Приложение. Некоторые справочные данные языка С.

1. Обозначения

Комментарии:

Многострочные – начинаются с символов /* и заканчиваются символами */

Однострочные – начинаются с символов //

Числа:

12 – в десятичной форме;

OxC – в шестнадцатеричной форме (префикс Ox);

0b1100 – в двоичной форме (префикс 0b);

2. Структура простейшей программы на языке С

```
#include <name.h>  // Подключение библиотеки с именем name
#include……..
……..              //Библиотек может использоваться несколько

char n;            //Описания переменных (могут быть в любом месте программы)
…….

Function  Name(x)  //Функции
{
……..

}


int main (void)    //Заголовок основной части программы
{                  //Начало основной части программы
……
……             //Здесь находится текст самой программы
……
}                  //Конец основной части программы
```

3. Стандартные типы данных языка С

Тип	Название	Размер в битах	Диапазон значений
bit	Бит	1	0, 1
char	Символ	8	-128.. 127
int	Целое число	16	-32768..32767
float	Вещественное число	32	$\pm 1{,}175.10^{-38}$.. $\pm 3{,}402\text{- }10^{38}$
long int	Длинное целое	32	-2147483648..2147483647
unsigned char	Беззнаковый символ или логическое значение	8	0..255 TRUE, FALSE
unsigned int	Беззнаковое целое	16	0..65535
unsigned long int	Беззнаковое длинное целое	32	0..4294967295

Объявления переменных:

```
int i;  //Объявление целочисленной переменной i
char cl, c2;  //Объявление символьных переменных cl и c2
int mas[10]  //Массив из десяти элементов типа int
```

4. Операторы

[ ]	Элемент массива
{}	Аналог Begin и end в Pascal
=	Присваивание
+ – * /	Арифметические операции
<, >, <=, >=	Сравнения
==, ! =	Равно, не равно
&&	Логическое «И»
| |	Логическое «ИЛИ»
++	Автоинкремент

Оператор ветвления:

```
if (условное_выражение) оператор1; else оператор2 ;
```

Условное выражение обязательно должно быть в скобках!

5. Циклические конструкции

Конструкция while:

```
while (условное_выражение)
{
// Выполнение тела цикла, если выражение истинно
}
```

Конструкция for:

Цикл for имеет следующий синтаксис.

```
for (выражение1; выражение2; выражение3)
{
// Выполнение тела цикла
}
```

Выражение1 выполняется только один раз при входе в цикл, и обычно представляет собой оператор присваивания некоторого начального значения счетчику цикла.
Выражение2 — это условное выражение, определяющее момент выхода из цикла (цикл выполняется до тех пор, пока оно равно TRUE или 1).
Выражение3 — еще один оператор присваивания, в котором обычно изменяется счетчик цикла или некоторая переменная, влияющая на выполнение условия в выражении2.

Пример

```
for (i=0;i<10;i++)    //До  тех  пор,  пока  i  меньше  10,
{a[i]=i;}             //присваивать значение элементам массива.
```

Конструкция do-while:

Вначале выполняется блок операторов, и только потом проверяется выполнение условия.

```
do
{
// Выполнение блока операторов цикла
}
while (условное_выражение) ;
```

Организация бесконечных циклов

Для организации бесконечного цикла в качестве условного выражения в конструкции while или do-while можно просто указать значение true или 1:

while (1) блок__операторов;

или

for (; ;) блок__операторов;

Оператор break

Если в теле любого цикла встречается оператор break, то происходит выход из цикла.

6. Директивы препроцессора

Директивы препроцессора, по сути, не являются составной частью языка С, а используются для подстановки кода в текст программы. Препроцессор — это особая программа, которая выполняет предварительную обработку данных до начала компиляции.

Директива #include

Используется фактически во всех программах для включения заголовочных файлов (библиотек), имеющий расширение .h, или файлов .c

#include <имя_файла>

Директива #fuse delay

Директива fuse delay предназначена для задания рабочей частоты процессора и разрешает использование в программе внутренних функций delayms() (задержка в миллисекундах) и delay_us () (задержка в микросекундах).

#use delay(частота)

Директива #use rs232

Директива препроцессора #use rs232 используется для инициализации последовательного порта, работающего по стандарту RS232.

```
#use rs232 (опция, опция, опция, ...)
```

7. Обработка прерываний

Пример – определение обработчика для прерывания от АЦП:

```
#include   <avr/signal.h>
INTERRUPT(SIG_ADC) {
// Здесь вносится пользовательский код
}
```

или

```
#include <avr/signal.h>
SIGNAL(SIG_ADC) {
// Здесь вносится пользовательский код }
```

Если возникает неожиданное прерывание (то есть, такое, для которого не определен обработчик), по умолчанию происходит сброс микроконтроллера в результате перехода по соответствующему вектору. Во избежание этого, следует добавить оператор signal ():

```
#include   <avr/signal.h>
SIGNAL(vector_default)
{
// Здесь вносится пользовательский код
}
```

В качестве идентификаторов прерываний, передаваемых в качестве параметров макросам INTERRUPT и SIGNAL, могут служить следующие:

SIG_ADC — аналого-цифровое преобразование завершено;
SIG_EEPROM_READY — память EEPROM готова;
SIG_INTERRUPT0 - SIG_INTERRUPT7 — внешнее прерывание;
SIG_OUTPUT_COMPARE0, SIG_OUTPUT_COMPARE2 - прерывание от таймера T0 или T2 по совпадению с регистром сравнения;
SIG_OUTPUT_C0MPARE1A- SIG_OUTPUT_COMPARE3A— прерывание от таймера T1 или T3 по совпадению с регистром сравнения A;
SIG_OUTPUT_COMPARE1B - SIG_OUTPUT_COMPARE3B — прерывание от таймера T1 или T3 по совпадению с регистром сравнения B;
SIG_OVERFLOW0 - SIG_OVERFLOW3 — прерывание по переполнению счетчика;
SIG__UART_DATA — прерывание по опустошению регистра данных UART;
SIG_UART_RECV — прерывание по завершению приема UART;
SIG_UART_TRANS — прерывание по завершению передачи UART;
и др.