Текстовые интерфейсы для встраиваемых систем

http://www.embedded.com/

EMBEDDED SYSTEMS PROGRAMMING SEPTEMBER 1995, p.66.

Steve Drevik,

Text Interfaces for Embedded Systems

Система текстового меню представляет собой "средний" интерфейс, более сложный, но меньше и проще, чем графический интерфейс в стиле Windows. В данной статье показано, как разработать систему текстовых меню, используя в качестве примера банкомат.

Пользовательский интерфейс для встраиваемых систем имеет большое разнообразие стилей и сложностей. Интерфейс может быть также прост, как набор кнопок с подсветкой для обратной связи, или так же сложен, как графический интерфейс Windows. Уровень сложности, как правило, ограничивается количеством доступной памяти и пространства кода, а хороший графический интерфейс с интерактивной справкой, как правило, требует больше одного мегабайта пространства кода, что находится вне диапазона большинства встраиваемых систем.

В этой статье рассказывается о разработке системы с текстовым меню, которая обеспечивает баланс между функциональностью, расширяемостью и пространством кода. Мы рассмотрим основы такой системы. Я буду предлагать дополнительные функции, которые могут быть многоуровневыми в основной системе.

Система текстового меню предоставляет пользователю полноэкранное меню, содержащее несколько вариантов выбора. Эти варианты могут вести к другим меню или выполнять другие функции. Это такой же тип интерфейса, как предоставляемый системами электронных досок объявлений (BBS), большинством банкоматов (ATM) и так далее. Текстовые меню могут быть показаны на встроенных ЖК-экранах или через последовательный порт для устройств, совместимых с эмуляцией терминалов VT100 или ANSI.

Текстовые меню обеспечивают ряд преимуществ:

- Они дают высокий уровень контроля и управления доступом со стороны пользователя (при достаточном количестве меню и подсказок).
- Они удобны - подсказки могут быть относительно большими: от 20 до 40 символов, и в меню легче ориентироваться, чем в интерфейсе командной строки (например, DOS).
- Они получают широкое распространение на различных интерфейсных устройствах (ПК, портативные терминалы, встроенные дисплеи).

Возможно, самым большим преимуществом для программиста является то, что хорошо продуманная система меню может быть легко изменена и адаптирована при увеличении требований и функциональности.

Дизайн

Меню определяется как содержание какого-либо экрана, показываемого пользователю. Меню будет предоставлять пользователю два или более вариантов выбора:

```
/* 25 строк на страницу является разумным для
большинства типов терминалов, но программист может
выделить место для текста заголовков, колонтитулов */

#define MAX_SELECTIONS 24

< вставьте здесь определение типа для SELECTION >

typedef struct _menu
{
    int id;
    int num_selections;
    SELECTION selection[ MAX_SELECTIONS];
}MENU;
```

Мы добавляем для использования в дальнейшем код идентификации каждого меню. Эти идентификационные коды могут быть прикреплены в коде к меткам с описанием используя перечисление:

```
enum _menu_ids
{
    MAIN_MENU,
    SECONDARY_MENU,
    ....
};
```

Выбор является основным строительным блоком меню. Когда что-то выбрано, это может привести в другое меню или вызвать функцию действия, такую, как сохранение файла или порождение задачи управления. Как минимум, он состоит из строки с подсказкой или текстовой строки, и выбора действия:

```
typedef struct _selection
{
    char prompt[ MAX_PROMPT_LEN];
    int (*function)(int);
    int fn_arg;
}SELECTION;
```

Здесь мы предполагаем, что выбор действия будет иметь целочисленный аргумент (для дополнительной гибкости) и будет возвращать целое число (для возвращения возможных ошибок).

Структура меню в банкомате (показываемого после вставки карты и ввода ПИН-кода) может быть определена, как показано в Листинге 1 9.

По мере добавления новых типов счетов (финансовые рынки, МРК, кредитные карты), всё, что необходимо сделать программисту, это добавить соответствующие пункты меню, а также написать соответствующие обработчики для **do_deposit()**, **do_withdrawl()** и **show_balance**

(). Если функции банкомата необходимо расширить, чтобы начать видео вызов в банк при помощи встроенной камеры и микрофона, к основному меню может быть добавлен новый элемент и чтобы поддержать новую возможность, пишется новая функция обработки.

Меню управляется одной функцией:

```
void mdi() /* интерфейс управления меню */
{
    char c;
    MENU *curr_menu = find_menu( MAIN_MENU);
    int curr_selection = 0;

    /* выполняем инициализацию ввода/вывода,
    отрисовываем вводную страницу и так далее */

    init_mdi()

    /* отрисовка главного меню */

    draw_menu( curr_menu, curr_selection);

    while( TRUE)
    {
```

Следующая инструкция - это то, где **mdi()** будет находится основную часть своего времени ожидания. Если эта программа работает в режиме реального времени, эта функция должна иметь методы для возврата контроля над процессором для многозадачного ядра.

```
    c = readch();
    // returns special codes defined by us
    // for arrow keys, page_up, and so on.
    switch( c)
    {
    case UP_ARROW:
        if( curr_selection !=0)
        {
            curr_selection--;
            draw_menu( curr_menu, curr_selection);
        }
        break;

    case DOWN_ARROW:
        if( curr_selection < ( curr_menu->num_selections-1))
        {
            curr_selection++;
            draw_menu( curr_menu, curr_selection);
        }
        break;

    case ENTER:
        ( *curr_menu->selection[ curr_selection].function)
        ( curr_menu->selection[ curr_selection].fn_arg);
        break;
    }
```

```
}
```

Функция **draw_menu()** не только устанавливает выбор на экране, но и показывает, какой выбор в настоящее время активен, используя инверсный режим печати текста или указывающую стрелку.

Мерцание экрана, вызванное перерисовкой меню при изменении выбора с помощью клавиш, может раздражать. Функция, которая снимает выделение текущего варианта и подсвечивает новый (одна функция, вызываемая с аргументами **FORWARD** или **BACKWARD**), является визуально более привлекательной.

Редакторы

Те же структуры меню могут быть расширены, чтобы сделать простые редакторы.

В дополнение к функции действия, вариант меню редактора будет иметь обработчик кнопок, принимающий консольный ввод и сохраняющий его в память, и, при необходимости, функцию подсказки, чтобы получить текущие настройки в память и построить строку для показа пользователю:

```
typedef struct _selection
{
    char prompt[MAX_PROMPT_LEN];
    int (*function)(int); /* функция действия */
    int fn_arg;
    char * (*pro_function)(int); /* функция подсказки */
    int pro_fn_arg;
    int (*key_function)(int); /* обработка кнопок */
    int key_fn_arg;
}SELECTION;
```

Предположим, что наш банкомат теперь позволяет клиенту обновить водительские права, введя соответствующую информацию (имя, номер лицензии, адрес). Этот код показан в Листинге 2 10.

Аргументы функций обработки кнопок и подсказки, как правило, будут одинаковы, и, вероятно, их можно объединить, чтобы сэкономить в меню место. Функции **display()** и **keyedit()** сами по себе будут в основном большими операторами **switch()**, обрабатывающими аргумент функции подсказки и обработки кнопок.

```
typedef struct _selection
{
    char prompt[MAX_PROMPT_LEN];
    int (*function)(int); /* функция действия */
    int fn_arg;
    char * (*pro_function)(int); /* функция подсказки */
    int (*key_function)(int); /* обработка кнопок */
    int pro_key_fn_arg;
}SELECTION;

/* глобальные переменные */
char global_curr_name[80];
```

```
int global_curr_lnum;
char global_curr_address[80];
char workbuf[80];

// временный буфер ввода/вывода
char * prompt(int state)
{
    switch(state)
    {
    case NAME:
        strcpy(workbuf, global_curr_name);
        break;

    case LNUM:
        sprintf(workbuf, "%d", global_curr_lnum);
        break;

    case ADDRESS:
        strcpy(workbuf, global_curr_address);
    break;
    }
    return(workbuf);
}
```

Функция **draw_menu()** теперь обходит все варианты выбора, выводит строку подсказки (как и раньше), а также проверяет наличие функции подсказки. Если она существует, система вызовет функцию подсказки, возвращающую строку в рабочем буфере порта (**workbuf**). Эта строка затем будет показана в правильном месте экрана:

```
void draw_menu(MENU * menu)
{
    int selection_num;
    SELECTION * selptr;

    /* очистить экран */
    cls();

    /* обойти все варианты выбора в ДАННОМ меню */

    for(selection_num=0; selection_nummenu->num_selections; selection_num++)
    {
        /* для удобства создаём указатель */
        selptr = &menu->selection[selection_num];

        /* Если это "активный" элемент меню
        (определяем это с помощью ссылки на глобальную
        переменную 'active_selection'), отрисовать его
        инвертированным текстом. */

        if (selection_num == active_selection)
            inverse(ON);

        /* напечатать строку подсказки */
        setcur(selection_num + MENU_INDENT_Y, MENU_INDENT_X);
```

```
        printf(selptr->prompt);

        /* если этот вариант имеет функцию подсказки, используем её! */
        if(selptr->pro_function)
        {
            /* напечатать строку подсказки */
            setcur(selection_num + MENU_INDENT_Y, PROMPT_INDENT_X);
            printf((*selptr->pro_function)(selptr->pro_key_fn_arg));
        }

        /* если необходимо, выключаем печать инвертированным текстом */

        if (selection_num == active_selection)
            inverse(OFF);
    }
}
```

Функция **mdi()** переходит к вызову функции обработки ввода с кнопок только после завершения верного ввода, как это показано в Листинге 3[11].

Теперь, когда имеется основная система меню, каким образом программист справляется с дополнительными потребностями? Этот очень универсальный подход к меню хорошо подходит для дополнительных возможностей.

Защита паролем

Чтобы назначить уровень привилегий доступа, к каждой структуре **SELECTION** может быть добавлена целочисленная переменная. Текущий уровень доступа пользователя может быть проверен, когда пользователь пытается изменить или активировать этот вариант. Если уровень доступа пользователя слишком мал, может быть сгенерировано сообщение. Кроме того, можно изменить функцию **draw_menu()**, чтобы сравнить текущий уровень доступа пользователя с необходимым до отрисовки каждого варианта, и не отрисовывать вариант, если уровень доступа пользователя является слишком низким. Таким образом, пользователь с низким уровнем доступа даже не увидит вариантов для доступа более высокого уровня .

Переключатели управления функциональностью

В некоторых приложениях программист разрабатывает один основной вариант исполняемого кода для всех клиентов, но может захотеть получать оплату за некоторые дополнительные программные опции. Нежелательно создавать и тестировать копию программного обеспечения для каждой возможной комбинации (отключение или включение опций с использованием переключателей компиляции). Идеальным вариантом было бы создать и протестировать одну копию программного обеспечения для всех клиентов, а также включать или отключать доступ к элементам, которые не приобретены.

Если оборудование имеет энергонезависимую память, такую как небольшая EEPROM, может быть сохранено битовое поле с указанием, какие опции клиентом приобретены.

Когда клиент хочет добавить функцию, может быть отправлена новая EEPROM. Битовое поле в EEPROM сравнивается с битовым полем функции, добавленным в каждую структуру **SELECTION**, и если соответствующие биты установлены, вариант отображается:

```
typedef struct _selection
{
    .... // original structure
    int features_bitfield;
}SELECTION;

enum _feature_bits
{
    FEATURE_ATM_VIDEOPHONE=0,
    FEATURE_ATM_RENEW_LICENSE,
    FEATURE_ATM_ORDER_MOVIE
};

#define FEATURE_ALL 0
#define FEATURE_ATM_VIDEOPHONE_BIT (0x01 << FEATURE_ATM_VIDEOPHONE)
#define FEATURE_ATM_RENEW_LICENSE  (0x01 << FEATURE_ATM_RENEW_LICENSE)
#define FEATURE_ATM_ORDER_MOVIE    (0x01 << FEATURE_ATM_ORDER_MOVIE)

MENU menu[] =
{
    { MAIN_MENU, 6,
    {"Perform A Deposit",          goto_menu(), DEPOSIT,   FEATURE_ALL},
    {"Perform a Withdrawl",        goto_menu(), WITHDRAWL, FEATURE_ALL},
    {"Obtain a Balance Statement", goto_menu(), BALANCE,   FEATURE_ALL},
    {"Make A Video Phone Call",    goto_menu(), BALANCE,
FEATURE_ATM_VIDEOPHONE_BIT},
    {"Watch A Movie Right Here",   goto_menu(), BALANCE,
FEATURE_ATM_VIDEOPHONE_BIT},
    {"Quit", logout(), 0}
    },

    /* тут могут быть другие меню */

};
```

Наконец, как требование для отрисовки каждого выбора, в **draw_menu()** вставляется следующее условие:

```
if(!curr_menu->selection[selection_loop_ctr].feature_bitfield ||
    curr_menu->selection[selection_loop_ctr].feature_bitfield &
    eeprom.feature_bitfield)
{
    /* отрисовать выбранное */
}
```

Если программист использует подход с использованием функции **hilite()** для перемещения подсвеченного варианта, необходимо будет также посмотреть на переключатели вариантов, чтобы указать отображаемые подсказки для перехода в следующее и предыдущее меню.

Возвращаемые значения

В оригинальном дизайне все функции обработчиков кнопок и действий возвращают статус. В предыдущих примерах показано только два возможных значения, **OK** и **NOT_OK**. Было бы

лучше предоставлять пользователю более конкретную информацию. Программист может заранее определить коды ошибок и даже массив строк, соответствующих сообщениям об ошибках. Функции обработки нажатий могут возвращать эти коды ошибок, и функция **mdi()** с помощью таблицы может преобразовывать эти коды в соответствующие строки ошибок.

В предыдущих примерах были варианты меню для возврата в предыдущее меню или непосредственно в главном меню. В действительности, это пустая трата вариантов выбора и нелогично для пользователей приложений DOS, которые часто ожидают для возвращения в предыдущее меню нажатия кнопок ESC или F10. Оператор **switch()** в **mdi()**, безусловно, может найти такой код, но программа должна знать, в какое меню надо вернуться. Решение заключается в добавлении идентификационного кода родительского меню в каждую структуру **MENU**. Программист может назначить обратную ссылку для каждого пункта меню.

Часто также удобно иметь глобальную переменную для обозначения идентификатора родительского меню. Программисты могут привести примеры, когда в одно подменю можно добраться из трёх или четырёх меню. Когда пользователь нажимает ESC, программа должна вернуться в правильное меню. В этом случае родительское меню не известно во время компиляции. Так что при входе в такое подменю присваиваем переменной в памяти альтернативный идентификатор родительского меню. Обработчик ESC в mdi() сначала проверит, установлена ли переменная. Если установлена, он переходит к меню с этим ID и очищает переменную. Если она не установлена, он смотрит в структуру MENU активного меню и использует указанный в ней ID родительского меню.

Приведенные выше примеры редакторов предполагают, что ширина поля редактирования был так мала, что можно поместить строку ввода после строки подсказки. Если строки подсказок имеют 30 символов и мы позволяем иметь некоторое пространство до достижения области правки, мы ограничены от 40-ка до 50-ти символами в терминале с 80-ти символьной строкой. Это может быть недостаточным для некоторых приложений.

Одним из решений является указать ширину поля в структуре **SELECTION** для каждого варианта, содержащего строку подсказки и функцию обработчика нажатий (ноль для нередактируемых полей). Функция **draw_menu** могла бы теперь быть разработана для обработки дополнительных полей, помещая дополнительные данные на дополнительные строки. Программист должен быть осторожным, потому что теперь меню с 10-ю вариантами может занять 24 строки на странице, если многие варианты длиннее, чем одна строка. Другой программист мог бы добавить новые варианты, которые используют три строки, и теперь меню уже не умещается на экране, хотя оно содержит только 11 параметров.

Если поля для ввода более 40-ка символов являются скорее нормой, чем исключением, оформление меню, вероятно, следует пересмотреть с учётом этого.

Динамические меню

В большинстве приложений строки подсказок в меню и действия будут определяться программистом во время компиляции. Тем не менее, есть случаи, когда пользователь может захотеть меню, в которых подсказка является динамической. Используя пример нашего банкомата с продлением лицензии, мы могли бы запрашивать у пользователя имена ближайших родственников в одном меню и переходить к другому меню редактора, чтобы запросить пользователя, что является точным соотношением для каждого данного имени:

```
Barney Williams:      Father
Eleanor Williams:     Mother
Michael Tate:        Husband
Kelly Williams:       Sister
Robert Williams:     Brother
```

В этом случае строки подсказки являются динамическими. Так как функция **mdi()** делает всё через указатель на активное меню, можно динамически создавать структуру **MENU**, заполнять соответствующие поля (количество вариантов, подсказки, права доступа, функции обработки) и указать активным указателем меню на такой блок памяти. В этой ситуации особое внимание необходимо уделять поддержанию правильного ID родительского меню и обработке кнопки возврата (ESC). Программист мог бы создать стандарт, в котором нулевой код ID родительского меню указывает на динамическое меню, показывая, что **active_menu_ptr** является указателем на память, которая должна быть освобождена перед переходом к родительскому меню (но не освобождать, пока не прочитан ID родителей меню).

Структура меню

Одним из недостатков перечисленных реализации является то, что структура **MENU** будет занимать одинаковый объём памяти, не зависимо от того, имеет меню две подсказки или все 25. Чтобы сэкономить место, программист мог бы создать несколько типов **MENU**, которые отличаются только с точки зрения максимально возможного количества подсказок (**TINY_MENU**, **SMALL_MENU**, **MEDIUM_MENU**, **LARGE_MENU** и так далее). Программист мог бы затем поместить каждое меню в соответственно определённую переменную, в зависимости от количества доступных подсказок. Соответствующий код можно найти в Листинге 4[12].

Для системы, которая выполняется из PROM, как правило, желательно поместить переменные MENU[] в PROM, запрещая их копирование с последующей ссылкой в оперативную память. Это может быть достигнуто созданием одного файла на языке Си (**menus.c**), который ничего не делает, но определяет или подключает заголовочный файл, который определяет переменные меню. Компилируйте такой файл с директивами, создающими результирующий объектный файл с уникальным сегментом. Затем системе расположения кода может быть приказано не копировать автоматически этот сегмент данных в оперативную память.

Это реализация системы текстового меню предоставляет интерфейс, который является масштабируемым с точки зрения количества меню, а также сложности набора функций. Представление меню как массива вариантов выбора делает структуры меню простыми для изменения и код хорошо поддаётся повторному использованию. Большая часть кода в обработчике ввода и функций подсказок будет идентична, ссылаясь на другие переменные системы. Добавление нового обработчика - это быстрая операция копирования и вставки. В самом деле, менеджер по развитию, возможно, начнёт видеть больше ошибок, вызванных копированием и вставкой кода программистом и забыванием изменить исходный код для новых потребностей. Тем не менее, преимущества повторного использования обычно перевешивают эту неприятность, особенно, если все участники предупреждены о такой ловушке.

Листинг 1

Простое меню банкомата.

```
enum _menu_ids
{
    MAIN_MENU,
```

```
    DEPOSIT_MENU,
    WITHDRAWL_MENU,
    BALANCE_MENU
};

MENU menu[] =
{
    { MAIN_MENU, 4,
    {"Perform A Deposit",          goto_menu(), DEPOSIT},
    {"Perform a Withdrawl",        goto_menu(), WITHDRAWL},
    {"Obtain a Balance Statement", goto_menu(), BALANCE),
    {"Quit", logout(), 0}
    },

    { DEPOSIT_MENU, 3,
    {"Deposit to Checking", do_deposit(), CHECKING},
    {"Deposit to Savings",  do_deposit(), SAVINGS),
    {"Go Back",             goto_menu(),  MAIN_MENU}
    },

    { WITHDRAWL_MENU, 3,
    {"Withdrawl from Checking", do_withdrawl(), CHECKING),
    {"Withdrawl from Savings",  do_withdrawl(), SAVINGS),
    {"Go Back",                 goto_menu(),    MAIN_MENU}
    },

    { BALANCE_MENU, 3,
    {"Print Checking Balance", show_balance(), CHECKING},
    {"Print Savings Balance",  show_balance(), SAVINGS},
    {"Go Back",                goto_menu(),    MAIN_MENU}
    },

    /* тут могут быть другие меню */
};
```

Листинг 2

Меню обновления лицензии.

```
MENU menu[] =
{
    /* тут предыдущее меню */

    {RENEW_LICENSE_MENU, 3,
    {"Enter Name:",           NULL, 0, prompt, NAME,    keyedit, NAME},
    {"Enter License Number:", NULL, 0, prompt, LNUM,    keyedit, LNUM},
    {"Enter Address:",        NULL, 0, prompt, ADDRESS, keyedit, ADDRESS},
    {"Quit, Back to Main Menu", goto_menu(), MAIN_MENU, NULL, 0, NULL, 0}
    },
    /* тут могут быть другие меню */
};
```

Листинг 3

Расширенная версия **mdi()**.

```
void mdi() /* интерфейс управления меню */
{
    char c;
    MENU *curr_menu = find_menu( MAIN_MENU);
    int curr_selection = 0;
    int status;
    draw_menu( curr_menu, curr_selection);
    while( TRUE)
    {
        c = readch();
// возвращает специальные коды, установленные нами для кнопок со стрелками,
перехода по страницам и т. д.
        switch( c)
        {
        case UP_ARROW:
            if( curr_selection !=0)
            hilite( FORWARD);
            break;

        case DOWN_ARROW:
            if( curr_selection < ( curr_menu->num_selections-1))
                hilite( REVERSE);
            break;

        case ENTER:
            (*curr_menu->selection[ curr_selection]. function)
            ( curr_menu->selection[ curr_selection]. fn_arg);
            break;

        default:
            if( curr_menu->selection[ curr_selection]. key_handler != NULL)
            {
                if( get_input())
                /* 'ложь', если вышли по ESC или CTRL-C, иначе 'истина' */
                {
                    status = (*curr_menu->selection[ curr_selection].
key_handler)
                        ( curr_menu->selection[ curr_selection]. pro_key_fn_arg);
                }
                else
                {
                    redraw_prompt();
                }
            }
            /* Иначе игнорировать ввод полностью
            или обратная связь с указанием ошибки для пользователя */
        }

int keyedit( int state)
{
    int okay = TRUE;
```

```
    switch(state)
    {
    case NAME:
        if(validate_name(workbuf))
            strcpy(global_curr_name,workbuf);
        else
            okay = FALSE;
        break;
    case LNUM:
        if(validate_lnum(workbuf))
            global_curr_lnum = atoi(workbuf);
        else
            okay = FALSE;
        break;
    case ADDRESS:
        if(validate_address(workbuf))
            strcpy(global_curr_address,workbuf);
        else
            okay = FALSE;
        break;
    }
    return(okay);
}
```

Листинг 4

Расширенное меню управления.

```
typedef struct _medium_menu
{
    int id;
    int num_selections;
    SELECTION selection[12];
}MEDIUM_MENU;

typedef struct _small_menu
{
    int id;
    int num_selections;
    SELECTION selection[6];
}SMALL_MENU;

typedef struct _tiny_menu
{
    int id;
    int num_selections;
    SELECTION selection[3];
}TINY_MENU;

/* Это несколько усложняет функцию goto_menu, но не намного */
void goto_menu /* Установить и отобразить активное меню */
(
    int new_menu_id /* идентификатор нового меню */
```

```
)

{
    MENU * menu;
    int i,j;
    extern MENU * curr_menu;
    menu = findmenu( new_menu_id);
    if( menu == NULL)
    {
        /* ошибка системы */
    }
    else
        curr_menu = menu; /* здесь curr_menu является глобальной или внешней
переменной */
    draw_menu( );
    return;
}
////////////////////////////////////////
MENU * findmenu
(
    int new_menu_id /* идентификатор нового меню */
)
{
    MENU * ptr;
    int i,j;

    /** сначала попытаться найти меню в списке больших меню **/
    for( i = 0; i < MENU_END; i++)
    {
        if( new_menu_id == menu[ i]. menu_id)
        {
            ptr = ( MENU *) &menu[ i];
            break;
        }
    }

    /** если не нашли, пробуем найти меню в списке коротких меню **/
    if( new_menu_id >= MENU_END)
    {
        for( i=MENU_END+1,j=0; i<MEDIUM_MENU_END; i++, j++)
        {
            if( new_menu_id == medium_menu[ j]. menu_id)
            {
                ptr = ( MENU *) &medium_menu[ j];
                break;
            }
        }
        if( new_menu_id >= MEDIUM_MENU_END)
        {
            for( i = MEDIUM_MENU_END+1, j = 0; i < SHORT_MENU_END; i++, j++)
            {
                if( new_menu_id == short_menu[ j]. menu_id)
                {
                    ptr = ( MENU *) &short_menu[ j];
                    break;
```

```
                }
            }
            if( new_menu_id >= SHORT_MENU_END)
            {
                for( i=SHORT_MENU_END+1,j=0;i < TINY_MENU_END; i++, j++)
                {
                    if( new_menu_id == tiny_menu[ j]. menu_id)
                    {
                        ptr = ( MENU *) &tiny_menu[ j];
                        break;
                    }
                }

                /** ошибка, если в двух списках соответствия не найдено **/
                if( new_menu_id >= TINY_MENU_END)
                {
                    return( NULL);
                }
            }
        }
    }
    return( ptr);
}
```