Как писать программы без ошибок

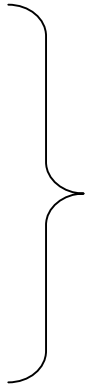
(Практическое пособие)

«Учите матчасть!»

- Контроллер
- Схемотехника
- Язык программирования
- Особенности компилятора

Этапы программирования

- Согласование ТЗ
- Формализация задачи
- Планирование
- Кодирование
- Отладка
- Тестирование
- Оформление документации
- Эксплуатация и сопровождение



этапы программирования

Планирование программы

- Расписать алгоритм
- Продумать модули
- Продумать данные
- Разделить периферию контроллера между процессами
- Учесть физические свойства обвеса
- Предусмотреть возможность расширения

Модули

- Функциональное разбиение
- Наличие готовых решений
- Возможность использования в следующих разработках

Продумать данные

- Назначение
- Объем
- Размещение
- Область видимости

Предусмотреть возможность расширения

- **Контроллер с запасом**
  - Убедиться в наличии контроллера на замену в случае увеличения объема кода
- **Смена компилятора**
  - Уникальные возможности
  - Недокументированные особенности
  - Типы данных

Переопределять типы данных

```
#if defined(__PICC__) || defined(__PICC18__)

    typedef signed   char    S08;
    typedef signed   int     S16;
    typedef signed   long    S32;
    typedef unsigned char    U08;
    typedef unsigned int     U16;
    typedef unsigned long    U32;

#endif
```

Примечание: далее в тексте для наглядности будут использоваться стандартные типы: char, int, long.

Написание программы

- Правила кодирования
- Правила оформления

Правила кодирования

- Соблюдать модульность
- Избегать условностей
- Не делать длинных и сложных выражений
- Правила констант
  - Не использовать числовые константы
  - Задавать константам осмысленные значения
  - Соблюдать систему счисления
  - Заключать константы и операнды макросов в круглые скобки
  - Заключать тела макросов в фигурные скобки
- Правила функций
  - Объявлять прототипы для всех функций
  - Проверять входные аргументы функций на правильность
  - Возвращать функцией код ошибки
  - Не делать очень больших функций
- Использовать сторожевой таймер

Соблюдать модульность

Характеристики модулей:

- Самобытность
- Самодостаточность
- Гибкость в настройке
- (Каждой программе – своя копия модуля)

Преимущества модульности:

- Мобильность
- Легкий поиск определений функций
- Замена одного модуля другим

Правила кодирования

- Соблюдать модульность
- **Избегать условностей**
- Не делать длинных выражений
- Правила констант
- Правила функций
- Использовать сторожевой таймер

Условности: типы данных

Неправильно:

```
char Counter;
Counter = 10;
while (Counter-- >= 0) ...;
```

Правильно:

```
signed char  Coutner;
Counter = 10;
while (Counter-- >= 0) ...;
```

Условности: приведение типов

```
int   i;
void *p;
```

Неправильно:

```
p = &i;
```

Правильно:

```
p = (void*)&i;
```

Побайтовое обращение к многобайтовой переменной

```
unsigned char lo, hi;
unsigned int  ui;
```

Неправильно:

```
lo = *((unsigned char*)&ui + 0);
hi = *((unsigned char*)&ui + 1);
```

Правильно:

```
lo = ui & 0xFF;
hi = ui >> 8;
```

Условности: Определение функций

```
    myfunc ()        // неправильно
    myfunc (void)    // неправильно
int myfunc ()        // неправильно
int myfunc (void)    // Правильно
```

Условности: пустые операторы

Неправильно:

```
while (!TRMT);              // Ожидаем освобождения буфера
TXREG = Data;
```

Правильно:

```
while (!TRMT) continue; // Ожидаем освобождения буфера
TXREG = Data;
```

Условности: оператор switch

В операторе **switch** нужно:

- Определять ветку default
- В каждом case ставить **break**
- неиспользуемые **break** закрывать комментариями

```
switch (...)
{
    case 0x00:
         ...
         break;
    case 0x01:
         ...
         // break; // Поставив закомментированный
                   // break, мы даем себе понять,
                   // что после обработки условия
                   // 0x01 мы хотим перейти к коду
                   // обработки условия 0x02, а
                   // не пропустили break случайно
    case 0x02:
         ...
         break;

    default:       // Обязательная ветка, даже
         ...       // если есть уверенность, что
                   // выражение в switch всегда
                   // принимает одно из описанных
}                  // значений
```

Условности

- Неинициализированные переменные
- Скобки в сложных выражениях
- «Такая ситуация никогда не случится!»

```
typedef struct
{
    unsigned int   seconds : 6;
    unsigned int   minutes : 6;
    unsigned int   hours   : 4;
} T_CLOCK;
```

Не делать длинных и сложных выражений

Неправильно:

```
t=sqrt(p*r/(a+b*b+v))+sin(sqrt(b*b-4*a*c)/(1<<(SHIFT_CONST-1)));
```

Правильно:

```
A = p*r;
B = a + b*b + v;
C = b*b – 4*a*c;
D = 1 << (SHIFT_CONST-1);

if (B == 0) ...;            // Ошибка: «деление на 0»
if (C  < 0) ...;            // Ошибка: «корень из отр. числа»
E = A/B;
if (E  < 0) ...;            // Ошибка: «корень из отр. числа»

t = sqrt(E) + sin(sqrt(C)/D);
```

Правила кодирования

- Соблюдать модульность
- Избегать условностей
- **Правила констант**
  - Не использовать числовые константы
  - Задавать константам осмысленные значения
  - Соблюдать систему счисления
  - Заключать константы и операнды макросов в круглые скобки
  - Заключать тела макросов в фигурные скобки
- Правила функций
- Использовать сторожевой таймер

Не использовать числовые константы

Неправильно:

```
char String[25];
...
for (i = 0; i <= 24; i++) String[i] = ‘ ‘;
```

Правильно:

```
#define STRING_SIZE      25
...
char String[STRING_SIZE];
...
for (i = 0; i <= STRING_SIZE - 1; i++) String[i] = ‘ ‘;
```

Указывать тип константы

Неправильно:

```
#define FOSC                4000000
#define MAX_INTERATIONS       10000
#define MIDDLE_TEMPERATURE       25
```

Правильно:

```
#define FOSC                4000000L
#define MAX_INTERATIONS       10000L
#define MIDDLE_TEMPERATURE       25.0
```

Задавать константам осмысленные значения

Неправильно:

```
#define BAUDRATE   25      // 9600 for 4 MHz
...
SPBRG = BAUDRATE;
```

Правильно:

```
#define FOSC         4000000L
#define BAUDRATE        9600L
...
SPBRG = ((FOSC + BAUDRATE*8) / (BAUDRATE*16) – 1;
```

Проверка правильности констант

```
// Задание параметров

#define FOSC          4000000L
#define BAUDRATE         9600L

// Вычисление ошибки (в процентах)

#define SPBRG_CONST    ((FOSC + BAUDRATE*8) / (BAUDRATE*16) - 1)
#define REAL_BAUDRATE  ((FOSC + (SPBRG_CONST+1)*8)/((SPBRG_CONST + 1)*16))
#define BAUDRATE_ERROR (100L * ((BAUDRATE - REAL_BAUDRATE)) / (BAUDRATE))

// Проверка ошибки на диапазон -2%..+2%

#if BAUDRATE_ERROR < -2 || BAUDRATE_ERROR > 2
    #error "Неправильно задана константа BAUDRATE"
#endif
```

Соблюдать систему счисления

Неправильно:

```
TRISA = 21;    // RA0,RA2,RA4 – входы, RA1, RA3 - выходы
TRISB = 65;    // 65 = 0x41 = 0b01000001

TMR1H = 0xF0;
TMR1L = 0xD8;  // Отсчет 10000 тактов
```

Правильно:

```
#define TRISA_CONST     0b00010101      // RA0,RA2,RA4 – входы
#define TRISB_CONST     0b01000001      // RB0, RB6 - входы
#define TMR1_WRITE(t) { TMR1H = t >> 8; TMR1L = t & 0xFF;}
...
TRISA = TRISA_CONST;
TRISB = TRISB_CONST;

TMR1_WRITE(-10000);  // Отсчет 10000 тактов
```

Заключать константы в круглые скобки

Неправильно:

```
#define PIN_MASK_1      1 << 2
#define PIN_MASK_2      1 << 5
...
PORTB = PIN_MASK_1 + PIN_MASK_2;
```

ä

```
PORTB = 1 << 2 + 1 << 5;
```

ä

```
PORTB = 1 << (2 + 1) << 5 = 1 << 8
```

Заключать константы в круглые скобки

Правильно:

```
#define PIN_MASK_1      (1 << 2)
#define PIN_MASK_2      (1 << 5)
...
PORTB = PIN_MASK_1 + PIN_MASK_2;
```

ä

```
PORTB = (1 << 2) + (1 << 5);
```

.

Заключать операнды макросов в круглые скобки

Неправильно:

```
#define MUL_BY_3(a)    a * 3
...
i = MUL_BY_3(4 + 1);
        ä
i = 4 + 1 * 3;    // = 7
```

Правильно:

```
#define MUL_BY_3(a)  (a) * 3
...
i = MUL_BY_3(4 + 1);
        ä
i = (4 + 1) * 3;  // = 15
```

Заключать тела макросов в фигурные скобки

Неправильно:

```
#define I2C_CLOCK()    NOP(); SCL = 1; NOP(); SCL = 0;
...
if (...) I2C_CLOCK();
```

```
if (...) NOP(); SCL = 1; NOP(); SCL = 0;
```

Правильно:

```
#define I2C_CLOCK()   { NOP(); SCL = 1; NOP(); SCL = 0; }
```

Правила кодирования

- Соблюдать модульность
- Избегать условностей
- Правила констант
- **Правила функций**
  - Объявлять прототипы для всех функций
  - Проверять входные аргументы функций
  - Возвращать код ошибки
  - Не делать очень больших функций
- Использовать сторожевой таймер

Проверять входные аргументы

- Инициализированные указатели
- Область допустимых значений
- Соответствие реальности

Написание программы

- Правила кодирования
- **Правила оформления**

Оформление

- Именование идентификаторов
- Форматирование текста
- Комментирование

Удобный инструментарий

Удобный редактор поможет избавиться от рутины, автоматически производя:

- Горизонтальные отступы
- Замену символа табуляции пробелами
- Подсветку синтаксиса
- Контекстную подстановку
- Перекрестные ссылки внутри исходного текста или целого проекта
- Вызов компилятора для сборки проекта из редактора
- и др.

(Рекомендую использовать SlickEdit)

Именование идентификаторов

- Объекты
  - Функции
  - Константы (и макросы)
  - Переменные
  - Типы
- Содержание имени
  - Имя должно быть осмысленным
  - Имя должно быть содержательным

Именование функций

<модуль>_<действие>_<объект>[_<суффикс>]

- **модуль**
  - **имя (или сокращенное имя) модуля, в котором определена функция**;
- **действие**
  - **глагол, определяющий назначение функции**
- **объект**
  - **существительное, определяющее параметрическую составляющую функции**
- **суффикс**
  - **необязательное поле, отражающее какую-либо дополнительную характеристику функции (rom, timeout).**

Пример имен функций

Неправильные:

```
CopyOneByteFromROMToRAM
Check
compare_and_get_max
WriteByte
```

Правильные:

```
i2c_write_byte
eeprom_write_byte
lcd_write_byte
```

Именование констант

- Заглавными буквами
- Слова разделяются символом ‘_’
- (Префикс в виде имени модуля или функционального назначения)

```
#define I2C_DEV_ADDR          0xA0
#define I2C_ADDR_WIDTH          16
```

Именование типов

- Заглавными буквами
- Слова разделяются символом ‘_’
- Префикс: “T_” или “TYPE_”

```
typedef struct
{
    unsigned long   seconds : 6;
    unsigned long   minutes : 6;
    unsigned long   hours   : 5;
} T_CLOCK;
```

Именование переменных

- Слова начинаются с заглавной буквы
- Пишутся без разделителя

```
unsigned char    BytesCounter;
signed   long    XSize, YSize;
```

Венгерская нотация

Для анализа этого кода требуется найти определение переменной Counter:

```
for (Counter  = 0; Counter  < 40000; Counter++) ...;
```

Этот код можно анализировать, не глядя в определение переменной wCounter:

```
for (wCounter = 0; wCounter < 40000; wCounter++)...;
```

Венгерская нотация

- Префикс области видимости
  - Без префикса – локальная или параметр функции
  - s_ - статическая
  - m_ - локальная для модуля
  - g_ - глобальная
- Префикс типа
  - uc – unsigned char
  - sc – signed char
  - ui – unsigned int (n)
  - si – signed int (w)
  - И т.д.

```
s_ucBytesCounter = 160;
```

Венгерская нотация

Преимущества:

- Предотвращает ошибки использования типов
- В большом коде помогает не запутаться в переменных
- Упрощает чтение чужого кода

Недостатки :

- Ухудшает читабельность кода
- Изменение типа ⟹ изменение имени
- Префикс не дает гарантии правильности задания типа

```
static signed char s_ucBytesCounter;
```

Оформление

- Именование идентификаторов
- **Форматирование текста**
- Комментирование

Форматирование

Для примера рекомендую ознакомиться с **AN2000.PDF** от micrium

- Текст файла должен быть разбит на секции
- Вертикальное выравнивание
- Горизонтальные отступы
- Не делать в строке больше символов, чем помещается на одном экране
- Одна строка – одно действие
- Разделять функциональные узлы или конструкции (for, if, …) пустыми строками
- Пробелы между операндами и операциями

Текст файла разбивать на секции

- Заголовок файла
- Хронология изменений
- Включаемые файлы
- Определения констант
- Определения макросов
- Определения типов
- Определения переменных
- Прототипов функций
- Описания функций

Правила оформления секций

- Каждая секция должна содержать только свои описания
- Секция должна быть единой
- Секции предшествует хорошо заметный блок комментария с ее названием

Вертикальное выравнивание

```
static unsigned char    BytesCounter;
static signed   int     Timer;
                double  Price;
                char   *p, c;
...
{
    BytesCounter =  0;
    Timer        = 10;
    Price        =  1.12;
    p            = &c;
}
```

Не делать длинных строк

```
int sendto(
            SOCKET                   s,
            const char              *buf,
            int                      len,
            int                      flags,
            const struct sockaddr *to,
            int                      tolen
          );
```

Пустые строки, пробелы между операндами и операциями, одна строка – одно действие

Неправильно:

```
for(i=0;i<10;i++)a[i]=0;
if(j<k){a[0]=1;a[1]=2;}else{a[0]=3;a[1]=4;}
```

Правильно:

```
for (i = 0; i < 10; i++) a[i] = 0;

if (j < k)
{
    a[0] = 1;
    a[1] = 2;
}
else
{
    a[0] = 3;
    a[1] = 4;
}
```

Оформление

- Именование идентификаторов
- Форматирование текста
- **Комментирование**

Почему не пишут комментарии

- «Время поджимает, писать некогда»
- «Это временный код, его не нужно комментировать»
- «Я и так все запомню»
- «Моя программа понятна и без комментариев»
- «В код, кроме меня, никто не заглядывает»
- «Комментарии делают текст пестрым и затрудняют чтение самой программы»
- «Я потом откомментирую»

Что должно быть в комментариях

- Спецификация функций
- Назначение переменной, константы, типа, макроса
- Краткое, емкое, безызбыточное описание действия или пояснение к нему
- Пометки о вносимых изменениях
- Указание отладочных узлов и временных конструкций

Спецификация функций

```
/*************************************************************
 *
 *  Function:       rs_buf_delete
 *
 *-----------------------------------------------------------
 *
 *  description:    Удаляем N байт из буфера
 *
 *  parameters:     uchar N – количество удаляемых байтов
 *
 *  on return:      void
 *
 *************************************************************/

void rs_buf_delete (uchar N)
{
    ...
```

Пометки о вносимых изменениях

```
/*****************************************************
 * Список изменений
 * ...
 * Версия 1.6 (22.10.2009):
 *     1. ...
 *     2. ...
 *     ...
 *     8. Иванов И.И., 17.09.2009: В функции
 *        rs_buf_delete добавлена проверка
 *        входного аргумента на 0
 * ...
 *****************************************************/
...
void rs_buf_delete (signed char N)
{
    // *1.6-8* if (N < 0) return;
    if (N <= 0) return;          // *1.6-8*
    ...
```

Указание отладочных узлов

```
void rs_buf_delete (signed char N)
{
    if (N <= 0) return;

    /*DEBUG*/  DebugCounter++;
    /*DEBUG*/  pin_DEBUG = 1;
    /*DEBUG*/  delay_ms(5);
    /*DEBUG*/  pin_DEBUG = 0;
    ...
```

Чего в комментариях быть **не должно**

1. Эмоций

```
RB0 = 1;            // Без этой хрени не работает
```

2. Описания устаревших действий

```
if (BufSize > 0) // Буфер не пуст, флаг
                 // разрешения установлен
```

3. Дублирования действия

```
BufSize = 0;     // Обнуляем переменную,
                 // показывающую размер буфера
```

Чего в комментариях быть **не должно**

4. Бесполезной информации

```
if (N < 15)       // Лучший вариант сравнения
```

5. Непонятных сокращений и жаргона:

```
A = X / Y + Z;   // В (*1*), т.н.у., обход
```

6. Ложной или вводящей в заблуждение информации:

```
if (timer < 100 && V < 10) // Если за 100мс напряжение
                           // стало выше 10 вольт
```

Расположение комментариев

Неправильный подход:

```
/* Инициализация портов*/
pin_DATA = 0;
pin_CLC = 0;

/* Очистка буфера */
for (i = 0; i < BUF_SIZE; i++) Buf[i] = 0;

/*Ожидание данных */
While (ReadData())
{
    ...
```

Расположение комментариев

Правильный подход:

```
pin_DATA = 0;                       /* Инициализация портов */
pin_CLC = 0;

for (i = 0; i < BUF_SIZE; i++)      /* Очистка буфера       */
    Buf[i] = 0;

while (ReadData())                  /* Ожидание данных      */
{
    ...
```

Многострочные комментарии

Неправильный подход:

```
/* Эта функция считывает начальные установки из EEPROM,
   проверяя контрольную сумму. Если контрольная сумма не
   сошлась, то будут приняты установки по умолчанию. */
   for (i = 0; i < BUF_SIZE; i++) ...;
```

Правильный подход:

```
/* Эта функция считывает начальные установки из EEPROM,    */
/* проверяя контрольную сумму. Если контрольная сумма не   */
/* сошлась, то будут приняты установки по умолчанию.       */
for (i = 0; i < BUF_SIZE; i++) ...;
```

Многострочные комментарии

Еще правильные варианты:

```
/*
 * Эта функция считывает начальные установки из EEPROM,
 * проверяя контрольную сумму. Если контрольная сумма не
 * сошлась, то будут приняты установки по умолчанию.
 */
for (i = 0; i < BUF_SIZE; i++) ...;
```

```
// Эта функция считывает начальные установки из EEPROM,
// проверяя контрольную сумму. Если контрольная сумма не
// сошлась, то будут приняты установки по умолчанию.
for (i = 0; i < BUF_SIZE; i++) ...;
```

Содержательная часть комментария

В данном примере приходится комментировать каждый **case** и **switch:**

```
switch (result)              // Проверяем состояние модема
{                            //
    case 0:                  // Модем выключен
         ...                 //
         break;              //
    case 1:                  // Модем готов к работе
         ...                 //
         break;              //
    case 2:                  // Модем производит дозвон
         ...                 //
         break;              //
    ...                      //
}
```

Содержательная часть комментария

Задав осмысленные имена переменной и констант, мы избавляемся от лишних комментариев:

```
switch (ModemState)                //
{                                  //
    case MODEM_OFF:                //
         …                         //
         break;                    //
                                   //
    case MODEM_READY:              //
         …                         //
         break;                    //
                                   //
    case MODEM_CALLING:            //
         …                         //
         break;                    //
    …                              //
}
```

Формулировка комментария

Совет:

- Формулируя комментарий, всегда представляйте, как будто комментарий читает другой человек. Это поможет сделать формулировку более четкой.

Отладка и тестирование

- Инструменты
- Резерв по ресурсам
- Заглушки и тестеры
- Компиляция
- Вывод отладочной информации
- Возможность блокировки вывода
- Резервное копирование

Инструменты

- Отладчик
- Симулятор
- Логический анализатор
- Осциллограф
- Мультиметр
- и пр.

Резерв по ресурсам

Нужен запас:

- периферии
  - Внутренняя периферия контроллера
    - USART, порты I/O, таймеры и пр.
  - Внешняя периферия
    - EEPROM, светодиоды, кнопки, LCD
- памяти ROM и RAM
- скорости

«Заглушки» и «тестеры»

Заглушки используются, когда:

- Нужны результаты ненаписанных подпрограмм
- При тестировании без внешних устройств
- Тестируется код, содержащий долговыполняемые функции, не участвующие в тесте подпрограммы

Тестеры используются:

- Для автоматической и полуавтоматической проверки соответствия функций спецификации

Компиляция: не игнорировать предупреждения

```
signed   int a;
unsigned int b;
if (a > b) ...; // Warning:
                // signed and unsigned comparison
```

Избавляемся от предупреждения:

```
if ((signed long)a > (signed long)b) ...;
```

Компиляция: не игнорировать предупреждения

```
if (a == b) …;
if (a = b)  …; // Warning:
               // Assignment inside relational expression
```

Избавляемся от предупреждения:

```
a = b;
if (a) …;
```

Блокировка вывода отладочной информации

- **на этапе компиляции**

```
#define DEBUG_ENABLE
...
/*D*/ #ifdef DEBUG_ENABLE
/*D*/ pin_DEBUG = 1;
/*D*/ #endif
```

- **на программно-аппаратном уровне**

```
/*D*/ if (pin_DEBUG_ENABLE) pin_DEBUG = 1;
```

Блокировка заглушек

```
char * GetGPSData (void)
{
#ifdef DUMMY_GPS_IN
    char test_string[] = “$GPRMC,A,123456,…”;
    return test_string;
#else
    /* Здесь код для работы с
     * реальными данными от GPS */
#endif
}
```

Резервное копирование

- Всегда сохранять резервные копии
  - Исходные тексты
  - Файлы проекта
  - HEX-файл
- Отмечать дату и номер версии

Список литературы

- **Э. Йодан «Структурное проектирование и конструирование программ», М. Мир, 1979**
- **Б. Керниган, Р. Пайк «Практика программирования»**
- **А. Голуб «Веревка достаточной длины, чтобы… выстрелить себе в ногу»**
- **С. Макконнелл «Совершенный код», Русская редакция, 2005**
- **http://micrium.com/download/an2000.pdf**
- **http://andromega.narod.ru/doc/micrium_an_2000_rus.pdf**

(полный вариант пособия размещен на сайте www.pic24.ru)