**ТРЕБОВАНИЯ, ПРЕДЪЯВЛЯЕМЫЕ К СДАВАЕМЫМ ПРОГРАММАМ**

**или**

**КАК ПИСАТЬ ПРОФЕССИОНАЛЬНЫЕ ПРОГРАММЫ**

Здесь изложены требования, предъявляемые к программам, написанным на языке "C/C++", которые студенты пишут на 1-м курсе.

1. **КРАТКАЯ ФОРМУЛИРОВКА ТРЕБОВАНИЙ**

1.1 Программа должна иметь осмысленное название (слово или словосочетание), не обязательно совпадающее с названием исполняемого файла, которое должно выводиться при запуске программы, а также должно присутствовать в комментарии в начале каждого файла программы.

1.2 Исходный код программы должен быть в каноническом виде:

1.2.1 Каждый файл программы должен начинаться с комментария, в котором должны быть указаны название и назначение программы и данного файла, автор и номер учебной группы.

1.2.2 Каждый оператор программы должен находиться на отдельной строке.

Исключением может быть однострочный условный оператор, например:

if( a==0 ) break;

1.2.3 Каждую фигурную скобку следует располагать на отдельной строке (открывающая фигурная скобка может располагаться в той же строке).

1.2.4 Так как язык программирования "C" основан на лексике английского языка, текст программы должен соответствовать правилам пунктуации английского языка и правилам написания математических формул. Перед операторами "запятая" и "точка с запятой" не следует ставить пробел, так же, как в английском языке не ставится пробел перед знаками препинания. Перед открывающей круглой или квадратной скобкой оператора, условия, функции или массива не следует ставить пробел, так как это принято в математике. Это не относится к фигурным скобкам составного оператора, структуры или тела функции.

1.2.5 Правильно использовать отступы. Текст программы должен начинаться с 1-го символа строки. Все операторы линейной части программы должны иметь слева одно и то же число пробелов (обычно 2 или 4, можно использовать табуляцию). При определении функции, цикла или условного оператора открывающую и закрывающую фигурные скобки следует располагать на уровне первой буквы первого слова той синтаксической конструкции, к которой они относятся. В случае условного оператора или цикла с одним оператором последний следует располагать либо на одной строке с управляющим оператором, либо на следующей строке с отступом слева от него. Величина отступов разного уровня вложенности относительно предыдущего уровня должна быть одинакова во всех файлах данной программы.

1.2.6 Блок switch следует располагать следующим образом. Все операторы case следует располагать на отдельных строках на одном уровне с предыдущими. Метки case должны быть отсортированы по возрастанию или по смыслу. При наличии метки default ее следует располагать в конце оператора switch, после меток case. Все операторы блока switch должны быть выровнены слева.

1.2.7 Оператор "точка с запятой", ограничивающий пустой цикл, должен располагаться на отдельной строке с отступом!

Иначе программист ее просто не увидит (и Вы через месяц тоже). Лучше такую конструкцию не использовать вообще.

1.2.8 Для имен макросов, а также для констант перечисляемого типа **enum** (например коды ошибок), использовать только заглавные буквы, а для всех остальных идентификаторов использовать маленькие буквы.

1.2.9 Программа должна быть снабжена необходимыми комментариями.

Излишнее комментирование (**каждой строчки кода**) не приветствуется.

Размещая комментарии справа от кода, чтение кода затрудняется, исключение только объявление переменных. Однострочные комментарии лучше размещать над строками кода:

int a, b, c; // коэффициенты квадратного уравнения

double x, y; // координаты точки

// Находим максимум в массиве

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

if( A[i] > max) **// если встретили элемент больше максимума**

max = A[i] **// заменяем max на новое значение**

}

1.2.10 Имена идентификаторов должны быть осмысленные и отсортированы по смыслу, последовательности использования или по алфавиту, при необходимости при объявлении написать комментарий.

Смотрите примеры хороших имен переменных в приложении.

1.2.11 Строки исходного текста не должны выходить за пределы экрана (не важно что оно помещалось у вас на другой машине).

1.2.12 В случае наличия пустых строк на неочевидном месте доказать необходимость их присутствия.

Пустую строку следует использовать для отделения блока кода, который решает какую-то подзадачу.

1.2.13 Не должно быть лишних круглых и фигурных скобок.

Например, если в цикле выполняется лишь один оператор, то фигурные скобки излишни, загромождают код.

1.2.14 Традиционно принято также отделять пробелом операнды (вместе со знаками унарных операций, если таковые имеются) от знаков бинарных арифметических, логических и адресных операций. Это значительно облегчает чтение программы, особенно учитывая то, что в языках C/C++ операторы могут состоять как из одного, так и из двух символов, в одном операторе могут присутствовать одновременно знаки как унарных, так и бинарных операций, а различные символьные знаки и их сочетания могут иметь разный смысл в зависимости от контекста. Сравните один и тот же оператор, записанный с пробелами, отделяющими знаки бинарных операций, и без пробелов:

if( \*ch != '=' && \*ch != '!' )

d = dm[mon]++ + (y % 4 == 0 && mon == 2 ? --x : 0);

if(\*ch!='='&&\*ch!='!')

d=dm[mon]+++(y%4==0&&mon==2?--x:0);

1.2.15 Знаки унарных операций (в том числе приведение типа), наоборот, не

следует отделять пробелом от операнда, т.к. должно быть сразу видно,

что это именно унарная операция, а не бинарная, и относится она к

тому операнду, к которому "прилипла". Примеры:

i++;

x = (double)i;

buffer = (char\*)malloc(vol);

fpuns = (unsigned far\*)cursor;

pfpuns = &fpuns;

dx = \*(unsigned\*)pfpuns;

1.3 Не применять оператор goto и метки.

В рамках лабораторных можно обойтись без них, «неопределенные» переходы в в коде запутывают логику. В реальных проектах встречается крайне редко.

1.4 Вычисления одних и тех же значений не должны повторяться.

1.5 Значения любого типа, встречающиеся в программе более одного раза, должны быть оформлены в виде констант или макросов.

1.6 Все неизменяемые значения должны быть объявлены константами или оформлены в виде макросов.

1.7 Не применять макросы для замены групп операторов.

1.8 Если имеются идентификаторы, начинающиеся с символа подчеркивания, доказать необходимость использования таких имен. Такие имена используются для полей класса.

1.9 Не изобретать велосипед. Использовать библиотечные функции, макросы и типы данных везде, где возможно. Применение своих функций, макросов и типов данных вместо библиотечных допускается в следующих случаях: 1) студенту так сформулировано задание; 2) код, написанный студентом, работает эффективнее, чем фирменный (доказательство факта лежит на студенте).

Если Вы изобрели велосипед, преподаватель вправе попросить упростить код за счет использования библиотечных функций.

1.10 В случае, если функция main возвращает нестандартное значение (int), доказать необходимость этого.

Данное требование можно оспаривать, но не забывайте про воспитание правильной привычки — соответствие кода стандартам.

1.11 Под каждую программу следует выделить отдельную директорию, название которой должно совпадать с именем исполняемого файла и в которой должны находиться только файлы, имеющие отношение к данной программе.

1.12 Компиляция должна быть без сообщений.

Самый распространенный вариант пропущенных предупреждений:

Переменная объявлена, но не используется

Функция должна возвращать какое-то значение, но не во всех случаях это происходит.

int foo( int a ){

if( a > 0 )

return 1;

}

1.13 Комментарии исходного текста программы и сообщения, выдаваемые программой при работе, не должны содержать грамматических, стилистических и фактографических ошибок, и для них следует использовать только официальную компьютерную терминологию. Не допускается использование жаргонных и нецензурных выражений. Использование несерьезных и простонародных слов допускается только в играх и других развлекательных программах.

1.14 Программа должна иметь удобный и интуитивно понятный пользовательский интерфейс. В заставке должна быть информация о названии, назначении и авторе программы. При необходимости программа должна выдавать сообщения о необходимых действиях со стороны пользователя.

1.15 По окончании работы программы вся динамически выделяемая память должна быть освобождена, все открываемые файлы должны быть закрыты!

1.16 Доказательство правильности работы программы лежит на студенте. Студент должен подготовить набор необходимых тестов, т.е различных входных данных, чтобы покрывались все режимы работы программы, в т.ч. для вырожденных случаев.

1.17 Не оптимальность алгоритма, замеченная преподавателем, должна быть исправлена.

1.18 Переменные, относящиеся к циклу, следует инициализировать непосредственно перед ним.

1.19 Не включать в заголовочные файлы тела функций.

2 **ДОПОЛНИТЕЛЬНЫЕ ТРЕБОВАНИЯ (со второй лабораторной работы)**

2.1 Программа не должна иметь глобальных переменных. Обосновать жесткую необходимость применения каждой глобальной переменной, если таковые имеются.

Глобальные переменные — это зло, поскольку их может менять кто хочет (любой модуль) и нет гарантии, что там ВСЕГДА содержится актуальная информация и никто ее не испортил. Ответственность за эту переменную никто не несет ((((

2.2 Локальные переменные не должны перекрывать аргументы функций и глобальные переменные, а аргументы функций не должны перекрывать глобальные переменные.

Перекрытие переменных — потенциальное место ошибок, которые сложно обнаруживать (только при доскональном чтении кода).

for( i …...){

for( j …...){

for( k …...)

for( i …...){

……..

}

}

}

2.3 Тело каждой функции (в том числе функции main), описание структуры данных должно занимать не более 20 строк (размер стандартной экранной страницы) исходного кода, включая комментарии и пустые строки, если таковые имеются.

Исключение - линейная функция, состоящая из блока switch. Если лимит строк исходного кода необходимо превысить для оптимизации работы программы, то каждый такой случай подлежит обсуждению с преподавателем в индивидуальном порядке. Превышение установленного лимита количества строк исходного кода функции, влечет за собой снижение оценки за сдаваемую программу.

2.4 Исходный код программы должен быть оформлен в виде проекта. Функцию main следует поместить в отдельный файл, в котором не должно быть никаких других функций. Остальные функции должны быть сгруппированы по смысловым признакам: математические - в одном файле, графические - в другом, работа с графами - в другом и т.д.

Порядок в файловой системе — порядок в голове. В большом проекте это очень важно.

2.5 Все файлы заголовков должны иметь защиту от множественного включения.

2.6 Директивы включения должны располагаться только в начале файла, их следует сортировать (по месту в иерархии системы, ниже уровень — выше позиция) и группировать. Оставляйте пустую строку между группами.

2.7 Программа должна иметь защиту от неправильного ввода данных.

Ни при каких обстоятельствах программа не должна зависать, прерываться с ошибкой (падать).

2.8 Вспомогательные функции, которые по своему смыслу могут быть использованы только в том файле, где они определены, должны быть снабжены модификатором static.

**Общие слова «Зачем?» это надо**

Главная цель введения данных требований — привить привычку писать чистый и удобочитаемый код. Это необходимо для облегчения модификации кода, его сопровождения и повторного использования. В то же время, сейчас лабораторные работы не демонстрируют эти процессы, которые в реальном мире отнимают в десятки раз больше времени и сил, чем собственно написание кода. Однако, для того, чтобы «вкусить» все прелести чтения чужого кода необходима работа в команде. *Постараемся познакомиться с этим во втором семестре.*

**Самоценность исходного кода**

Ценность работы программиста — это не только работающая корректно, оптимально, быстро, качественно программа, хотя это само по себе первоочередно и требует аккуратности и внимательности разработчика. В реальной жизни программные продукты не разрабатываются раз и навсегда, они требуют сопровождения, возможно исправления ошибок, масштабируемости решений, постоянного улучшения, добавления нового функционала. Поэтому качество исходного кода, его читабельность, расширяемость очень важны. Плохо написанный код (даже если его много), зачастую проще переписать, чем исправлять.

**Необходимость единого стиля программирования**

Не должна вызывать сомнений необходимость введения единого стиля работы в коллективе (не только программистов, но и в любом другом) при работе над одним проектом. Безболезненная стыковка результатов работы каждого в единое целое может быть осуществлена только тогда, когда весь коллектив работает по единому стандарту. Еще существует такое понятие, как корпоративный стиль, официально утверждаемый руководством предприятия.

**Промышленная эстетика**

Еще одно соображение. Мало у кого возникает сомнение в необходимости промышленной эстетики. Действительно, любая вещь, от самой простой до

очень сложной, как правило, изготавливается таким образом, чтобы не только выполнять свои функции, но и быть красивой, радовать глаз. Чтобы с ней удобно, приятно было работать. Для примера достаточно взять тот же компьютер. Нетрудно убедиться, что отвечает требованиям эстетики не только его внешний вид, но и внутреннее расположение узлов, несмотря на то, что необходимость снимать корпус возникает нечасто. Все блоки компьютера расположены упорядоченно, детали на платах стоят ровными рядами. Хотя, казалось бы, зачем: собирают компьютер один раз, модернизируют и ремонтируют нечасто. Можно было бы все вкривь и вкось поставить, ведь сборка компьютеров в значительной степени автоматизирована, а инженерам-электронщикам за то и деньги платят, чтобы они эти деньги отрабатывали тяжелым трудом, а не расслаблялись на работе от удовольствия, не правда ли?

Тогда давайте писать код не просто так, а эстетично, удобно для остальных.

Следует еще во время учебы приучить себя писать качественные

программы. Не следует относиться к лабораторным работам как к чему-то

не серьезному, мотивируя это тем, что эти программы в дальнейшем

никому не понадобятся. Программы действительно никому не понадобятся,

а вот программисты, умеющие писать качественные программы - очень

даже понадобятся.

Кто не верит, что стандарты к оформлению кода это важно и это не прихоть преподавателя, то почитайте подробное обсуждение стиля C++ на Хабре: <https://habr.com/ru/post/477722/>, <https://habr.com/ru/post/172091/>

**Примеры хороших имен переменных/функций**

| number / num / n  sum  length /len  max / min  column / col  string / str  start /st  old/new  stop  begin /end  finish  top/bottom  left/right  size  color  position / pos  level / lev  first / last  result / res  value / val | i, j, k, l, ii, jj – переменные итераторы  index /idx – индекс  flag /flg – переменная-флаг (true/false)  shift / sh – сдвиг  x, y, z – координаты  pointer /ptr – указатель  current / cur – текущий  temporary / temp / tmp – временный  previous / prev – предыдущий  next – следующий  sigma / alpha / rho / phi – греческие буквы из матем. формул  source / src – исходная (переменная)  destination / dst – место назначения | Для функций:  get/set  calc/compute  add/remove create/destroy  start/stop  insert/delete  show/hide  suspend/resume  put  print  open/close  pull/push/pop  cut  process /proc  generate / gen  initialize / init  find  is\_empty  to\_array  xy2idx / idx2xy |
| --- | --- | --- |

Use sys/lib/std

**Примеры кода**

Описание структуры:

struct palette\_type {

unsigned char size;

signed char colors[MAXCOLORS+1];

};

Инициализация массива структур:

struct item menu[] = {

/\* Highlight Char Pos \*/

0, "Quit", /\* Q 0 \*/

0, "Circles", /\* C 0 \*/

0, "Rotating Sphere", /\* R 0 \*/

0, "Tunnel", /\* T 0 \*/

0, "Spiral", /\* S 0 \*/

0, "Inverted Spiral", /\* I 0 \*/

0, "Bug", /\* B 0 \*/

0, "Adjust Window", /\* A 0 \*/

0, "Mode Change", /\* M 0 \*/

0, NULL

};

Переменные перечислимого типа с инициализацией и без, варианты:

enum EGA\_COLORS {

EGA\_BLACK = 0, /\* dark colors \*/

EGA\_BLUE = 1,

EGA\_GREEN = 2,

EGA\_CYAN = 3,

EGA\_RED = 4,

EGA\_DARKGRAY = 56, /\* light colors \*/

EGA\_LIGHTBLUE = 57,

EGA\_LIGHTGREEN = 58,

EGA\_LIGHTCYAN = 59,

EGA\_LIGHTRED = 60,

EGA\_YELLOW = 61,

EGA\_WHITE = 62

};

enum CGA\_COLORS {

CGA\_LIGHTGREEN = 1, CGA\_LIGHTRED = 2, CGA\_YELLOW = 3,

CGA\_LIGHTCYAN = 4, CGA\_LIGHTMAGENTA = 5, CGA\_WHITE = 6

};

enum put\_image\_ops { COPY\_PUT, XOR\_PUT, OR\_PUT, AND\_PUT, NOT\_PUT };

enum COLORS {

BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LIGHTGRAY, DARKGRAY,

LIGHTBLUE, LIGHTGREEN, LIGHTCYAN, LIGHTRED, LIGHTMAGENTA, YELLOW,

WHITE

};

Функция (отступ в один пробел маловато):

void text\_screen\_print(int left, int top, int right, int bottom)

{

char \*buffer;

int i, j, index;

buffer = (char\*)malloc((right - left + 1) \* (bottom - top + 1) \* 2);

gettext(left, top, right, bottom, buffer);

putc('\n', stdprn);

for(i = top, index = 0; i <= bottom; i++)

{

for(j = left; j <= right; j++, index += 2)

putc(buffer[index], stdprn);

putc('\n', stdprn);

}

free(buffer);

}

Цикл:

for( i = 0; i < 2; i++ )

{

b = 2.0 \* M\_PI \* v1;

d2 = sin(b);

g = sqrt(2.0 \* log(1.0 / v2)) \* lambda;

r2 = g \* d2;

sum += r2 \* r2;

}

Цикл с постусловием:

do

{

scanf("%lf", &s);

if( s <= 0.0 )

warn\_positive(px, py, 0, s);

} while( s <= 0.0 );