Стиль кода- это набор правил и соглашений, используемых при написании исходного кода на языке программирования. Наличие общего стиля программирования облегчает понимание и поддержание исходного кода, а также упрощает взаимодействие нескольких человек при написании программы.

Нужно уметь выбирать понятные, передающие нужный смысл названия переменных. Это необходимо просто для того, чтобы не запутаться в реализации кода. Например, привожу код программы:

/\*

ID: Din

PROG: ride

LANG: C++

\*/

#include <iostream>

#include <fstream>

#include <cstring>

using namespace std;

cons t char ans[2][5]={”GO”, “STAY”}; // использую такое название константы char, потому что “GO”, “STAY”- неизменяющиеся фразы, являющиеся ответами в данной программе.

int i, g, work1=1, work2=1, cometlen, grouplen, res1, res2 ;

//i,g- переменные, постоянно использующиеся в циклах. Для меня это две самые привычные переменные, используемые в циклах. Work1, work2- переменные, используемые для обозначения произведения числовых значений букв в строках, называющих каметы и группы. Work1- для каметы, work2- для группы людей. Cometlen, grouplen- переменные, называющие количество символов в строках. Cometlen- в названии кометы, grouplen- в названии группы людей. Res1, res2- от английского residue- остаток. Обозначает остатки от деления произведений числовых значений букв в строках на 47. Res1- для кометы. Res2- для группы людей.

unsigned char mass1, mass2;

// mass1, mass2- названия массивов char, содержащих буквы строк с названиями. Mass1-каметы, mass2- группы людей.

char comet[7], group[7];

// comet, group с английского переводятся, как Комета, группа. Использую для обозначения строк, содержащих название кометы и группы людей.

int main()

{

ifstream cin ("ride.in");

ofstream cout ("ride.out");

cin >>comet >>group;

cometlen=strlen(comet);

for (i=0; i<= cometlen -1; i++)

{

mass1=comet[i];

mass1=(int)mass1-‘A’-1;

work1=work1\*(int)mass1;

}

// cout <<work1<<"\n"; // Комментарии- пояснения к исходному тексту программы, находящиеся непосредственно внутри комментируемого кода. С точки зрения компилятора, комментарии — часть текста программы, не влияющая на её работу. Комментарии не оказывают никакого влияния на результат компиляции программы или её интерпретацию. Помимо исходных текстов программ, комментарии также применяются в языках разметки и языках описания Вызываются посредством команды «//”, чтобы ввести в комментарий одну строку, или «/\*”, чтобы открыть комментарий и «\*/”, чтобы закрыть комментарий. Здесь комментарий используется для того, чтобы вывести на экран work1-произведение числовых значений букв в составе названия кометы. Так как по условию, в ответе программы нам необходимо вывести «GO” или “STAY” и просто для удобства и уменьшения шанса запутаться, в конечном коде необходимо убрать этот промежуточный вывод. Но иногда, чтобы найти ошибку, приходится снова писать команду. Поэтому бывает очень удобно просто поместить команду в комментарий.

grouplen=strlen(group);

for (g=0; g<= grouplen -1; g++)

{

mass2=group[g];

mass2=(int)mass2-‘A’-1;

work2=work2\*(int)mass2;

}

// cout <<work2<<"\n";

res1=work1%47;

res2=work2%47;

//if (res1==res2)

cout <<ans[res1==res2 ]<<”\n";

//}else {cout <<ans[1] <<”\n";}

return 0;

}

Единственное отличие констант от переменных, что после инициализации их нельзя изменить. Константы удобны и при желании улучшить понимание своей программы для других, ведь смысловое имя более понятно чем какое-то число.

Венгерская нотация- это соглашение об именовании переменных, констант и прочих идентификаторов в коде программ. Назвал Чарльз Симони. Является стандартом Microsoft. Суть Венгерской табуляции заключается в том, что переменные начинаются с заранее обговоренных префиксов. Причем нет общих устойчивых префиксов в программировании, у каждого программиста они могут быть свои.

В своих программах Венгерской табуляцией не пользуюсь, так как считаю, что это возможность программистов высокого уровня, а меня она чаще только путает. Из своих программ могу привести следующие примеры: префикс n и префикс i являются показателями того, что данная переменная относится к типу int.

Стиль форматирования- стиль записи кода программы. Каждому программисту удобно по-своему оформлять код. И каждый старается создать код, смысл которого будет сразу виден невооруженным глазом, не нужно будет вдумываться и мысленно переставлять части программы. Для того, чтобы коды был «приятен» для восприятия не нужно жалеть пробелов и табуляций (переводов строки). Например:

using namespace std;

inti, g, sum1=1, sum2=1, y, x, rez1, rez2 ;

unsigned char mass1, mass2;

char comet[7], group[7];

//Описываем переменные разных типов. Переменный разных типов записываем в разных строках. Раздел описания переменных разделяем от других разделов пустой строкой.

intmain()

{

// Начинается основная часть программы. Запись intmain(){ return 0;} записывает на одном уровне, а все содержание основной части с небольшим отступом.

ifstreamcin ("ride.in");

ofstreamcout ("ride.out");

cin>>comet>>group;

// Описываем файлы вводы и вывода данных. И обозначаем ввод. Этот отдел также отделяем от других пустыми строками.

y=strlen(comet);

for (i=0; i<=y-1; i++)

{

mass1=comet[i];

mass1=(int)mass1-64;

sum1=sum1\*(int)mass1;

}

// cout<<sum1<<"\n";

// В данной части программы работа идет только со строкой comet, поэтому для удобства я тоже отделяю ее от других частей программы пустыми строками. Также эта часть программы содержит цикл. Содержание цикла записывают с небольшим отступом.

x=strlen(group);

for (g=0; g<=x-1; g++)

{

mass2=group[g];

mass2=(int)mass2-64;

sum2=sum2\*(int)mass2;

}

// cout<<sum2<<"\n";

// В данной части программы работа идет только со строкой group, поэтому для удобства я тоже отделяю ее от других частей программы пустыми строками.

rez1=sum1%47;

rez2=sum2%47;

//Здесь работаем с переменнами типа int, хотя до этого работали со строками, поэтому также отделяют эту часть программы пустыми строками.

if (rez1==rez2) {cout<<"GO\n";}

else {cout<<"STAY\n";}

// Вывод. Отделяю пустыми строками.

return 0;

}

Не нужно жалеть места. Чтобы самому не путаться в своем же коде нужно всегда четко разделять части программы, отдельные команды писать в отдельных строках (хотя бы для того, чтобы не переставлять команды, если понадобится поставить комментарий)

Также очень удобно бывает делить программу на модули. Модуль- это законченный фрагмент программы, своеобразная подзадача, используемая для решения главной задачи. Привожу пример, как бы выглядела приведенная мною выше программа с использованием некоторых модулей.

void in()

{

ifstreamcin ("ride.in");

cin>>comet >>group;

}

void solution()

{

y=strlen(comet);

for (i=0; i<=y-1; i++)

{

mass1=comet[i];

mass1=(int)mass1-64;

sum1=sum1\*(int)mass1;

}

// cout<<sum1<<"\n";

x=strlen(group);

for (g=0; g<=x-1; g++)

{

mass2=group[g];

mass2=(int)mass2-64;

sum2=sum2\*(int)mass2;

}

// cout<<sum2<<"\n";

rez1=sum1%47;

rez2=sum2%47;

}

void out()

{

ofstreamcout ("ride.out");

if (rez1==rez2) {cout<<"GO\n";}

else {cout<<"STAY\n";}

}

int main()

{

in();

solution();

out();

return 0;

}

// void in()- модульввода, void solution()- модульрешенияи void out()- модульвывода. Использую тип void, так как данные модули ничего не возвращают. То есть сначала я описываю все модули, а уже потом в функции intmain() в нужном порядке вызываю их.