлением функций, специально предназначенных для работы со строками однобайтовых символов.

Сам же класс std::basic_string является т.н. классом Бога, поскольку охватывает довольно много принципов и способов работы с данными. Его очень трудно изменять, поэтому и приходится использовать специализации.

Типичный пример неудобства работы с общим интерфейсом std::string – это работа с индексами.

2 Как осуществляется интернационализация и локализация программ?

Интернационализация – это адаптация продукта для потенциального использования практически в любом месте. Локализация – это добавление специальных функций для использования в некотором определённом регионе. Интернационализация и локализация учитывают особенности национального и мирового сообществ.

Они осуществляются с помощью локалей – объектов локального контекста. Это наборы правил, регламентирующие принципы работы со строками в данной системе, например, как выводить время, дату, числа, валюту и т.д. Все эти компоненты называются фасетами, т.е. фасеты – это объекты для учёта отдельных особенностей, и вместе они образуют локаль.

Для выполнения правил, определяемых локалью, она передаётся в какой-либо поток. Потоки же корректируют ввод и вывод в соответствии с локалью. Стандартный вид локали: язык_зона.код [@ модификатор]. В языке C устанавливается одна глобальная локаль, например, setlocale(). Этот же метод установки локали для потока применим и в C++, однако в духе ООП здесь предложены классы std::locale(), позволяющие работать с локалями точечно, т.е. использовать различные для различных потоков.

Также можно отметить, что язык C по умолчанию поддерживает только локали C и POSIX и никаких региональных локалей. OS Linux по умолчанию имеет кодировку UTF-8 и, не нуждаясь в setconsole(), может использовать региональные локали.

3 Чем отличаются многобайтовые кодировки от широких кодировок?

Многобайтовые кодировки (multibyte):

- кодируют символы переменным числом байт;
- являются более компактными, а потому лучше подходят для хранения данных.

Широкие символы (widechar):

- кодируют символы постоянным числом байт;
- являются более удобными (например, легко можно вычислить количество элементов в последовательности символов по известному размеру), а потому лучше подходят для использования в программах.

4 Какие компоненты входят в стандарт кодирования символов Unicode?

В стандарт кодирования символов Unicode входят:

- *Универсальный набор символов (UCS)* – коды. UCS представляет собой таблицу с номерами строк от 0x0 до 0x10FFFF (шестнадцатиричная система счисления). Это размера с запасом хватает на размещение всех изветсных символов (более миллиона строк). Первые 127 символов соответсвуют символам таблички ASCII с теми же кодами (чем обеспечивается обратная совместимость с ASCII). Затем расположены символы в порядке от наиболее популярных к менее популярным. Кроме того, в таблице есть специальные служебные участки (например, участок D800 – DFFF может обеспечивать функционирование кодировки UTF-16). Наконец, одной из важных отметок является код FFFF, соответсвующий 65536 символам.
- *Семейство кодировок (UTF).* Существует довольно много различных кодировок, которые по-разному работают с UCS. Например, UTF-32 является широкой, кодирует символ 4 байтами, соответственно, каждый символ имеет свой код (так можно закодировать 2^{32}, чем с избытком хватает). UTF-8 и UTF-16 являются многобайтовыми и их принципы работы с UCS несколько сложнее (есть дополнительная логика работы с UCS).

Запись самих символов выглядит как U + code, где code – номер символа из UCS.

5 Для решения каких задач удобно использовать регулярные выражения?

Регулярные выражения удобно использовать в качестве грамматики для описания паттернов при поиске в некотором тексте фрагментов, удовлетворяющих данному шаблону-паттерну.

Литература.

[1] Конспект семинара. Макаров И.С.

[2] ...