**Билеты к экзамену с ответами**

**1. Объявление переменных на языке C++**

Объявление переменной в C++ происходит таким образом: сначала указывается тип данных для этой переменной а затем название этой переменной.

**2. Стандартные типы данных**

* **int** (**4 байта**) — целочисленный тип данных.
* **float** (**4 байта**)— тип данных с плавающей запятой.
* **double (8 байт)** — тип данных с плавающей запятой двойной точности.
* **char (1 байт)** — символьный тип данных.
* **bool (1 байт)** — логический тип данных.
* **string** – массив char (класс)

**3. Операции над данными**

Математические операции и операции, реализованные функциями класса (для string и char)

**4. Вычисление выражений**

Для вычисления математических выражений используются библиотеки cmath и math.h.

**5. Преобразования типов**

В C++ различают явное и неявное преобразование [типов данных](http://cppstudio.com/obuchenie_cpp/tipy_dannyh). Неявное преобразование типов данных выполняет компилятор С++, ну а явное преобразование данных выполняет сам программист. О преобразовании типов данных скажу следующее: «Результат любого вычисления будет преобразовываться к наиболее точному типу данных, из тех типов данных, которые участвуют в вычислении».

Явное преобразование – указание типа данных или же запись вида «15.0» - вещественный тип.

Функция atoi библиотеки cstdlib преобразует char в int.

**6. Алгоритмы и способы из их записи**

Алгоритмом называется точная инструкция исполнителю в понятной для него форме, определяющая процесс достижения поставленной цели на основе имеющихся исходных данных за конечное число шагов.

Основными свойствами алгоритмов являются:

1. Универсальность (массовость) - применимость алгоритма к различным наборам исходных данных.

2. Дискретность - процесс решения задачи по алгоритму разбит на отдельные действия.

3. Однозначность - правила и порядок выполнения действий алгоритма имеют единственное толкование.

4. Конечность - каждое из действий и весь алгоритм в целом обязательно завершаются.

5. Результативность - по завершении выполнения алгоритма обязательно получается конечный результат.

6. Выполнимость - результата алгоритма достигается за конечное число шагов.

Алгоритм считается правильным, если его выполнение дает правильный результат.

**Способы записи алгоритмов**

Для записи алгоритмов используют самые разнообразные средства. Выбор средства определяется типом исполняемого алгоритма. Выделяют следующие основные способы записи алгоритмов:

- вербальный, когда алгоритм описывается на человеческом языке;

- символьный, когда алгоритм описывается с помощью набора символов;

- графический, когда алгоритм описывается с помощью набора графических изображений.

Общепринятыми способами записи являются графическая запись с помощью блок-схем и символьная запись с помощью какого-либо алгоритмического языка.

**7. Управляющие структуры: линейный, циклический и ветвящийся**

Линейный – блок операторов, разделенных пустым оператором «;».

Ветвящийся – операторы выбора (if, if else, switch case).

Циклический – операторы повторения (for, while, do while).

**8. Операторы языка С++**

Пустой - ;

Составной – {}

Условный – if (выражение) else (выражение)

Переключатель - **switch** (*<выражение>*)

{ **case** *<константное выражение 1>*: *<операторы 1>*

**case** *<константное выражение 2>*: *<операторы 2>*

**...**

**case** *<константное выражение N>*: *<операторы N>*

[**default**: *<операторы>*]

}

Оператор цикла с предусловием - **while** (*<выражение>*) *<оператор>*

Оператор цикла с постусловием - **do** *<оператор>* **while** *<выражение>*;

Оператор пошагового цикла- **for** ([*<начальное выражение>*];

[*<условное выражение>*];

[*<выражение приращения>*])

*<оператор>*

Оператор разрыва – break

Оператор возврата – return

**9. Объявление одномерного и многомерного массива**

int a[16]; int a[5][3];

int a[16] = { 5, -12, -12, 9, 10, 0, -9, -12, -1, 23, 65, 64, 11, 43, 39, -15 }; // инициализация одномерного массива

int a[5][3] = { {4, 7, 8}, {9, 66, -1}, {5, -5, 0}, {3, -3, 30}, {1, 1, 1} } – двумерного массива

Динамический массив – через указатель:

int \*mas = new int[n];

int \*\*mas = new int\*[n]

for (int i = 0; i <n; i++)

mas[i] = new int[n];

**10. Операции с массивами**

Сортировки, очистка, заполнение – все реализуется с помощью функций собственного написания.

sizeoff – длина в байтах объекта в [].

**11. Типовые алгоритмы работы с массивами**

Объявление, инициализация, поиск, сортировка, передача в функцию через указатель, по адресу.

**12. Объявление переменных-строк**

string (string) – массив элементов char.

char string[10];

string string;

Ввод – cin.getline(string, n);

**13. Операции над строками**

| **Функция** | **Пояснение** |
| --- | --- |
| [**strlen**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cstring-string-h/funkciya-strlen/)**(**имя\_строки**)** | определяет длину указанной строки, без учёта нуль-символа |
| **Копирование строк** | |
| [**strcpy**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cstring-string-h/funkciya-strcpy/)**(**s1,s2**)** | выполняет побайтное копирование символов из строки  s2 в строку s1 |
| [**strncpy**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cstring-string-h/funkciya-strncpy/)**(**s1,s2, n**)** | выполняет побайтное копирование n символов из строки  s2 в строку s1. возвращает значения s1 |
| **Конкатенация строк** | |
| [**strcat(**s1,s2**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cstring-string-h/funkciya-strcat/) | объединяет строку s2 со строкой s1. Результат сохраняется в s1 |
| [**strncat(**s1,s2,n**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cstring-string-h/funkciya-strncat/) | объединяет n символов строки s2 со строкой s1. Результат сохраняется в s1 |
| **Сравнение строк** | |
| [**strcmp(**s1,s2**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cstring-string-h/funkciya-strcmp/) | сравнивает строку s1 со строкой s2 и возвращает результат типа int: 0 –если строки эквивалентны, >0 – если s1<s2,  <0  — если s1>s2 С учётом регистра |
| [**strncmp(**s1,s2,n**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cstring-string-h/funkciya-strncmp/) | сравнивает n символов строки s1 со строкой s2 и возвращает результат типа int: 0 –если строки эквивалентны, >0 – если s1<s2,  <0  — если s1>s2 С учётом регистра |
| **stricmp(**s1,s2**)** | сравнивает строку s1 со строкой s2 и возвращает результат типа **int**: 0 –если строки эквивалентны, >0 – если s1<s2,  <0  — если s1>s2 Без учёта регистра |
| **strnicmp(**s1,s2,n**)** | сравнивает n символов строки s1 со строкой s2 и возвращает результат типа int: 0 –если строки эквивалентны, >0 – если s1<s2,  <0 — если s1>s2 Без учёта регистра |
| **Обработка символов** | |
| [**isalnum(**c**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cctupe-ctupe-h/funkciya-isalnum/) | возвращает значение true, если с является буквой или цифрой, и false в других случаях |
| [**isalpha(**c**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cctupe-ctupe-h/funkciya-isalpha/) | возвращает значение true, если с является буквой,  и false в других случаях |
| [**isdigit(**c**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cctupe-ctupe-h/funkciya-isdigit/) | возвращает значение true, если с является цифрой, и false в других случаях |
| [**islower(**c**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cctupe-ctupe-h/funkciya-islower/) | возвращает значение true, если с является буквой нижнего регистра, и false в других случаях |
| [**isupper(**c**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cctupe-ctupe-h/funkciya-isupper/) | возвращает значение true, если с является буквой верхнего регистра, и false в других случаях |
| [**isspace(**c**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cctupe-ctupe-h/funkciya-isspace/) | возвращает значение true, если с является пробелом, и false в других случаях |
| [**toupper(**c**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cctupe-ctupe-h/funkciya-toupper/) | если символ с, является символом нижнего регистра, то функция возвращает преобразованный символ с в верхнем регистре, иначе символ возвращается без изменений. |
| **Функции поиска** | |
| [**strchr(**s,c**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cstring-string-h/funkciya-strchr/) | поиск первого вхождения символа св строке s**.**В случае удачного поиска возвращает указатель на место первого вхождения символа с**.**Если символ не найден, то возвращается ноль. |
| [**strcspn(**s1,s2**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cstring-string-h/funkciya-strcspn/) | определяет длину начального сегмента строки s1, содержащего те символы, которые не входят в строку s2 |
| [**strspn(**s1,s2**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cstring-string-h/funkciya-strspn/) | возвращает длину начального сегмента строки s1, содержащего только те символы, которые входят в строку s2 |
| **strprbk(**s1,s2**)** | Возвращает указатель  первого вхождения любого символа строки s2 в строке s1 |
| **Функции преобразования** | |
| [**atof(**s1**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cstdlib-stdlib-h/funkciya-atof/) | преобразует строку s1 в тип double |
| [**atoi(**s1**)**](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cstdlib-stdlib-h/funkciya-atoi/) | преобразует строку s1 в тип int |

**14. Алгоритмы работы со строками**

Смотри выше. Вообще не понимаю, че это за вопрос. Все варианты – комбинации функций строк. Векторы мы же не проходили.

**15. Библиотечные функции работы с текстом**

Ээээ, че?

**16. Объявление указателей**

Указатель – переменная, значением которой является адрес ячейки памяти. Указатель может ссылаться на переменную или функцию. Для этого нужно знать адрес переменной или функции. Так вот, чтобы узнать адрес конкретной переменной в С++ существует унарная операция взятия адреса &.

int var = 123; // инициализация переменной var числом 123

int \*ptrvar = &var; // указатель на переменную var (присвоили адрес переменной указателю)

cout << "&var    = " << &var << endl;// адрес переменной var содержащийся в памяти, извлечённый операцией взятия адреса

cout << "ptrvar  = " << ptrvar << endl;// адрес переменной var, является значением указателя ptrvar

cout << "var     = " << var << endl; // значение в переменной var

cout << "\*ptrvar = " << \*ptrvar << endl; // вывод значения содержащегося в переменной var через указатель, операцией разименования указателя

**18. Указатели в массивах и строках**

Массивы – смотри раздел массивов.

Строки – char \*string = new char [n];

Указывает на первый элемент массива char.

**19. Объявление и описание функций**

|  |
| --- |
| returnDataType functionName( dataType argName1, dataType argName2, ..., dataType argNameN), где: |

returnDataType — возвращаемый тип данных

functionName — имя функции

dataType — тип данных

argName1...N — имена параметров функции (количество параметров неограниченно)

**20. Передача параметров по ссылке и значению**

В функцию параметры могут передаваться как по значению, так и по ссылке.

При передаче параметров по значению они при выходе из функции не изменятся.

**...**

**int func(int k){**

**k\*=2;**

**return k;**

**}**

**void main(){**

**int z=1, y=3, k;**

**k=func(z)+func(y);**

**cout<<z<<" "<<y;**

...

При передаче же параметров по ссылке при выходе из функции из значения могут измениться. Как, например, в следующем фрагменте:

**...**

**int func(int &k){**

**k\*=2;**

**return k;**

**}**

**void main(){**

**int z=1, y=3, k;**

**k=func(z)+func(y);**

**cout<<z<<" "<<y;**

...

Разница между двумя этими фрагментами весьма невелика - всего 1 символ амперсанда (&) в строке: int func(int &k)

**21. Функции с параметром по умолчанию**

При обращении к функции, можно опускать некоторые её аргументы, но для этого необходимо при объявлении прототипа данной функции проинициализировать её параметры какими-то значениями, эти значения и будут использоваться в функции по умолчанию. Аргументы по умолчанию должны быть заданы в прототипе функции.  Если в функции несколько параметров, то параметры, которые опускаются должны находиться правее остальных. Таким образом, если опускается самый первый параметр функции, то все остальные параметры тоже должны быть опущены. Если опускается какой-то другой параметр, то все параметры, расположенные перед ним, могут не опускаться, но после него они должны быть опущены. Разработаем программу, в которой объявим функцию с аргументами по умолчанию.

#include "stdafx.h"

#include <iostream>

#include <cmath> // описывает работу математической функции sqrt() - квадратный корень

using namespace std;

double heron\_space(const double a = 5, const double b = 6.5, const double c = 10.7); //параметры функции инициализированы по умолчанию

int main()

{

    cout << "S = " << heron\_space()     << endl << endl; // все параметры используются по умолчанию

    cout << "S = " << heron\_space(10,5) << endl << endl; // только последний параметр используется по умолчанию

    cout << "S = " << heron\_space(7)    << endl << endl; // два последних параметра берутся по умолчанию, а первый равен 7

    system("pause");

    return 0;

}

double heron\_space(const double a, const double b, const double c) // функция вычисления площади треугольника по формуле Герона

{

    const double p = (a + b + c) / 2;   // полупериметр

    cout << "a = " << a << "\nb = " << b << "\nc = " << c << endl;

    return (sqrt(p \* (p - a) \* (p - b) \* (p - c))); // формула Герона для нахождения площади треугольника

}

**22. Перегрузка функций**

Под перегрузкой функции понимается, определение нескольких функций (две или больше) с одинаковым именем, но различными параметрами. Наборы параметров перегруженных функций могут отличаться порядком следования, количеством, типом. Таким образом перегрузка функций нужна для того, чтобы избежать дублирования имён функций, выполняющих сходные действия, но с различной программной логикой.

У нас есть две функции с разной сигнатурой, но одинаковыми именами (перегруженные функции). Сигнатура – это комбинация имени функции с её параметрами. Как же вызывать эти функции? А вызов перегруженных функций ничем не отличается от вызова обычных функций.

Компилятор самостоятельно выберет нужную функцию, анализируя только лишь сигнатуры перегруженных функций. Минуя перегрузку функций, можно было бы просто объявить функцию с другим именем, и она бы хорошо справлялась со своей задачей.

**23. Шаблон функций**

Шабло́ны (англ. template) — средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).

Итак, описание шаблона начинается с ключевого слова template за которым в угловых скобках («<» и «>») следует список параметров шаблона. Далее, собственно идет объявление шаблонной сущности (например, функция или класс), т. е. имеет вид:

template < template-parameter-list > declaration.

Теперь давайте напишем шаблонную функцию my\_swap. Исходя из упомянутой выше структуры объявления шаблона следует, что наша функция будет выглядеть так:template < параметры\_шаблона > описание\_функции.

Напишем функцию:

**template** < **typename** T >

**void** **my\_swap** ( T & first , T & second )

{

T **temp**(first) ;

first = second ;

second = temp ;

}

typename в угловых скобках означает, что параметром шаблона будет тип данных. T — имя параметра шаблона. Вместо typename здесь можно использовать слово class: **template < class T >**В данном контексте ключевые слова typename и class эквивалентны (лично мне больше нравится typename, а кому-то class). Далее, в тексте шаблона везде, где мы используем тип T, вместо T будет проставляться необходимый нам тип.

**24. Форматирование вывода данных**

/n – перенос строки

/t – горизонтальная табуляция

/v – вертикальная табуляция

Форматы printf:

%d — целое число типа int со знаком в десятичной системе счисления;

%u — целое число типа unsigned int;

%x — целое число типа int со знаком в шестнадцатеричной системе счисления;

%o — целое число типа int со знаком в восьмеричной системе счисления;

%hd — целое число типа short со знаком в десятичной системе счисления;

%hu — целое число типа unsigned short;

%hx — целое число типа short со знаком в шестнадцатеричной системе счисления;

%ld — целое число типа long int со знаком в десятичной системе счисления;

%lu — целое число типа unsigned long int;

%lx — целое число типа long int со знаком в шестнадцатеричной системе счисления;

%f — вещественный формат (числа с плавающей точкой типа float);

%lf — вещественный формат двойной точности (числа с плавающей точкой типа double);

%e — вещественный формат в экспоненциальной форме (числа с плавающей точкой типа float в экспоненциальной форме);

%c — символьный формат;

%s — строковый формат.

**25. Рекурсивные алгоритмы и функции**

В программировании рекурсия тесно связана с функциями, точнее именно благодаря функциям в программировании существует такое понятие как рекурсия или рекурсивная функция. Простыми словами, рекурсия – определение части функции (метода) через саму себя, то есть это функция, которая вызывает саму себя, непосредственно (в своём теле) или косвенно (через другую функцию).  Типичными рекурсивными задачами являются задачи: нахождения n!, числа Фибоначчи. Такие задачи мы уже решали, но с использованием циклов, то есть итеративно. Вообще говоря, всё то, что решается итеративно можно решить рекурсивно, то есть с использованием рекурсивной функции.

**26. Структуры и работа с ними**

Структура — это агрегатный тип данных, так как может содержать в себе разнотипные элементы. Синтаксис объявления структуры в С++ отличается от C. Хотя версия C остается правильной для  C++. Получается, что в С++ можно двумя стилями объявления структур пользоваться, а в языке C — только одной. Смотрим синтаксис объявления структуры в языке С++:

|  |  |
| --- | --- |
| 1  2  3  4  5 | struct Name  {      type atrib;      // остальные элементы структуры  } structVar1, structVar2, ...; |

где,

* struct — ключевое слово, которое начинает определение структуры
* Name — имя структуры
* type — тип данных элемента структуры
* atrib — элемент структуры
* structVar1-2 — структурные переменные

Доступ к элементам структуры так же прост, как использование символа «точка». Предположим. что у нас есть структурная переменная с именем car и у нее есть элемент с именем speed, к которому, мы сейчас получим доступ:

|  |  |
| --- | --- |
| 1 | car.speed; |

Классы и структуры C++ почти идентичны, за исключением того факта, что структуры по умолчанию открыты для доступа, а классы — закрыты.

Что такое очереди, вы можете узнать из этой статьи. Сейчас я хочу вам рассказать про замечательный контейнер STL — <queue>. Как вы уже наверное догадались, этот контейнер реализует модель очереди. Итак, чтобы использовать в своей программе очередь, необходимо подключить следующий заголовочный файл:

|  |  |
| --- | --- |
| 1 | #include <queue> |

Давайте рассмотрим простой пример использования очереди — queue:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #include <iostream>  #include <queue>        // подключаем заголовочный файл очереди  #include <string>       // заголовочный файл для работы со строками типа string  using namespace std;    int main()  {      queue<string> myQueue;     // создаем пустую очередь типа string        // добавили в очередь несколько элементов типа string      myQueue.push("No pain ");      myQueue.push("- no gain");        cout << "Количество элементов в очереди: " << myQueue.size() << endl;      cout << "\nВот они: " << myQueue.front() << myQueue.back();        myQueue.pop(); // удаляем один элемент в очереди      cout << "\nТеперь в очереди остался один элемент: " << myQueue.front();      return 0;  } |