

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение
высшего образования «Казанский национальный исследовательский
технический университет им. А.Н. Туполева-КАИ»

Институт компьютерных технологий и защиты информации

Кафедра систем информационной безопасности

Индекс по учебному плану): Б1.О.12.01

Специальность: 10.05.02 Информационная безопасность
телекоммуникационных систем

Специализация: "Разработка защищенных телекоммуникационных систем"

Методические указания к лабораторным работам
по дисциплине

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА

Казань 2021

Представление чисел. Преобразования типов.

Представление чисел

Код, когда каждая десятичная цифра кодируется четырьмя двоичными разрядами, называется двоично-десятичным. (BCD).

Двоично-десятичные числа, в свою очередь, могут быть представлены в одном из следующих форматов:

- в формате ASCII;
- неупакованном двоично-десятичном формате;
- упакованном двоично-десятичном формате.

Представление чисел в формате ASCII удобно в тех случаях, когда необходимо выполнять ввод чисел с консоли или вывод на какое-либо устройство, например дисплей или принтер. Для получения правильного результата при выполнении арифметических операций с такими числами необходимо проводить корректировку результата с помощью специальных команд.

ASCII-числа представлены следующим образом: старшие 4 разряда каждого байта содержат значение $3h$, а младшие 4 разряда — значение десятичной цифры.

Например, число 6591 в формате ASCII представлено как 36 35 39 31h, при этом самый старший байт содержит значение $36h$, а самый младший — $31h$.

Числа в неупакованном двоично-десятичном формате отличаются от их ASCII-представления тем, что старшие 4 разряда таких чисел установлены в 0. Они называются зоной. Таким образом, в одном байте могут быть представлены числа в диапазоне от 0 до 9.

Например, число 6591 в неупакованном формате выглядит как 06 05 09 01b, причем самый старший байт содержит число 06h, а самый младший — 01h.

Упакованный двоично-десятичный формат. Каждый байт упакованного числа содержит две десятичные цифры, то есть каждое десятичное число представлено четырьмя битами. Например, число 6591 в упакованном формате будет представлено как 6591h, причем старший байт содержит значение 65h, а младший — 91h. Таким образом, в одном байте могут быть представлены числа в диапазоне от 00 до 99.

Упакованные двоичные числа можно только складывать и вычитать, другие операции требуют дополнительного преобразования в неупакованный формат. В настоящее время упакованные двоично-десятичные числа находят ограниченное применение.

Двоично-десятичные числа описываются при помощи директив db и dt.

Обработка данных в форматах ASCII и BCD

Для получения высокой производительности компьютер выполняет арифметические операции над числами в двоичном формате. Во многих случаях новые данные вводятся программой с клавиатуры в виде ASCII-символов в десятичном формате. Аналогично, вывод информации на экран осуществляется в ASCII-кодах. В то же время для выполнения арифметических операций над числовыми величинами необходима специальная обработка. Процессор позволяет производить такую обработку с помощью специальных ассемблерных команд, предназначенных для выполнения арифметических операций непосредственно над числами в формате ASCII:

- AAA (ASCII Adjust for Addition) — коррекция для сложения ASCII-кода;
- AAS (ASCII Adjust for Subtraction) — коррекция для вычитания ASCII-кода.

- AAD (ASCII Adjust for Division) — коррекция для деления ASCII-кода;
- AAM (ASCII Adjust for Multiplication) — коррекция для умножения ASCII-кода;

Эти команды кодируются без операндов и выполняют автоматическую коррекцию содержимого регистра AX. Коррекция необходима, так как ASCII-код представляет собой так называемый неупакованный десятичный формат, в то время как компьютер выполняет арифметические операции в двоичном формате.

Команда AAA.

AAA анализирует значение младшей тетрады регистра al.

Если ее значение меньше 9, то флаг CF сбрасывается в 0.

Иначе

1. К содержимому младшей тетрады прибавляется 6.
2. Флаг CF устанавливается в 1 – это означает перенос в старший разряд.

```
.data
num1      DB      38h
num2      DB      34h
.code
        xor      AX,      AX
        mov      AL,      num1
        mov      BL,      num2
        add      AL,      BL
        aaa
```

Из примера видно, что регистр AL содержит значение 38h, а регистр BL — 34h. Числа 38h и 34h представляют собой два байта в формате ASCII. После выполнения сложения (команда ***add*** AL, BL) регистр AL будет содержать значение 6Ch. После выполнения коррекции (команда ***aaa***)

регистр AX будет содержать неупакованное двоично-десятичное число 0102h.

Команда *aaa* проверяет правый полубайт в регистре AL. Если его значение находится между A и F или флаг AF равен 1, то к регистру AL прибавляется 6, к регистру AH прибавляется 1, а флаги AF и CF устанавливаются в 1. Кроме того, команда *aaa* устанавливает в 0 левый полубайт в регистре AL. Для того чтобы получить символьное представление ASCII-числа, необходимо выполнить операцию поразрядного ИЛИ над левым полубайтом результата с помощью команды *or*. Для регистра AX эта операция будет выглядеть так:

or AX, 3030H

Сложение многобайтовых ASCII-чисел требует организации цикла, который выполняет обработку справа налево с учетом переноса.

Обработка упакованных двоично-десятичных целых чисел.

Для корректировки результата сложения и вычитания упакованных двоично-десятичных целых чисел предназначены две команды:

- *DAA (Decimal Adjust after Addition)*
- *DAS (Decimal Adjust after Subtraction).*

Команда DAA

Команда *DAA* преобразовывает двоичное число, находящееся в регистре AL и полученное в результате выполнения команд *ADD* или *ADC* в упакованный десятичный формат.

Пример. Сложить два упакованных десятичных числа 35 и 48.

Младшая цифра результата (*7Dh*) больше 9, поэтому выполняется коррекция результата. Коррекция старшей цифры результата, равной 8, не выполняется:

mov al,35h

add al,48h ; AL = 7Dh

daa ; AL = 83h (после коррекции)

Алгоритм работы команды *DAA*:

Если младшая тетрада регистра AL > 9 или флаг AF = 1, то

$AL = AL + 6;$

$CF = CF$ или *Перенос_Последнего_Сложения*;

$AF = 1;$

иначе

$AF = 0;$

Если старшая тетрада регистра $AL > 9$ или $CF = 1$, то

$AL = AL + 60H;$

$CF = 1;$

иначе

$CF = 0;$

Команда DAS

Команда *DAS* преобразовывает двоичное число, находящееся в регистре *AL* и полученное в результате выполнения команд *SUB* или *SBB* в упакованный десятичный формат.

Пример. Из упакованного десятичного числа 85 вычитается число 48 и выполняется коррекция полученного результата:

Mov bl, 48h

Mov al, 85h

Sub al, bl ; AL = 3Dh

Das ; AL = 37h (после коррекции)

Алгоритм работы команды *DAS*:

Если младшая тетрада регистра $AL > 9$ или $AF = 1$, то

$AL = AL - 6;$

$CF = CF$ или *Заем_Последнего_Вычитания*;

$AF = 1;$

иначе

$AF = 0;$

Если старшая тетрада регистра $AL > 9FH$) или $CF = 1$, то

$AL = AL - 60H$;

$CF = 1$;

иначе

$CF = 0$;

Задания

1. Даны два упакованных десятичных числа.
 - a. Выполнить сложение этих чисел.
 - b. Выполнить вычитание этих чисел.
2. Даны два числа в формате ASCII .
 - a. Выполнить сложение этих чисел.
 - b. Выполнить вычитание этих чисел.
2. Даны два числа в формате BCD .
 - a. Выполнить сложение этих чисел.
 - b. Выполнить вычитание этих чисел.

Лабораторная работа №2

Структуры. Объединения. Записи.

Структуры

Структура – это тип данных, состоящий из фиксированного числа элементов разных типов.

Для использования структур необходимо:

- Описать структуру (Задать шаблон структуры), т.е. создать новый тип данных.
- Определить экземпляр структуры, т.е. объявить переменную типа структура и проинициализировать ее.

Описание может быть только одно, а определений переменных описанного типа может быть много.

Описание структуры

Описать структуру можно при помощи директивы `STRUC`.

имя_структуры `STRUC`

<Описание полей>

имя_структуры ENDS

Описания полей это список из директив описания данных. При помощи этих директив можно определить размер полей и, в случае необходимости, начальные значения.

Описание структуры должно появиться в программе до того, как будет объявлена переменная с типом данной структуры. Таким образом, описание структуры надо размещать в начале сегмента данных. (либо перед описаниями сегментов).

Пример 1:

Объявить структуру, содержащую следующие сведения:

Фамилия – 10 символов

Номер группы - число

Адрес- 20 символов

```
stud struc
    nam db 15 dup (0)
    n    dw 0
    adr db 20 (0)
stud ends
```

Пример 2.

Описать структуру, содержащую информацию об адресе, и проинициализировать поля этой структуры.

```
adres struc
    dom db 16
    qu  db 31
    ru  db 'Baumana $'
    gor db 'Kazan $'
adres ends
```


Определение данных типа структура

[имя_переменной] имя_структуры <[список значений]>

Имя_переменной – необязательное поле. Если имя переменной опустить, будет выделена область памяти под структуру.

Список значений – необязательное поле.

Если список не указывать, то поля НЕ будут проинициализированы даже значениями по умолчанию, заданными в шаблоне структуры.

Если список есть, то он заключается в угловые скобки.

Если угловые скобки оставить пустыми, то все поля структуры будут проинициализированы значениями по умолчанию, заданными в шаблоне.

Можно проинициализировать отдельные поля переменной. При этом, пропущенные поля должны отделяться запятыми. Эти поля будут проинициализированы значениями по умолчанию, заданными в шаблоне структуры.

Пример.

```
x    adres
y    adres <>
z    adres <52, , "Mira $">
```

Примечание:

- если строка не содержит пробелов, то она может заключаться в одиночные кавычки. Если строка содержит пробелы, то кавычки должны быть двойными.
- Значения, которыми инициализируется поля переменной, не должны быть длиннее значений, которые по умолчанию присваиваются полям структуры. Иначе будет выдано сообщение об ошибке.

Обращение к полям структуры

Для доступа используется символ «точка».

Адресное_выражение.имя_поля

Пример:

```
.model tiny
.DATA
stud struc
    a    db    0
    n    db    0
stud ends
x        stud
.code
start:
    push cs
    pop ds
    push ds
    mov x.a,20
    mov x.n,100
    mov ax,4c00h
    int 21h
end start
```

Объединения

Объединение – это разновидность структуры, все поля которой размещаются начиная с одного и того же адреса. Размер объединения равен размеру самого длинного поля. Синтаксис описания объединения:

```
имя_структуры UNION
<Описание полей>
имя_структуры ENDS
```

Элементами структур могут быть объединения и наоборот.

Пример:

```
.model tiny
u union
```

```

        x      db      3
        y      db      ?
u ends

a struc
        s      db      5
        t      u      <>

```

```

a ends

```

```

w union
        m      db      4
        k      a      ?

```

```

w      ends

```

```

.data

```

```

p1      a      <>
p2      a      <10,5>
p3      w      <6>

```

```

.code

```

```

n:      push cs
        pop ds
        mov  p1.t.y,'A'
        mov  p2.t.x,'S'
        mov  p3.k.t.y,15

```

```

end     n

```

Записи

Запись – это разновидность структуры. Память под поля структуры выделяется побитно. Размер записи равен сумме размеров ее полей и не может превышать 8, 16 или 32 бита.

Синтаксис описания записи:

Имя_записи RECORD <описание элементов>

Элементы описываются следующим образом:

Имя_поля: размер поля в битах [= значение];

Пример:

```
ru record a:3=5, b:1=0, c:4=10
```

Примечание:

- Поля записываются в одну строчку через запятую.
- Если сумма длин полей больше 8 или 16, то под запись выделяется соответственно 16 или 32 разряда.
- Если сумма длин полей меньше 8, 16 или 32, то поля записи прижимаются к младшим разрядам.
- Если инициализирующие значения не помещаются в отведенные поле, то отбрасываются старшие разряды.

```
ru record a:3=5, b:1=1, c:4=12
```

bc содержимое ячейки памяти (1 байт)

```
ru record a:1=5, b:1=0, c:7=10
```

0a 01 расположение в памяти: младший байт, старший байт.

Объявление переменной типа запись

Синтаксис объявления переменной:

Имя_переменной имя_записи значения_полей

Пример:

```
ru record a:2=2, b:3=5, c:3=7
```

1. Не требуется инициализация полей.

```
x ru ?
```

поля будут проинициализированы нулями

```
?? x1 ru ?
```

2. Поля инициализируются значениями по умолчанию.

```
y ru <>
```

```
AF y ru <>
```

3. Переопределение полей

z ru <6,3,11>

Поля переменной z примут значения

9B z ru <6,3,11>

4. Переопределение отдельных полей. Так же, как в случае инициализации структур, можно пропускать значения отдельных полей, но запятые должны оставаться на своих местах. Пропущенные поля примут значения по умолчанию. Список значений заключается в угловые скобки.

s ru <1,4>

6C s ru <1,4>

4. Выборочное переопределение полей. Список значений заключается в фигурные скобки. В скобках указываются только имена и значения переопределяемых полей. Остальные поля примут значения по умолчанию.

t ru {b=6}

B7 t ru {b=6}

Доступ к полям записи

Для организации доступа к полям записи применяются специальные операторы:

Width имя элемента записи - длина поля записи в битах

Width Имя экземпляра записи или **Width** имя типа записи – длина всей записи в битах

Пример:

shr al, width p4+width p3

Mask имя поля - локализует биты элемента записи.

Пример:

and al, mask p3

Для выделения элемента записи надо:

- Поместить запись в регистр

- При помощи маски **Mask** и оператора **and** локализовать биты в регистре.
- Сдвинуть биты элемента к младшим разрядам регистра с помощью оператора **shr**. Количество разрядов можно получить при помощи имени поля записи.

Чтобы поместить поле в запись, надо:

- При помощи команды **shl** сдвинуть поле на число разрядов, соответствующее имени поля.
- Обрезать лишние разряды при помощи команды **and** и маски
- Обнулить разряды поля в записи при помощи команды **and** и инвертированной маски
- С помощью команды **or** наложить новое значение поля на запись

Пример.

Дана запись. Увеличить значение поля p3 на 1.

```
.model tiny
```

```
.DATA
```

```
tr record p1:3=7,p2:1=0,p3:2=3,p4:1=1
```

```
r tr {p3=2}
```

```
.code
```

```
n:  push      cs
```

```
    pop  ds
```

```
    mov  al,r
```

```
    and  al, mask p3
```

```
    shr  al, width p4
```

```
    inc  AL
```

```
    shl  al,width p4
```

```
    and  al,mask p3
```

```

and    r,not mask p3
or     r,AL
mov    ax,4c00h
int    21h
end n

```

Задания

1. Объявить и проинициализировать структуру, содержащую следующие сведения:

Название детали – 10 символов

Цена - число

Количество - 20 символов

Создать переменную созданного типа и проинициализировать поле названия. Вывести на экран название детали.

2. Объявить и проинициализировать запись, содержащую следующие сведения:

Название книги – 10 символов,

Автор - 20 символов,

Количество страниц - число

Создать переменную созданного типа и проинициализировать поле количества страниц.. Увеличить в два раза количество страниц.

Лабораторная работа №3

Процедуры. Библиотеки

Директивы определения данных MASM

Директива	Название	Размер памяти	Значение
BYTE	байт	8 битов	8-разрядным целочисленное выражение или символьная константа

SBYTE	Байт со знаком	8 битов	8-разрядным целочисленное выражение или символьная константа
WORD	слово	16 битов	16-разрядным целочисленное выражение
SWORD	слово со знаком	16 битов	16-разрядным целочисленное выражение
DWORD	двойное слово	32 бита	32-разрядное целое значение
SDWORD	двойное слово со знаком	32 бита	32-разрядное целое значение
QWORD	учетверенное слово	64 бита	64-разрядное целое значение
TBYTE		10 байтов	80-разрядное целое значение, используется для хранения десятичных упакованных целых чисел (двоично-кодированных целых чисел).

Процедуры

Передача параметров в процедуру

Механизмы передачи параметров:

- ***По значению.*** Процедура использует копию значения параметра. Способ применяется при передаче параметров небольших размеров (байт или слово). Значение параметра копируется в регистр (или в стек).
- ***По ссылке.*** В процедуру передается адрес параметра. Способ применяется при передаче больших массивов данных и в тех случаях, когда процедура должна модифицировать данные. Адрес параметра может передаваться через регистр или через стек.
- ***По возвращаемому значению.*** Процедуре передается адрес параметра, процедура создает его копию (в регистре) и работает с ней, а перед возвращением в вызывающую программу записывает по адресу параметра возвращаемое значение.

- **По результату.** Отличается от передачи по возвращаемому значению тем, что переданный адрес используется для записи результата.
- **По имени.** Передача параметра осуществляется с помощью макроопределений.

Например:

```
Name      macro par
            Mov ax,par
```

```
Endm
```

```
Name 30
```

```
Call procname      ; procname – имя процедуры
```

Такой механизм применяется

- при передаче параметров в языках высокого уровня – функция получает адрес специальной программы, которая вычисляет адрес передаваемого по имени параметра.
- Директивой EQU
- Препроцессором при обработке директивы #define.

Передача параметров выполняется

- В регистрах.
- В глобальных переменных. Передача параметров в глобальных переменных считается неэффективной, так как при этом невозможна рекурсия и повторная входимость.
- В стеке.
- В потоке кода.

При этом данные, передаваемые в процедуру помещаются в прямо в тексте программы после команды Call.

- В блоке параметров.

При этом в процедуру передается адрес области памяти, содержащей параметры.

Передача через общую область памяти.

При этом применяются сегменты типа `common`. Необходимо, чтобы общие сегменты памяти имели одинаковые имена. Такой способ применяется при передаче параметров в процедуру, содержащуюся в другом модуле.

Команда ENTER– вход в процедуру

Синтаксис:

ENTER размер, уровень вложенности

Команда `Enter` создает стековый кадр заданного размера и уровня вложенности.

Операнды – числа:

- Размер – размер стекового кадра в килобайтах.
- Уровень вложенности может принимать значения от 0 до 31.

Эта команда используется для вызова процедуры, которая использует динамическое распределение памяти в стеке для своих локальных переменных.

Она

- помещает в стек указатель на стековый кадр текущей процедуры и
- той процедуры, из которой вызывалась текущая,
- создает стековый кадр нужного размера для вызываемой процедуры и
- помещает в `EBP` адрес начала кадра.

Действие:

`Push bp`

`Sub Sp,<n> ;`

Команда LEAVE– выход из процедуры

Выполняет действия, противоположные `Enter`.

Синтаксис:

LEAVE

- LEAVE копирует содержимое EBP в ESP, при этом из стека удаляется весь кадр, созданный последней выполненной командой ENTER,
- считывает из стека EBP для предыдущей процедуры, при этом ESP восстанавливает значение, которое он имел до вызова последней команды ENTER.

Действие:

Mov sp, bp

Pop bp

Пример.

`prim1 proc`

`enter 8,1 ; это соответствует выполнению следующих команд`

`;push ebp`

`;mov ebp,esp`

`;sub esp,8`

`mov eax,[ebp+8]`

`mov ebx,[ebp+12]`

`leave ; это соответствует выполнению следующих команд`

`;mov esp,ebp`

`;pop ebp`

`ret 8`

`prim1 endp`

Задания

1. Дан массив из 10 значений типа Byte. Написать процедуру для нахождения суммы элементов массива. Параметры передавать в стеке. Вызов процедуры выполнить при помощи команды ENTER, а возврат в вызывающую программу при помощи команды LEAVE. Результат вернуть в аккумуляторе.

2. Дан массив из 5 значений типа SWORD. Написать процедуру для нахождения количества положительных элементов массива. Параметры передавать в стеке. Вызов процедуры выполнить при помощи команды ENTER, а возврат в вызывающую программу при помощи команды LEAVE. Результат вернуть в аккумуляторе.
3. Дан массив из 3 значений типа SBYTE. Написать процедуру для нахождения количества отрицательных элементов массива. Параметры передавать через регистры. Вызов процедуры выполнить при помощи команды ENTER, а возврат в вызывающую программу при помощи команды LEAVE. Вернуть результат по возвращаемому значению.

Лабораторная работа №4

Программирование в среде Windows. Динамически загружаемые библиотеки. Функции API.

ПРОГРАММИРОВАНИЕ В СРЕДЕ WINDOWS

Операционные системы MS-Dos и Windows имеют разные идеологии программирования.

Программа Dos работает по определенному алгоритму. Такая программа при необходимости обслуживания со стороны операционной системы сама выполняет запрос.

Логика работы программы под управлением WINDOWS:

- Программа пассивна. Она реагирует на сообщения операционной системы.
- Сообщения операционной системы приходят в случайные моменты времени. (похоже на прерывания).

При составлении программ по Windows надо соблюдать следующие соглашения:

- Программы выполняются в защищенном режиме. Поэтому мы должны использовать директиву .386, которая разрешает использование набора команд для этого процессора.
- Каждая программа имеет в своем распоряжении 4-х гигабайта оперативной памяти. (Windows запускает каждую программу в отдельном виртуальном пространстве). Это значит, что программа может обращаться по любому адресу в этих пределах.
- Есть только одна модель памяти - FLAT. Можно использовать любой сегментный регистр для адресации к любой ячейке памяти. Не надо загружать сегментные регистры. Они сразу установлены в нужные значения. При адресации все регистры могут рассматриваться как адресные.
- При вызове процедур применяется соглашения, определяемые как *Stdcall*.
- Можно применять как стандартные, так и упрощенные директивы определения сегментов.

Windows использует esi, edi, ebp и ebx внутренне и не ожидает, что значение в этих регистрах меняются. Если вы используете какой-либо из этих четырёх регистров в вызываемой функции, не забудьте восстановить их перед возвращением управления Windows. Вызываемая (callback) функция - это функция, которая вызывается Windows. Очевидный пример - процедура окна. Это не значит, что вы не можете использовать эти четыре регистра. Просто не забудьте восстановить их значения перед передачей управления Windows.

Программирование в Windows основано на использовании функций API.

Функции API

API (Application Programming Interface) - это интерфейс - интерфейс прикладных программ.

Если разрабатываемое вами приложение имеет функцию, позволяющую обращаться к нему из других приложений, то это - API вашего приложения. Параметры, которые принимает ваша функция, образуют её API, так как они являются средством, при помощи которого другие приложения взаимодействуют с данной функцией.

Операционная система Windows предоставляет большой набор функций, позволяющих различным приложениям обмениваться информацией с Windows, иначе говоря, функции API позволяют организовать интерфейс между прикладной программой и средой, в которой работает эта программа. Вызов функций API позволяет программе получать доступ к ресурсам среды и управлять ее работой.

Эти функции называют Windows API. Любое из приложений, работающее в среде Windows, прямо или косвенно вызывает функции, входящие в Win32 API. Win32 API — 32-разрядный API для современных версий Windows.

Функции Windows API скомпонованы в динамически связанные библиотеки.

Динамически загружаемые библиотеки (Dynamic-link libraries)

Основная разница между статическими и динамическими библиотеками заключается в следующем. Если используется статическая библиотека, то на стадии редактирования связей в состав исполняемого модуля включаются все функции, для которых обнаружено обращение из текста исходной программы. В отличие от этого вызов модуля из динамической библиотеки происходит только на стадии выполнения

программы. При таком подходе библиотечные функции не включаются в состав исполняемого модуля, его размеры становятся меньше и, тем самым, экономится место, занимаемое исполняемыми файлами на диске.

При использовании динамически загружаемых библиотек для работы исполняемого модуля требуется присутствие в оперативной памяти динамической библиотеки, из которой в случае необходимости потребуется запустить тот или иной модуль. Поэтому предусматривается режим компиляции с включением всех вызываемых функций в состав исполняемого модуля.

Одно из достоинств DLL состоит в том, что, сколько бы приложений (процессов) не работало с функциями одной и той же DLL, код DLL существует в единственном экземпляре.

Динамические подключаемые библиотеки содержат общедоступные процедуры. Динамически-загружаемые библиотеки хранятся в файлах с расширением dll. (а также exe или drv).

Механизм DLL-библиотек появился вместе с операционной системой Windows и является ее неотъемлемой частью. Суть этого механизма в том, что в процессе компоновки исполняемого модуля с использованием внешних процедур в него помещаются не сами процедуры, а только их названия (номера) вместе с названиями DLL-библиотек, в которых они содержится.

Три основные библиотеки это:

Kernel32.dll. Эта библиотека предназначена для работы с объектами ядра операционной системы и ее функции позволяют управлять памятью и другими системными ресурсами, .

User32.dll. Здесь сосредоточены функции для управления окнами - основным видом объектов операционной системы. Функции этой DLL выполняют следующие действия:

- Обработка сообщений,

- Работа с меню,
- Работа с таймерами,

GDI32.dll. Эта библиотека, обеспечивающая графический интерфейс операционной системы (Graphics Device Interface). В состав этой библиотеки входят

- Функции управления выводом на экран дисплея,
- Функции управления выводом принтера,
- функции для работы со шрифтами.

API функции, используемые в приложении, должны быть объявлены. Операционная система Windows поддерживает две системы кодировки символов ANSI (однобайтовая) и Unicode (двухбайтовая). Поэтому существует две набора API функций: одна для ANSI и другая для Unicode. На конце имен API функций для ANSI стоит "A". В конце имен функций для Unicode находится "W".

Большинство прототипов для API-функций содержатся в include-файлах. Файлы подключения имеют расширение .inc и прототипы функций DLL находятся в .inc файле с таким же именем, как и у этой DLL. Например, ExitProcess экспортируется kernel32.lib, так что прототип ExitProcess находится в kernel32.inc. Если использовать include-файл, то можно обращаться к именам API функций без постфикса. Include-файлы могут определить и выбрать подходящую для вашей платформы функцию

Пример. Программа, которая ничего не делает.

```
386
.model flat, stdcall
option casemap:none

include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
```



```
.code
start:
    invoke ExitProcess, 0
end start
```

Прототип функции *ExitProcess* :

```
ExitProcess proto uExitCode:DWORD
```

uExitCode - это значение, которое программа вернет операционной системе после завершения.

В нашем примере возвращается 0.

Задания

1. Даны два значения типа DWORD Написать программу для WINDOWS для сложения этих чисел
2. Даны X и Y : два значения типа SBYTE. Написать программу для WINDOWS для вычитания Y из X
3. Даны X и Y : два значения типа SBYTE. Написать программу для WINDOWS для сравнения Y и X
4. Дано X значение типа SWORD. Написать программу для WINDOWS для вычитания инвертирования разрядов X
5. Дано X значение типа SWORD. Написать программу для WINDOWS для вычитания обнуления 3 6 разрядов X

Лабораторная работа №5

Оконные функции. Каркасное приложение.

Оконные приложения

Эти приложения имеют типовую структуру. Основа этой структуры называется каркасным приложением. Это приложение содержит

минимальный программный код, необходимый для работы Windows-приложения.

Состав Windows-приложения:

- Главная функция
- Цикл обработки сообщений
- Оконная функция

Главная функция содержит код для инициализации приложения в среде Windows . В результате выполнения главной функции на экране появляется окно.

Действия главной функции:

- Регистрация класса окна.
- Создание окна.
- Отображение окна.
- Запуск цикла обработки сообщений.
- Завершение выполнения приложения.

Цикл обработки сообщений предназначен для взаимодействия с внешним миром при помощи сообщений. Он организует обработку сообщений в оконной процедуре.

Оконные функции предназначены для обработки поступающих сообщений. Оконные функции вызываются из операционной системы, а не из приложения, в котором содержатся

Венгерская нотация

Имена идентификаторов предваряются префиксами, состоящими из одного или нескольких символов.

Префиксы, задающие тип

Префикс	Сокращение от	Смысл	Пример
s	string	строка	sClientName

sz	zero-terminated string	строка, ограниченная нулевым символом	szClientName
n, i	int	целочисленная переменная	nSize, iSize
l	long	длинное целое	lAmount
b	boolean	булева переменная	bIsEmpty
a	array	массив	aDimensions
t, dt	time, datetime	время, дата и время	tDelivery, dtDelivery
p	pointer	указатель	pBox
lp	long pointer	двойной (дальний) указатель	lpBox
r	reference	ссылка	rBoxes
h	handle	дескриптор	hWindow
m_	member	переменная-член	m_sAddress
g_	global	глобальная переменная	g_nSpeed
C	class	класс	CString
T	type	тип	TObject
I	interface	интерфейс	IDispatch
v	void	отсутствие типа	vReserved

Префиксы, задающие смысл

Префикс	Сокращение от	Смысл	Пример
i	index	Индекс	int ix; Array[ix] = 10;
d	delta	Разница между значениями	int a, b; ... dc = b - a;
n	number	Количество	size_t nFound = 0;

Если имя функции заканчивается суффиксом A, то приложение требует, чтобы система посылала сообщения с текстовыми параметрами набора ANSI.

Если имя функции заканчивается суффиксом W, то приложение требует, чтобы система посылала сообщения с текстовыми параметрами набора Unicode.

Дескриптор процесса

Операционная система Windows является многозадачной. Это значит, что одновременно могут выполняться несколько процессов. (неточно: несколько потоков в рамках одного процесса и несколько процессов одновременно.)

Процесс – это программа, которая загружена для выполнения. Ей выделяется область размером 4 Гб. В эту область могут быть загружены исполняемые файлы и динамически загружаемые библиотеки (*dll*-библиотеки) .

Для того чтобы можно было различать эти исполняемые файлы или библиотеки, им при загрузке присваиваются уникальные номера (описатели экземпляра) – дескрипторы.

Функция `GetModuleHandle` извлекает дескриптор указанного модуля, если файл был отображен в адресном пространстве вызывающего процесса.

Синтаксис

```
HMODULE GetModuleHandle( LPCTSTR lpModuleName);
```

Параметры

`lpModuleName` – Указатель на символьную строку с нулем в конце, которая содержит имя модуля (или `.dll` или `.exe` файл). Если расширение имени файла опускается, в конец добавляется заданное по умолчанию библиотечное расширение `.dll`. Символьная строка имени файла может включать в себя конечный символ точки (`.`), который указывает, что имя модуля не имеет расширения. Строка не должна определять путь. Когда

определяется путь, убедитесь, что используются обратные слэши (\), а не прямые слэши (/). Имя сравнивается (независимо от ситуации) с именами модулей в текущий момент отображаемыми в адресном пространстве вызывающего процесса.

Если этот параметр - NULL, GetModuleHandle возвращает дескриптор файла, используемый, чтобы создать вызывающий процесс (.exe файл).

Возвращаемые значения

Если функция завершается успешно, возвращаемое значение - дескриптор указанного модуля.

Если функция завершается ошибкой, возвращаемое значение - NULL. Чтобы получить дополнительную информацию об ошибке, вызовите

Замечания

Окно

Окно представляет собой объект Windows, управляемый с помощью дескриптора. Окна можно создавать, перемещать, уничтожать и т.д. с помощью специального набора функций Windows.

Окно – это объект. Объекты в среде Windows однозначно определяются с помощью дескрипторов.

Дескриптор (handle) представляет собой 16-разрядное слово. Дескриптор можно рассматривать, как индекс (номер) объекта в некоторой системной таблице объектов. Он всегда передаётся первым параметром в функцию Windows, работающую с этим объектом.

Регистрация оконного класса

Окно в Windows создается на основе оконного класса. Оконный класс представляет собой структуру с основными свойствами всех окон этого класса, например:

- форма курсора в рабочей области окна,
- необходимость автоматического обновления окна при изменении его размеров,

- адрес процедуры обработки сообщений.

В Windows существует несколько стандартных классов окон с заранее заданными свойствами. Однако, каждое приложение, как правило, регистрирует свой собственный оконный класс с тем, чтобы можно было управлять всеми свойствами создаваемого окна. Для создания оконного класса требуется объявить структуру `WndClass`, заполнить ее поля соответствующими значениями и зарегистрировать ее в Windows.

`WNDCLASS STRUC`

<code>CLSSTYLE</code>	<code>DD ? ;</code>	стиль окна
<code>CLWNDPROC</code>	<code>DD ? ;</code>	указатель на процедуру окна
<code>CLSCBCLSEX</code>	<code>DD ? ;</code>	информация о доп. байтах для
данной структуры		
<code>CLSCBWINDEX</code>	<code>DD ? ;</code>	информация о доп. байтах для окна
<code>CLSHINST</code>	<code>DD ? ;</code>	дескриптор приложения
<code>CLSHICON</code>	<code>DD ? ;</code>	идентификатор иконы окна
<code>CLSHCURSOR</code>	<code>DD ? ;</code>	идентификатор курсора окна
<code>CLBKGROUND</code>	<code>DD ? ;</code>	идентификатор кисти окна
<code>CLMENNAME</code>	<code>DD ? ;</code>	имя-идентификатор меню
<code>CLNAME</code>	<code>DD ? ;</code>	специфицирует имя класса окон

`WNDCLASS ENDS`

Поля структуры:

CLSSTYLE -битовые флаги, задающие начальные свойства всех окон, создаваемых на основе оконного класса.

CLWNDPROC - указатель на CALLBACK-функцию, вызываемую для обработки сообщений, адресованных окнам этого класса. CALLBACK-функция отличается от обычной тем, что вызывается извне прикладной программы (в данном случае самой системой Windows).

CLSCBCLSEX - дополнительное количество байт, выделяемой за структурой оконного класса (по умолчанию = 0).

CLSCBWINDEX - дополнительное количество байт, выделяемое для экземпляра окна (по умолчанию эта память =0).

CLSHINST - дескриптор экземпляра приложения, регистрирующего оконный класс.

CLSHICON - дескриптор пиктограммы, отображаемой на экране при минимизации окна

CLSHCURSOR - дескриптор курсора, который рисуется в окне при отображении окна на экране.

CLBKGROUND - дескриптор кисти, которая используется для закрашивания фона окна.

CLMENNAME - идентификатор меню, загружаемого из ресурса прикладной программы. Это меню используется для окна по умолчанию, если при его создании CreateWindow не указано другое меню.

CLNAME - строковый идентификатор оконного класса в Windows.

Когда все поля структуры WNDCLASS заполнены соответствующими значениями, производится регистрация оконного класса с помощью функции RegisterClass.

Функция находится в файле user32.dll

Функция RegisterClass регистрирует класс окна для последующего использования при вызове функции CreateWindow или CreateWindowEx. Класс окна может регистрироваться только один раз.

Функция RegisterClass была заменена функцией RegisterClassEx.

Синтаксис

ATOM RegisterClass(CONST WNDCLASS* lpWndClass);

Параметры

lpWndClass - Указатель на структуру WNDCLASS. Вы должны заполнить структуру соответствующими атрибутами класса перед передачей её в функцию.

Возвращаемое значение

Если функция завершается успешно, возвращаемое значение уникально идентифицирует зарегистрированный класс. Это значение может использоваться только функциями CreateWindow, GetClassInfo, FindWindow и UnregisterClass.

Если функция завершается ошибкой, возвращаемое значение равняется нулю.

Замечания

Если Вы регистрируете класс окна, используя функцию RegisterClassA, приложение сообщает системе, что окна созданного класса предполагают сообщения с текстом или символьными параметрами, которые используют символьный набор ANSI.

Если Вы регистрируете его, используя RegisterClassW, приложение требует, чтобы система посылала сообщения с текстовыми параметрами на Unicode.

Все классы окна, которые регистрирует приложение, становятся незарегистрированными, когда класс завершает работу.

Создание окна зарегистрированного класса

Для создания окна на основе некоторого класса служит функция CreateWindow. Она определяет класс, заголовок, стиль окна и (необязательно) начальную позицию и размер окна.

Синтаксис:

```
HWND CreateWindowEx(  
    DWORD dwExStyle, // улучшенный стиль окна  
    LPCTSTR lpClassName, // указатель на зарегистрированное  
                          имя класса  
    LPCTSTR lpWindowName, // указатель на имя окна  
    DWORD dwStyle, // стиль окна  
    int x, // горизонтальная позиция окна  
    int y, // вертикальная позиция окна  
    int nWidth, // ширина окна  
    int nHeight, // высота окна  
    HWND hWndParent, // дескриптор родительского или  
                     окна владельца  
    HMENU hMenu, // дескриптор меню или ID  
                     дочернего окна  
    HANDLE hInstance, // дескриптор экземпляра  
                     приложения
```


LPVOID lpParam // указатель на данные создания
окна

);

Параметры:

dwExStyle – Определяет расширенный стиль окна.

lpClassName – указатель на зарегистрированное имя класса. Имя класса может быть любым именем, зарегистрированным функцией RegisterClass или RegisterClassEx.

lpWindowName – Указывает на строку с нулевым символом на конце, которая определяет имя окна. Если стиль она определяет область заголовка, заголовок окна, указанный lpWindowName отображается в области заголовка.

dwStyle – Определяет стиль создаваемого окна. Этот параметр может быть комбинацией стилей окна.

x – Определяет начальную горизонтальную позицию окна. x - начальная x-координата левого верхнего угла окна.

Если этот параметр установлен как CW_USEDEFAULT, система выбирает заданную по умолчанию позицию для левого верхнего угла окна и игнорирует параметр y. Параметр CW_USEDEFAULT допустим только для перекрывающихся окон; если он определен для выскакивающего или дочернего окна, x и y параметры устанавливаются в нуль.

y – Определяет начальную вертикальную позицию окна. параметр y - начальная y-координата левого верхнего угла окна.

Если перекрывающее окно создано с установленным битом стиля WS_VISIBLE, а x параметр установлен как CW_USEDEFAULT, система игнорирует параметр y.

nWidth – Определяет ширину окна в единицах измерения для устройства. Для перекрывающихся окон, nWidth является, или шириной окна в экранных координатах, или параметром CW_USEDEFAULT. Если nWidth - CW_USEDEFAULT, система выбирает заданную по умолчанию ширину и высоту для окна; заданная по умолчанию ширина простирается от

начальной x-координаты до правого края экрана, а заданная по умолчанию высота простирается от начальной y-координаты до верхней части области пиктограмм. Значение CW_USEDEFAULT допустимо только для перекрывающихся окон; если CW_USEDEFAULT определено для выскакивающего или дочернего окна, nWidth и nHeight устанавливаются в нуль.

nHeight – Определяет высоту окна в единицах измерения устройства. Для перекрывающихся окон, nHeight - высота окна, в экранных координатах. Если параметр nWidth установлен как CW_USEDEFAULT, система игнорирует nHeight.

hWndParent – Дескриптор окна родителя или владельца создаваемого окна. Правильный дескриптор окна должен быть выдан при создании дочернего окна или окна владельца. Этот параметр является необязательным для выскакивающих окон.

hMenu – Дескриптор меню или определяет идентификатор дочернего окна в зависимости от стиля окна. Для перекрывающегося или выскакивающего окна, hMenu идентифицирует меню, которое будет использоваться с окном; если должно использоваться меню класса, он может быть значением ПУСТО (NULL). Для дочернего окна, параметр hMenu определяет идентификатор дочернего окна, целочисленное значение, используемое элементом управления диалогового окна, чтобы предупреждать своего родителя о событиях. Прикладная программа определяет идентификатор дочернего окна; он должен быть уникальным для всех дочерних окон того же самого родительского окна.

hInstance

- Windows 95/98/Me: Дескриптор экземпляра модуля, который будет связан с окном.
- Windows NT/2000/XP: Это значение игнорируется.

lpParam – Указывает на значение, переданное окну через структуру CREATESTRUCT, переданную в параметре lpParam сообщения

WM_CREATE. Если прикладная программа вызывает CreateWindow, чтобы создать рабочее окно многодокументного интерфейса (MDI), lpParam должен указывать на структуру CLIENTCREATESTRUCT.

Возвращаемые значения

Если функция завершается успешно, возвращаемое значение - дескриптор созданного окна.

Если функция завершилась ошибкой, возвращаемое значение - ПУСТО (NULL).

Замечания

Перед возвратом значения, функция CreateWindow отправляет сообщение WM_CREATE оконной процедуре. Для перекрывающихся, выскакивающих и дочерних окон CreateWindow отправляет окну сообщения WM_CREATE, WM_GETMINMAXINFO и WM_NCCREATE. Параметр lpParam сообщения WM_CREATE содержит указатель на структуру CREATESTRUCT. Если определен стиль WS_VISIBLE, CreateWindow отправляет окну все сообщения, требующиеся, чтобы активизировать и показать окно.

Показать созданное окно

Функция ShowWindow устанавливает состояние показа определяемого окна.

Синтаксис

```
BOOL ShowWindow (  
    HWND hWnd, // дескриптор окна  
    int nCmdShow // состояние показа окна  
);
```

Параметры

hWnd– Идентифицирует окно.

nCmdShow – Определяет, как окно должно быть показано. Этот параметр первоначально игнорируется, когда прикладная программа

вызывает ShowWindow, если программа, которая запустила прикладную программу, обеспечивает структуру STARTUPINFO. Иначе, при первом вызове функции ShowWindow, это значение должно быть значением, полученным функцией WinMain в ее параметре nCmdShow. В последующих обращениях, этот параметр может быть одним из следующих значений:

SW_HIDE - Скрывает окно и активизирует другое окно.

SW_MAXIMIZE - Развертывает определяемое окно.

SW_MINIMIZE - Свертывает определяемое окно и активизирует следующее окно верхнего уровня в Z-последовательности.

SW_RESTORE - Активизирует и отображает окно. Если окно свернуто или развернуто, Windows восстанавливает в его первоначальных размерах и позиции. Прикладная программа должна установить этот флажок при восстановлении свернутого окна.

SW_SHOW - Активизирует окно и отображает его текущие размеры и позицию.

SW_SHOWDEFAULT - Устанавливает состояние показа, основанное на флажке SW_, определенном в структуре STARTUPINFO, переданной в функцию CreateProcess программой, которая запустила прикладную программу.

SW_SHOWMAXIMIZED - Активизирует окно и отображает его как развернутое окно.

SW_SHOWMINIMIZED - Активизирует окно и отображает его как свернутое окно.

SW_SHOWMINNOACTIVE - Отображает окно как свернутое окно. Активное окно остается активным.

SW_SHOWNORMAL - Отображает окно в его текущем состоянии. Активное окно остается активным.

SW_SHOWNOACTIVATE - Отображает окно в его самом современном размере и позиции. Активное окно остается активным.

SW_SHOWNORMAL - Активизирует и отображает окно. Если окно свернуто или развернуто, Windows восстанавливает его в первоначальном размере и позиции. Прикладная программа должна установить этот флажок при отображении окна впервые.

Возвращаемые значения

Если функция завершилась успешно, возвращается значение отличное от нуля. Иначе возвращаемое значение - ноль.

Перерисовать видимую часть окна

Описание:

BOOL UpdateWindow

(HWND hWnd // указатель на окно
);

Если область обновления окна непуста, то посылает сообщение **WM_PAINT** прямо оконной функции данного окна.

Параметры:

hWnd: Идентификатор окна.

Сообщения

Сообщение - это запись определенной структуры, которая содержит всю информацию о происшедшем событии. Под событием понимается факт свершения элементарного действия, от которого может зависеть ход выполнения программы. Событиями, например, являются: нажатие клавиш на клавиатуре, перемещение мыши, истечение заданного промежутка времени.

Структура сообщения в Windows имеет следующий вид:

MSGSTRUCT STRUC

MSHWN	DD ? ; идентификатор окна, ; получающего сообщение
MSMESSAGE	DD ? ; идентификатор сообщения
MSWPARAM	DD ? ; доп. информация о сообщении
MSLPARAM	DD ? ; доп. информация о сообщении
MSTIME	DD ? ; время отправки сообщения

MSPT DD ? ; положение курсора, во время
посылки сообщения
MSGSTRUCT ENDS

Поле **MSHWND** содержит 16-разрядный дескриптор окна, в котором возникло сообщение.

В поле **MSMESSAGE** помещается двухбайтный код (тип) сообщения. Это может быть сообщение Windows или сообщение, определенное пользователем. Приложения могут использовать только младшее слово; старшее слово зарезервировано системой

Поля **MSWPARAM** и **MSLPARAM** содержат дополнительную информацию и зависят от типа сообщения. В сообщениях, определяемых пользователем, они могут использоваться для передачи необходимой информации.

В поле **MSTIME** система Windows помещает время в миллисекундах, которое истекло с момента запуска системы до постановки сообщения в очередь.

Поле **MSPT** указывает позицию курсора мыши в экранных координатах на момент возникновения события.

Сообщения могут поступать в программу от многочисленных источников:

- Пользователь генерирует сообщения, нажимая клавиши на клавиатуре или перемещая мышь и нажимая ее кнопки;
- Сама среда Windows может посылать сообщения прикладной программе для уведомления о тех или иных событиях;
- Программа может вызвать функции Windows, результатом которых является посылка сообщений программе;
- Прикладная программа может посылать сообщение самой себе;
- Прикладная программа может посылать сообщения другим прикладным программам;

Для каждого приложения Windows организует отдельную очередь сообщений прикладной программы, которая по умолчанию вмещает до восьми сообщений.

Обработку прикладной очереди сообщений осуществляет уже само приложение. Для этого программа организует так называемый цикл обработки сообщений. В нем осуществляется выбор нового сообщения из очереди прикладной программы и вызов диспетчера для его обработки соответствующей функцией.

Выбор нового сообщения из очереди прикладной программы

Функция GetMessage извлекает сообщение из очереди сообщений вызывающего потока и помещает его в заданную структуру. Эта функция регулирует поступление отправленных сообщений до тех пор, пока помещенное в очередь сообщение доступно для извлечения.

Синтаксис:

```
BOOL GetMessage(  
    LPMSG lpMsg,  
    HWND hWnd,  
    UINT wMsgFilterMin,  
    UINT wMsgFilterMax  
);
```

Параметры

lpMsg – Указатель на структуру MSG, которая принимает информацию из очереди сообщений потока.

hWnd – Дескриптор окна, чьи сообщения должны быть извлечены. Окно должно принадлежать вызывающему потоку. *Значение ПУСТО (NULL) имеет специальное предназначение: при этом GetMessage извлекает сообщения для любого окна, которое принадлежит вызывающему потоку, и сообщения потока, помещенные в очередь вызывающего потока при помощи использования функции PostThreadMessage.*

wMsgFilterMin – Определяет целочисленную величину самого маленького значения сообщения, которое будет извлечено.

wMsgFilterMax – Определяет целочисленную величину самого большого значения сообщения, которое будет извлечено.

Если wMsgFilterMin и wMsgFilterMax являются оба нулевыми, функция GetMessage возвращает все доступные сообщения (то есть никакой фильтрации в диапазоне значений не выполняется).

Возвращаемые значения:

Если функция извлекает сообщение WM_QUIT, величина возвращаемого значения - нуль., иначе - не нуль.

Если имеется ошибка, величина возвращаемого значения - (минус)1. Например, функция завершается ошибкой, если hWnd - недопустимый дескриптор окна или lpMsg - недопустимый указатель.

Замечания

Прикладная программа обычно использует величину возвращаемого значения, чтобы выявить, закончил ли работать главный цикл обработки сообщений и выходить ли из программы.

Функция GetMessage извлекает сообщения, связанные с окном, идентифицированным параметром hWnd или любого из его дочерних окон, как определено функцией IsChild, и в пределах диапазона значений сообщения, заданных параметрами wMsgFilterMin и wMsgFilterMax.

Обратите внимание! на то, что приложение может использовать в параметрах wMsgFilterMin и wMsgFilterMax только младшее слово ; старшее слово зарезервировано для системы.

Обратите внимание! на то, что функция GetMessage всегда извлекает сообщения WM_QUIT, независимо от того, какие значения Вы задаете для параметров wMsgFilterMin и wMsgFilterMax.

Транслировать клавиатурные сообщения в ASCII-коды

Функция TranslateMessage переводит сообщения виртуальных клавиш в символьные сообщения. Символьные сообщения помещаются в очереди

сообщений вызывающего потока для прочтения в следующий раз, когда поток вызовет функцию GetMessage или PeekMessage.

Синтаксис:

```
BOOL TranslateMessage(  
    const MSG* lpMsg  
);
```

Параметры

lpMsg – Указатель на структуру MSG, которая содержит информацию о сообщении извлеченную из очереди сообщений вызывающего потока при помощи использования функции GetMessage или PeekMessage.

Возвращаемые значения

Если сообщение переведено (то есть символьное сообщение помещено в очереди сообщений потока), величина возвращаемого значения не нуль, иначе- нуль.

Замечания

Функция TranslateMessage не изменяет сообщение, указанное параметром lpMsg.

Комбинации WM_KEYDOWN и WM_KEYUP создают сообщение WM_DEADCHAR или WM_CHAR. Комбинации WM_SYSKEYDOWN и WM_SYSKEYUP создают сообщение WM_SYSDEADCHAR или WM_SYSCHAR.

TranslateMessage создает сообщения WM_CHAR только для клавиш, которые отображают символы ASCII при помощи драйвера клавиатуры.

Если прикладные программы обрабатывают сообщения виртуальной клавиши для некоторой другой цели, они не должны вызывать TranslateMessage. Например, прикладная программа не должна вызывать TranslateMessage если функцией TranslateAccelerator возвращается значение не нуль. Обратите внимание! на то, что то, что приложение ответственно за извлечение и диспетчеризацию сообщений о вводе данных

блока диалога. Большинство прикладных программ использует для этого главный цикл сообщений. Однако, чтобы дать возможность пользователю перемещаться и выбирать органы управления при помощи использования клавиатуры, приложение должно обратиться к функции *IsDialogMessage*.

Вернуть управление Windows с передачей сообщения предназначенному окну.

Функция *DispatchMessage* распределяет сообщение оконной процедуре. Обычно она используется, чтобы доставить сообщение, извлеченное функцией *GetMessage*.

Синтаксис:

```
LRESULT DispatchMessage(  
    const MSG* lpmsg  
);
```

Параметры

lpmsg – Указатель на структуру *MSG*, которая содержит сообщение.

Возвращаемое значение

Величина возвращаемого значения определяется значением, которое возвращает оконная процедура. Несмотря на то, что это значение зависит от отправляемого сообщения, возвращаемое значение, как правило, игнорируется.

Замечания

Структура *MSG* должна содержать допустимые значения сообщений. Если параметр *lpmsg* указывает на сообщение *WM_TIMER*, а параметр *lParam* сообщения *WM_TIMER* имеет значение не ПУСТО (*NULL*), *lParam* указывает на функцию, которая была вызвана вместо оконной процедуры.

Закончить данный процесс со всеми подзадачами (потоками).

Функция *ExitProcess* заканчивает работу процесса и всех его потоков.

Синтаксис

```
VOID ExitProcess(  

```

UINT uExitCode // код выхода для всех потоков

);

Параметры

uExitCode – Определяет код выхода для процесса, и для всех потоков, которые завершают работу в результате вызова этой функции.

У этой функции нет возвращаемого значения.

Замечания

Функция *ExitProcess* - предпочтительный метод завершения процесса. Эта функция обеспечивает чистое отключение процесса. Такое завершение включает в себя вызов функций точек входа всех связанных динамически подключаемых библиотек (DLL) со значениями, указывающими, что процесс отключается от DLL. Если процесс заканчивается путем вызова *TerminateProcess*, DLL, к которым процесс подключен, не уведомляются о завершении процесса. После того, как все связанные DLL исполнили любое значение завершения, эта функция завершает работу текущего процесса.

Завершение процесса происходит по нижеследующим причинам:

1. Все дескрипторы объектов, открытые процессом, закрываются.
2. Все потоки в процессе завершают свою работу по исполнению кода.
3. Состояние объекта процесса становится сигнальным, удовлетворяя любые потоки, которые ждали завершения процесса.
4. Состояния всех потоков процесса, становятся сигнальными, удовлетворяя любые потоки, которые ждали завершения работы потоков.
5. Состояние завершения процесса изменяется из *STILL_ACTIVE* в значение выхода процесса.

Завершение процесса не заставляет дочерние процессы закончить свою работу.

Завершение процесса необязательно удаляет объект процесса из операционной системы. Объект процесса удаляется тогда, когда закрывается последний дескриптор процесса.

Внимание! Вызов функции ExitProcess в DLL может привести к неожиданному приложению или системным ошибкам. Убедитесь в том, что вызов функции ExitProcess из DLL происходит только в случае, если Вы знаете, какое приложение или элементы системы были загружены в DLL и в том, что в этом контексте вызвать функцию ExitProcess безопасно.

Функция ExitProcess не возвращает значения до тех пор, пока в потоках не отработают в их DLL процедуры инициализации или отключения.

Оконные функции

В связи с тем, что сообщения связаны с окнами, функции обработки сообщения называют оконными функциями

Функция **MessageBox** создает, показывает на экране и использует окно сообщения. Окно сообщения содержит определяемое программой сообщение и заголовок, плюс любая комбинация предопределенных значков и командных кнопок.

Синтаксис

```
int MessageBox(  
    HWND hWnd,  
    LPCTSTR lpText,  
    LPCTSTR lpCaption,  
    UINT uType  
);
```

Параметры

hWnd – Дескриптор окна владельца, которое создает окно сообщения. Если этот параметр - ПУСТО (NULL), окно сообщения не имеет окна владельца.

lpText – Указатель на символьную строку с нулем в конце, которая содержит сообщение показываемое на экране.

lpCaption – Указатель на символьную строку с нулем в конце, которая содержит заголовок диалогового окна (окна сообщения). Если этот параметр

- ПУСТО (NULL), используется заданный по умолчанию заголовок Error (Ошибка).

uType – Устанавливает содержание и режим работы диалогового окна. Этим параметром может быть комбинация флажков из ниже перечисленных групп флажков.

Чтобы обозначить кнопки, показываемые на экране в окне сообщения, задайте одно из ниже перечисленных значений. (вставить)

Функция **PostQuitMessage** указывает системе, что поток сделал запрос на то, чтобы завершить свою работу (выйти). Это обычно используется в ответ на сообщение WM_DESTROY.

Синтаксис

```
void PostQuitMessage(  
    int nExitCode  
);
```

Параметры

nExitCode – Определяет код завершения прикладной программы. Это значение используется как параметр wParam сообщения WM_QUIT.

Возвращаемых значений нет.

Замечания

Функция PostQuitMessage помещает сообщение WM_QUIT в очередь сообщений потока и немедленно возвращает значение; функция просто указывает системе, что потоку требуется прекратить свою работу в какое-то время в будущем.

Когда поток извлекает сообщение WM_QUIT из своей очереди сообщений, он должен выйти из своего цикла обработки сообщений и вернуть управление системе. Значение выхода, возвращенное системой должно быть параметром wParam сообщения WM_QUIT.

Функция **DefWindowProc** вызывается оконной процедурой по умолчанию, чтобы обеспечить обработку по умолчанию любого сообщения окна, которые приложение не обрабатывает. Эта функция гарантирует то, что обрабатывается каждое сообщение. Функция DefWindowProc вызывается с теми же самыми параметрами, принятыми оконной процедурой.

Синтаксис

```
LRESULT DefWindowProc(  
HWND hWnd,  
UINT Msg,  
WPARAM wParam,  
LPARAM lParam  
);
```

Параметры

hWnd – Дескриптор оконной процедуры, которая получает сообщение.

Msg – Определяет сообщение.

wParam – Определяет дополнительную информацию о сообщении.

Содержание этого параметра зависит от значения параметра ***Msg***.

lParam – Определяет дополнительную информацию о сообщении.

Содержание этого параметра зависит от значения параметра ***Msg***.

Возвращаемые значения

Величина возвращаемого значения определяет результат обработки сообщения и зависит от сообщения.

Отправка сообщения окну

Функция **SendMessage** отправляет заданное сообщение окну или окнам. Функция вызывает оконную процедуру для заданного окна и не возвращает значение до тех пор, пока оконная процедура не обработает сообщение.

*Чтобы отправить сообщение и вернуть немедленно значение, используйте функцию **SendMessageCallback** или **SendNotifyMessage**. Чтобы поместить сообщение в очередь сообщений потока и вернуть*

немедленно значение, используйте функцию *PostMessage* или *PostThreadMessage*.

Синтаксис

```
LRESULT SendMessage(  
    HWND hWnd,  
    UINT Msg,  
    WPARAM wParam,  
    LPARAM lParam  
);
```

Параметры

hWnd – Дескриптор окна, оконная процедура которого примет сообщение.

Msg – Определяет сообщение, которое будет отправлено.

wParam – Определяет дополнительную конкретизирующую сообщение информацию. Устанавливает максимальное число TCHAR (символов), которое будет скопировано, включая символ завершения ноль-терминатор.

lParam – Определяет дополнительную конкретизирующую сообщение информацию. Указатель на буфер, который принимает текст.

Возвращаемые значения

Величина возвращаемого значения определяет результат обработки сообщения; он зависит от отправленного сообщения.

Обработка сообщений в оконной функции

Приложение может иметь несколько оконных функций. Указатель на оконную функцию передается функции `RegisterClass` при регистрации класса окна. Таким образом, количество оконных функций определяется количеством зарегистрированных классов окон.

Когда создается экземпляр окна, то оконная функция будет связана с этим окном до конца работы приложения.

При поступлении сообщения `Windows` вызывает оконную функцию и передает ей параметры. (приложение выбирает сообщение из очереди

приложения и отправляет его Windows, а затем Windows вызывает нужную оконную функцию и передает ей при вызове параметры сообщения.

Формат оконной функции:

LRESULT CALLBACK WindowProc (HWND hwnd,UINT message,
WPARAM wparam, LPARAM lparam)

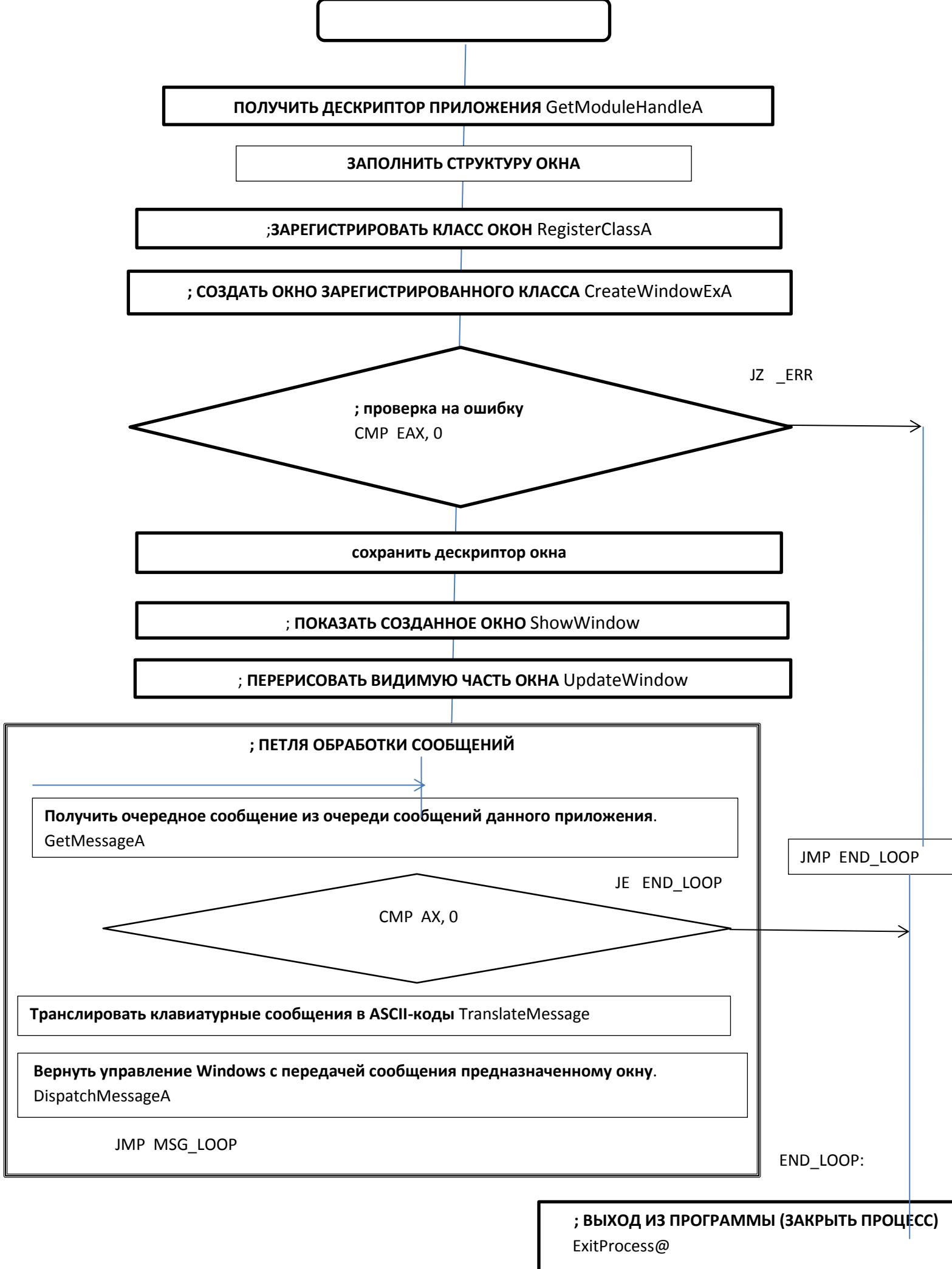
Hwnd – дескриптор окна, которому предназначено сообщение

Message – идентификатор сообщения (характеризует тип сообщения)

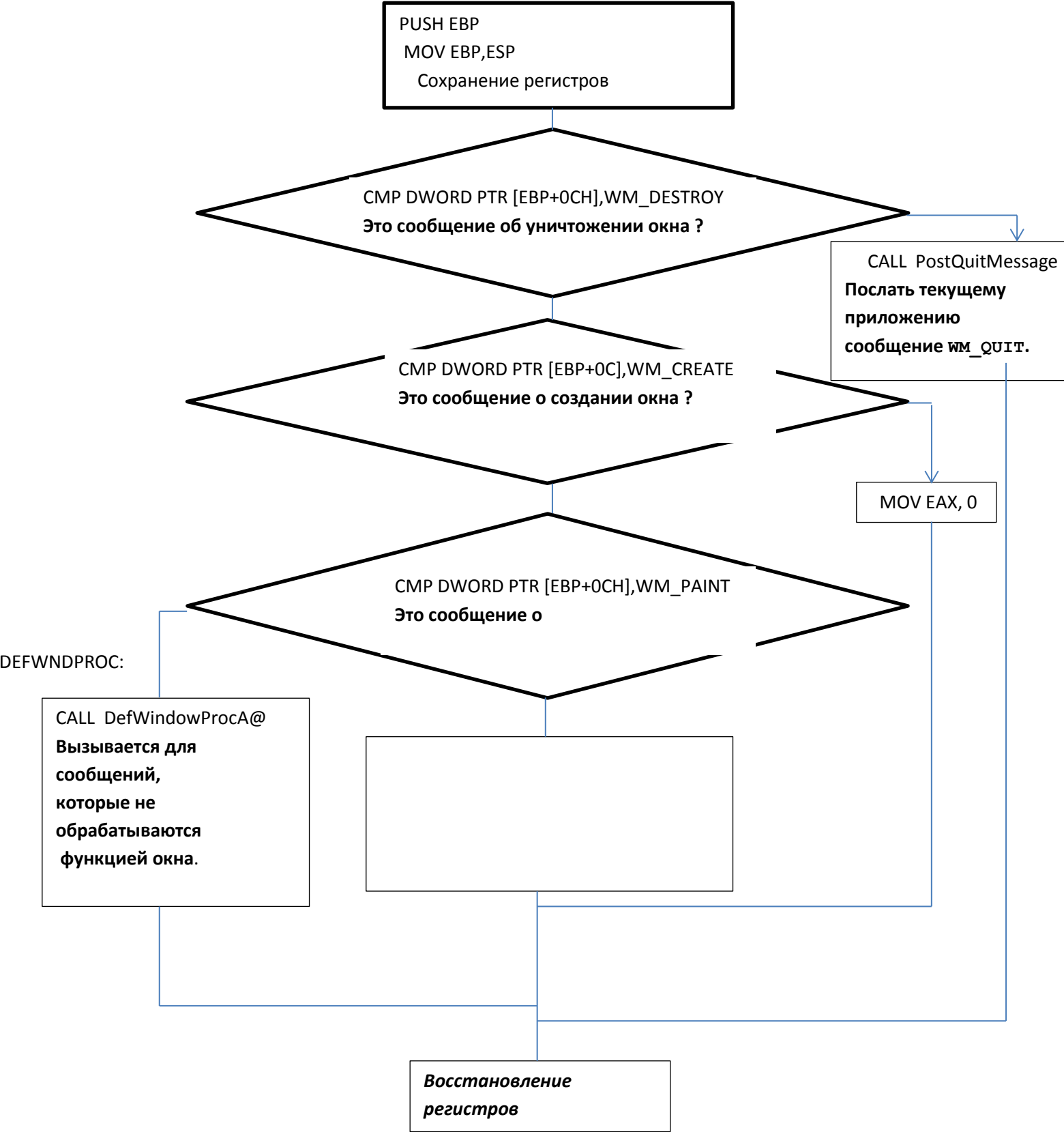
Wparam и lparam дополнительные параметры – копии соответствующих полей поступившего сообщения.

Оконная функция распознает характер сообщения и передает управление той ветви функции, которая выполняет соответствующие действия. Оконная функция должна в начале работы сохранять, в конце – восстанавливать регистры ebp, esp,edi. Это связано с повторной входимостью.

Блок-схема каркасного приложения



Блок-схема оконной процедуры



Задания

1. Создать окно с сообщением «Это оконное приложение»
2. Создать окно с сообщением «Окно» и одной кнопкой
3. Создать окно с сообщением «Пример» и двумя кнопками

Лабораторная работа №6

Консольные приложения

Консоль – устройства ввода-вывода, непосредственно подключенные к компьютеру (клавиатура и монитор). Консольными приложениями называют приложения, которые используют для связи с пользователем команды (текстовые строки), вводимые с клавиатуры.

Создание консоли

Консоль создается при помощи функции `Alloc Console`.

Функция не имеет параметров.

При успешном завершении возвращается ненулевое значение. В случае ошибки функция возвращает 0.

Получение дескриптора консоли

Функция `GetStdHandle` извлекает дескриптор для стандартного ввода данных стандартного вывода или стандартной ошибки устройства.

Синтаксис

```
HANDLE GetStdHandle(  
    DWORD nStdHandle // ввод, вывод или ошибка устройства  
);
```

Параметры

`nStdHandle` - Стандартное устройство, для которого дескриптор должен быть возвращен. Этот параметр может быть одним из следующих значений.

STD_INPUT_HANDLE - Дескриптор стандартного устройства ввода данных. Вначале, это - дескриптор консольного буфера ввода, equ -10.

STD_OUTPUT_HANDLE - Дескриптор устройства стандартного вывода. Вначале, это - дескриптор активного экранного буфера консоли, equ -11.

STD_ERROR_HANDLE- Дескриптор стандартной ошибки устройства. Вначале, это - дескриптор активного экранного буфера консоли, equ -12.

Возвращаемые значения

Если функция завершается успешно, возвращаемое значение - дескриптор определяемого устройства. Дескриптор имеет права доступа GENERIC_READ - для чтения и GENERIC_WRITE- для записи.

Если функция завершается с ошибкой, возвращаемое значение - флажок INVALID_HANDLE_VALUE.

Освобождение консоли

Функция FreeConsole освобождает консоль (завершает созданный процесс из консоли).

Возвращаемое значение:

В случае ошибки функция возвращает 0, иначе - ненулевое значение

Функция не имеет параметров.

Чтение из консоли

Функция ReadConsole читает символьный ввод данных из консольного буфера ввода и удаляет их из буфера.

Синтаксис

BOOL ReadConsole(

HANDLE hConsoleInput , // дескриптор буфера ввода консоли

LPVOID lpBuffer, // буфер данных

DWORD nNumberOfCharsToRead, // число символов для чтения

LPDWORD lpNumberOfCharsRead , // число прочитанных символов

LPVOID lpReserved // зарезервировано

);

Параметры

hConsoleInput - Дескриптор консольного буфера ввода. Дескриптор должен иметь право доступа GENERIC_READ (для чтения).

lpBuffer - Указатель на буфер, куда будет прочитана информация из консольного буфера ввода. Общий размер требуемого буфера, должен быть меньше чем 64КБ

nNumberOfCharsToRead - Размер буфера

lpNumberOfCharsRead - Указатель на переменную, которая принимает число фактически прочитанных символов.

lpReserved - Зарезервировано, должно быть ПУСТО (NULL).

Возвращаемые значения

Если функция завершается успешно, величина возвращаемого значения - не ноль. Иначе - ноль.

Вывод на консоль

Для вывода текстовой строки на экран применяется функция API WriteConsoleA.

Функция WriteConsole записывает символьную строку в экранный буфер консоли, начинающийся с текущей позиции курсора.

Синтаксис

BOOL WriteConsole(

HANDLE hConsoleOutput, // дескриптор экранного буфера

CONST VOID * lpBuffer, // буфер записи

DWORD nNumberOfCharsToWrite, // число символов для записи

LPDWORD lpNumberOfCharsWritten, // число записанных символов

LPVOID lpReserved // зарезервировано

);

Параметры

`hConsoleOutput` - Дескриптор экранного буфера консоли. Дескриптор можно получить при помощи функции `GetStdHandle`. Дескриптор должен иметь право доступа `GENERIC_WRITE` – доступ для записи.

`lpBuffer` - Указатель на буфер, содержащий выводимый текст. Общий размер должен быть меньше чем 64КБ.

`nNumberOfCharsToWrite` - Количество выводимых символов.

`lpNumberOfCharsWritten` - Указатель на переменную, в которую будет записано количество фактически выведенных символов.

`lpReserved` - Зарезервирован, должно быть ПУСТО (NULL).

Возвращаемые значения

Если функция завершается успешно, величина возвращаемого значения - не ноль. Иначе - ноль.

Замечания

Буфер, где находится выводимый текст необязательно должен заканчиваться нулем, так как в функцию передается количество выводимых символов.

Функция `WriteConsole` записывает символы в экранном буфере консоли в текущей позиции курсора. Позиция курсора продвигается вперед, по мере написания символов. Функция `SetConsoleCursorPosition` устанавливает текущую позицию курсора.

Функции `GetConsoleMode` -получить режимы ввода данных консольного буфера вводаизвлекать данные и

`SetConsoleMode` - установить режимы ввода данных консольного буфера ввода

Функция `SetConsoleTextAttribute` устанавливает атрибуты символов (цвета текста и цвета фона).

`GetConsoleScreenBufferInfo` -извлекает информацию о заданном экранном буфере консоли.

`SetConsoleTitle` – определение заголовка окна.

`SetConsoleCursorPosition` – установить позицию курсора.

Создание собственной консоли

начало

освободить уже существующую консоль
CALL FreeConsole@0

CALL AllocConsole

получить HANDLE ввода
CALL GetStdHandle@4

получить HANDLE вывода
CALL GetStdHandle

установить новый размер окна консоли
CALL SetConsoleScreenBufferSize

задать заголовок окна консоли
CALL SetConsoleTitleA

установить позицию курсора
CALL SetConsoleCursorPosition

задать цветовые атрибуты выводимого текста
CALL SetConsoleTextAttribute

вывести строку
CALL WriteConsoleA

ждать ввод строки
CALL ReadConsoleA

задать цветовые атрибуты выводимого
текста

```
CALL SetConsoleTextAttribute@8
```

вывести полученную строку

```
CALL WriteConsoleA
```

закрыть консоль

```
CALL FreeConsole@0
```

```
CALL ExitProcess@4
```

Задания

1. Написать программу, которая выводит на экран сумму двух чисел,
2. Написать программу, которая суммирует два числа, введенных с клавиатуры.
3. Написать программу, которая выводит на экран разность двух чисел,
4. Написать программу, которая вычисляет разность двух чисел, введенных с клавиатуры.