Агеева 110Б-21

**Теория VIII курсового проекта.**

**I. Строковый тип. Библиотека <string.h>**

Строковый тип (в программировании) — тип данных, значениями которого является произвольная последовательность символов алфавита. Каждая переменная такого типа может быть представлен фиксированным количеством байтов или иметь произвольную длину.

**Строки в C**, как и в большинстве языков программирования высокого уровня рассматриваются как отдельный тип, входящий в систему базовых типов языка. Так как язык C по своему происхождению является языком системного программирования, то **строковый тип данных в C как таковой *отсутствует*, а в качестве строк в С используются обычные массивы символов.** Завершающий ноль является признаком конца строки.

**Представление в памяти**  
Некоторые языки программирования накладывают ограничения на максимальную длину строки, но в большинстве языков подобные ограничения отсутствуют.  
Основные проблемы в машинном представлении строкового типа:  
строки могут иметь достаточно существенный размер (до нескольких десятков мегабайтов);  
изменяющийся со временем размер — возникают трудности с добавлением и удалением символов.  
В представлении строк в памяти компьютера существует два принципиально разных подхода.  
**Представление массивом символов**В этом подходе строки представляются массивом символов; при этом размер массива хранится в отдельной (служебной) области. От названия языка Pascal, где этот метод был впервые реализован, данный метод получил название Pascal Strings.  
**Преимущества**  
программа в каждый момент времени «знает» о размере строки, и операции добавления символов в конец, копирования и получения размера строки выполняются достаточно быстро;  
строка может содержать любые данные;  
возможно на программном уровне следить за выходом за границы строки при её обработке;  
возможно быстрое выполнение операции вида «взятие N-ого символа с конца строки».  
**Недостатки**  
проблемы с хранением и обработкой символов произвольной длины;  
увеличение затрат на хранение строк — значение «длина строки» также занимает место и в случае большого количества строк маленького размера может существенно увеличить требования алгоритма к оперативной памяти;  
ограничение максимального размера строки. В современных языках программирования это ограничение скорее теоретическое, так как обычно размер строки хранится в 32-битовом поле, что даёт максимальный размер строки в 2 147 483 647 байт (2 гигабайта).  
**Метод «завершающего байта»**Второй метод заключается в использовании «завершающего байта». Одно из возможных значений символов алфавита (как правило, это символ с кодом 0) выбирается в качестве признака конца строки, и строка хранится как последовательность байтов от начала до конца. Есть системы, в которых в качестве признака конца строки используется не символ 0, а байт 0xFF (255) или код символа «$».  
Метод имеет три названия — ASCIIZ (символы в кодировке ASCII с нулевым завершающим байтом), C-strings (наибольшее распространение метод получил именно в языке Си) и метод нуль-терминированных строк.  
**Преимущества**  
отсутствие дополнительной служебной информации о строке (кроме завершающего байта);  
возможность представления строки без создания отдельного типа данных;  
отсутствие ограничения на максимальный размер строки;  
экономное использование памяти;  
простота получения суффикса строки;  
возможность использовать алфавит с произвольным размером символа (например, UTF-8).

**Объявление строк в C**

Строки реализуются посредством массивов символов. Поэтому объявление строки имеет следующий синтаксис:

char имя[длина];

Объявление **строки в С** имеет тот же синтаксис, что и объявление одномерного символьного массива. Длина строки должна представлять собой целочисленное значение (в стандарте C89 – константа, в стандарте C99 может быть выражением). Длина строки указывается с учетом одного символа на хранение завершающего нуля, поэтому максимальное количество значащих символов в строке на единицу меньше ее длины. Например, строка может содержать максимально двадцать символов, если объявлена следующим образом:  
char str[21]; Инициализация строки в С осуществляется при ее объявлении, используя следующий синтаксис:  
char str[длина] = строковый литерал;

*Пример:* const char message[] = "Gogogo!"

Так как строки на языке С являются массивами символов, то к любому символу строки можно обратиться по его индексу. Для этого используется синтаксис обращения к элементу массива, поэтому первый символ в строке имеет индекс ноль.

printf(“%c ”, str[i]);

**Операции со строками**

Большинство операций языка Си, имеющих дело со строками, работает с указателями. Для размещения в оперативной памяти строки символов необходимо:

1. выделить блок оперативной памяти под массив;

2. проинициализировать строку.

Для выделения памяти под хранение строки могут использоваться функции динамического выделения памяти. При этом необходимо учитывать требуемый размер строки:

char \*name;

name = (char\*)malloc(10);

scanf("%9s", name);

Для ввода строки использована функция scanf(), причем введенная строка не может превышать 9 символов. Последний символ будет содержать '\0'.

## Функции для работы со строками в С

Все библиотечные функции, предназначенные для работы со строками, можно разделить на три группы:

1. ввод и вывод строк;
2. преобразование строк;
3. обработка строк.

*Функции вывода строк*

Для вывода строк можно воспользоваться рассмотренной ранее функцией

printf("%s", str); // str — указатель на строку

*Функция ввода символов*

Для ввода символов может использоваться функция

char getchar();

которая возвращает значение символа, введенного с клавиатуры. Указанная функция использовалась в рассмотренных ранее примерах для задержки окна консоли после выполнения программы до нажатия клавиши.

**string.h** — [заголовочный файл](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%84%D0%B0%D0%B9%D0%BB) [стандартной библиотеки языка Си](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0_%D0%A1%D0%B8), содержащий функции для работы со [строками, оканчивающимися на 0](https://ru.wikipedia.org/wiki/%D0%9D%D1%83%D0%BB%D1%8C-%D1%82%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B0), и различными функциями работы с памятью.

В основном функции библиотеки не работают с выделением памяти, поэтому эта обязанность возлагается на программиста.

**Основные функции стандартной библиотеки string.h**

char \***strcat**(char \*s1, char \*s2) - присоединяет s2 к s1, возвращает s1

char \***strсpy**(char \*s1, char \*s2) - копирует строку s2 в строку s1, включая '\0', возвращает s1

int **strcmp**(char \*s1, char \*s2) - сравнивает s1 и s2, возвращает значение 0, если строки эквивалентны

int **strlen**(char \*s) - возвращает количество символов в строке s

char **\*strset**(char \*s, char c) - заполняет строку s символами, код которых равен значению c, возвращает указатель на строку s

char **\*strnset**(char \*s, char c, int n) - заменяет первые n символов строки s символами, код которых равен c, возвращает указатель на строку s

int **strncmp**(const char \*, const char \*, size\_t) - лексикографическое сравнение первых n байтов строк

**II. Библиотечные средства обработки списков в С++**

## список ****list****

Это структура данных, которая построена на двусвязных списках. Это значит, что любой элемент знает только о предыдущем и о следующем элементах.

**Библиотечные средства обработки списков в С++**  
Контейнерные классы в С++ – это классы предназначенные для хранения данных, организованных определенным образом. К этим классам, в частности, относятся массивы, стеки и списки. Для каждого типа контейнера определены методы для работы с его элементами, не зависящие от конкретного типа данных, поэтому один и тот же тип контейнера можно использовать для хранения данных различных типов. Эта возможность реализована с помощью стандартной библиотекой шаблонов (STL – Standart Template Library), части С++, куда входят контейнерные классы, алгоритмы и итераторы.  
Использование контейнеров позволяет значительно повысить надежность программ, их переносимость и универсальность, а также уменьшить сроки их разработки.  
Естественно, эти преимущества не даются даром: универсальность и безопасность использования контейнерных классов не могут не отражаться на быстродействии программы. Снижение быстродействия в зависимости от реализации компилятора может быть весьма значительным. Кроме того, для эффективного использования контейнеров требуется затратить значитильные усилия на вдумчивое освоение библиотеки.  
Преимущества списков в целом:  
1. Эффективная вставка и удаление элементов по всему списку (фиксированное время).  
2. Эффективное перемещение элементов внутри списка или между ними (фиксированное время).  
3. Перемещение между элементами назад-вперед (время изменяется линейно).  
По сравнению с другими типами контейнеров списки лучше приспособлены к вставке, удалению и перемещению элементов. То есть, в частности, к сортировке.  
Однако, ссылки назад-вперед заполняют память, а отсутствие индексов замедляет доступ к данным.  
**Реализация.**  
Класс list в STL реализован в виде простого двунаправленного списка, с указателями на предыдущий и следующий элементы. Класс же следит за хранением списка в памяти, сжимая или расширяя его в соответствии с выполняемым действием. Для задания списка на С++ нужно два параметра.  
Объявление шаблона:  
template < class T, class Allocator = allocator<T> > class list;

Где Т – тип элемента, а Allocator (необязательный параметр) – объект отвечающий за выделение памяти. По умолчанию используется шаблон класса allocator, который универсален и работает по простейшему алгоритму.  
**Пример объявления списков в коде программы:**  
#include <list>  
list<int> first; // пустой список int-ов;  
list<int> second (4,100); // в списке 4 элемента со значением 100;  
list<int> third (second.begin(),second.end()); //пробегает от second.begin до second.end и записывает полученные значения в список (элемент second.begin записывается, а second.end уже нет);  
list<int> fourth (third); // копирует список third в fourth.

**Пример задания списка из массива:**  
int myints[] = {16,2,77,29};  
list<int> fifth (myints, myints + sizeof(myints) / sizeof(int) );

**Публичные функции класса(Public member functions):**  
(constructor) Создает список;  
(destructor) Удаляет список;  
operator= Копирует список (определяет оператор =, например, third=fifth, в этом случае список third удаляется, потом в него копируется содержимое списка fifth. При этом они оба становятся одинаковой длины, но хранятся в памяти отдельно, т.е. third не ссылается на fifth);

**Итераторы** (iterator и reverse\_iterator – типы, определены внутри класса)**:**  
begin Устанавливает итератор на начало списка;  
end Устанавливает итератор на конец списка;  
rbegin Устанавливает реверсный итератор на реверсное начало (то есть итератор, просматривающий список с конца на конец списка);  
rend Устанавливает реверсный итератор на реверсный конец (то есть итератор, просматривающий список с конца на начало списка);

**Память:**  
empty Проверяет список на пустоту;  
size Возвращает размер списка;  
max\_size Возвращает максимально достижимый размер списка судя по оставшейся памяти и библиотечным ограничениям (если такие есть);  
resize Меняет размер;

**Доступ к значению:**  
front К значению первого элемента (именно к значению, как к обычной переменной);  
back К значению последнего элемента;

**Библиотека <list>**

Имя функции Описание

pop\_front удалить элемент в начале

pop\_back удалить элемент в конце

push\_front добавить элемент в начала

push\_back добавить элемент в конец

front обратится к первому элементу

back обратиться к последнему элементу

insert добавить элемент в какое-то место

copy вывести все элементы списка (и не только)

unique удалить все дубликаты

merge добавление другого списка

**insert**

С помощью его можно добавить новый элемент в любую часть контейнера (в нашем случае для списка). Вот как он работает:

insert (<позиция>, <значение>);

**copy**

Вообще он имеет несколько видов применения:

1.Вывод элементов.

2.Запись элементов.

А также копирования какого-то количества ячеек и вставка их в позицию Y.

Чтобы его использовать дополнительно нужно подключить библиотеку — <iterator>

**unique**

Удаляет все повторяющиеся элементы (дубликаты). Использовать его очень просто:

myspisok.unique();

**merge**

Добавляет существующему списку еще один.

**erase**

Если вы хотите удалить какой-то промежуток элементов или всего лишь один элемент, то это функция справиться с этим на раз-два. Вот как она работает:

<список>.erase(<начальная позиция (от)>, <конечная позиция (до)>);

<позиция> = <список>.erase(<позиция>); // удаляем одну ячейку

**remove**

Чтобы удалить все элементы со значением X нужно использовать данную конструкцию <имя списка>.remove(X)

Библиотека **<iterator>:** Реализует классы и шаблоны для работы с [итераторами](https://ru.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80).