Санкт-Петербургское государственное бюджетное профессиональное образовательное учреждение
«Академия управления городской средой, градостроительства и печати»

УТВЕРЖДАЮ
Заместитель директора
по учебно-методической работе
О.В.Фомичева
«26» декабря 2025 г.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПЛАНИРОВАНИЮ И ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

ОП.07 «ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ»

специальности 09.02.13 Интеграция решений с применением технологий искусственного интеллекта

Форма обучения -очная

Санкт-Петербург
2025

Разработчик: Ипатова С.В./Оболенская Е.Г., методисты СПб ГБПОУ АУГСГиП

Одобрены на заседании цикловой комиссии

Общетехнических дисциплин и компьютерных технологий

Протокол № 4

От 09.12.2025 г.

Председатель цикловой комиссии:

Шурухина И.Е.

В рамках программы учебной дисциплины обучающимися осваиваются умения и знания

формируемые ПК, ОК, ЛР	Умения	Знания
ОК 01-02, ОК 04-06 ПК 2.2 ПК 2.5 ЛР13-17	– проектировать реляционную базу данных; – использовать язык запросов для программного извлечения сведений из баз данных;	– основы теории баз данных; – модели данных; – особенности реляционной модели и проектирование баз данных; – изобразительные средства, используемые в ER- моделировании; – основы реляционной алгебры; – принципы проектирования баз данных; – обеспечение непротиворечивости и целостности данных; – средства проектирования структур баз данных; – язык запросов SQL.

ОК 01. Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02. Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях.

ОК 04. Эффективно взаимодействовать и работать в коллективе и команде.

ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста.

ОК 06. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных российских духовно-нравственных ценностей, в том числе с учетом гармонизации межнациональных и межрелигиозных отношений, применять стандарты антикоррупционного поведения.

ПК 2.2. Осуществлять процедуры администрирования баз данных.

ПК 2.5. Подготавливать данные для базы знаний

Порядок выполнения самостоятельной работы обучающимся

Выполнение самостоятельных работ по учебной дисциплине предполагает:

1. Изучение необходимого теоретического материала по теме самостоятельной работы (конспектирование).
2. Постановку задачи Построение блок-схемы алгоритма решения задачи.
3. Составление программы.
4. Решение контрольного примера (численные значения исходных величин задаются студентом).
5. Подготовку отчета о выполненной работе и его защиту.

Каждая работа оформляется отдельно. Работы собираются в папку с титульным листом.

Критерии оценки

<table>
<tr><td rowspan="2">№ п/п</td><td rowspan="2">Критерии оценки</td><td>Работа выполнена</td><td>Работа выполнена не полностью</td><td>Работа не выполнена</td></tr>
<tr><td>Высокий уровень 3 балла</td><td>Средний уровень 2 балла</td><td>Низкий уровень 1 балл</td></tr>
<tr><td></td><td>Правильность и самостоятельность выполнения всех этапов работы</td><td>Практическая работа выполнена правильно и в установленный срок</td><td>При выполнении самостоятельной работы обучающийся допускал незначительные ошибки, часто обращался за помощью к преподавателю</td><td rowspan="3">1.Самостоятельная работа не выполнена
2.Обучающийся выполнял работу только с помощью преподавателя и других учащихся
3.Обучающийся не имеет конспекта и заготовки отчета по самостоятельной работе
4.Отчет выполнен и оформлен небрежно, без соблюдения установленных требований</td></tr>
<tr><td></td><td>Наличие конспекта, по теме самостоятельной работы
Наличие заготовки отчета к самостоятельной работе</td><td>Имеется заготовка отчета к самостоятельной работе
Содержание конспекта полностью соответствует теме самостоятельной работы</td><td>Заготовка отчета имеется в наличии, но с недочетами, не полными таблицами и т.п. Конспект имеется в наличии, но содержит не полный материал теме</td></tr>
<tr><td></td><td>Правильность оформления</td><td>Оформление отчета полностью соответствует требованиям</td><td>В оформлении отчета имеются незначительные недочеты и небольшая небрежность</td></tr>
<tr><td>Оценка</td><td colspan="2">Оценка4-5 баллов «удовлетворительно»</td><td>6-7 баллов «хорошо»</td><td>8-9 баллов «отлично»</td></tr>
</table>

Самостоятельная работа №1

Цель работы: Изучение приёмов работы с отладчиком ассемблера и с Help DOS на примере простых тестовых программ типа . exe и. com.

Программа на ассемблере в общем случае включает в себя сегмент данных, сегмент стека и кодовый сегмент, каждый из которых заключен в скобки <имя_сегмента> segment <параметры>…<имя_сегмента> ends.

Com-программа состоит из одной кодовой секции и не требует предварительной настройки сегментных регистров и стека. Приводимые ниже простые примеры могут быть использованы в качестве шаблонов для использования в данной и других работах по ассемблеру.

```
        Title prim_exe
; сегмент стека
Sseg segment para STACK ‘stack’
        Db 256 dup(?) ; резерв для стека
Sseg ends
; сегмент данных
dseg segment para public ‘data’
        db ‘что-нибудь: hello’; например, здесь вы можете написать свои ФИО
dseg ends
; сегмент кода
cseg segment para public ‘code’
        begin proc far
                assume ss:sseg, cs:cseg, ds:dseg ; назначение баз для транслятора
                push ds
                mov ax, 0
                push ax ; в стеке подготовлены данные для возврата по ret far
                mov ax, dseg
                mov ds, ax ; в ds адрес сегмента указанный для транслятора
                ;<набор команд>
                ret ; возврат на нулевое слово PSP с дальнейшим выходом в DOS
        begin endp
cseg ends
        END begin ; передавать управление на указанную точку входа

        Title prim_com
Cseg segment para ‘code’
        Assume cs:cseg, ds:cseg, ss:cseg ; все сегменты на один адрес
        ORG 100h ; резерв для PSP
        Begin: jmp main ; обход данных
        ; данные
        db ‘что-нибудь: hello’; например, здесь вы можете написать свои ФИО
        ; секция кода
        main proc near
                ;<набор команд>
                ret ; возврат на нулевое слово PSP с дальнейшим выходом в DOS
        main endp
Cseg ends
        End begin
```

Полученный текстовый файл обычно с расширением. asm подвергают трансляции, например, tasm fil_name, получая. obj, затем компоновщиком получают загрузочный файл нужного типа, например, tlink/t fil_name даёт нам fil_name. com, а без ключа /t - fil_name. exe.

Полученный исполняемый файл выполнить под управлением подходящего отладчика, например, afd file_name, отслеживая состояние регистров и области данных.

Самостоятельная работа №2

Цель работы: Изучение команд пересылки данных языка ассемблер с выводом сообщений на экран.

Команды пересылки данных занимают центральное место в архитектуре любото процессора. Условно их можно разбить на несколько групп: регистровые, память-регистр, стековые, память-память. Обмен типа память-память реализуется специальными командами, называемые цепочечными. Регистровые команды запрещают использование двух сегментных регистров.

Запись команд на ассемблере обычно задаётся в виде

{метка:} {префикс} Операция {операнды} {;комментарий},

где {} обозначают возможность пропуска,

значением метки является адрес (текущее значение счетчика команд), по которому можно передать управление,

префикс – rep и его модификации, используется в циклах,

операнды – регистры, переменные, метки, константы – имеют ассоциированный с ними атрибут типа (byte, word, dword…),

; определяет комментарий.

Операнды-переменные определяются директивами ассемблера DB, DW, DD, …, которые задают тип и размер памяти. Директива в общем случае имеет следующий формат

{имя} Директива {операнды} {; комментарий}.

Операнды-переменные характеризуются логическим адресом EA, который используется для вычислений физического адреса и могут задаваться следующим выражением

{Сегментн-рег. :}{тип ptr} имя_переменной_или_выражение {[B_reg+I_reg]},

где Сегментн-рег. Префикс сегментного регистра,

[B_reg+I_reg] – выражение с базовым и индексным регистром, которое может включать один из них или оба, регистры базы: BX, BP, индексные регистры: SI, DI.

Источники логического адреса EA

Обращение к памяти	База	Вариант	Смещение
Выборка команды	CS	Нет	IP
Стековая	SS	Нет	SP
Переменная	DS	CS, SS, ES	EA
Цепочка-источник	DS	CS, SS, ES	SI
Цепочка-получатель	ES	Нет	DI
BP как база	SS	CS, DS, ES	EA

Ниже в таблице даны варианты пересылок данных, которые требуется реализовать, используя возможно большее количество режимов адресации.

№ варианта	Требуемая перестановка	Тип исходных данных
0	BCDA	Byte, word, dword, byte
1	ACDB	Word, byte, dword, byte
2	ADBC	Dword, byte, word, byte
3	ABDC	Dword, word, byte, byte
4	BDAC	Word, dword, byte, byte
5	BADC	Word, byte, byte, dword
6	BCAD	Byte, dword, word, byte

7	CDAB	Byte, word, byte, dword
8	CBAD	Byte, dword, byte, word
9	CABD	Byte, byte, dword, word

Реализовать посимвольный ввод данных в ячейку с именем Pole, представленную ниже. Результат записать в ячейку поля памяти, подготовленную для вывода в нужном формате для прерывания BIOS и DOS.

Pole

A	B	C	D	Вспомогательное поле ввода-вывода

Условно рабочие ячейки в Pole поименованы буквами A, B, C, D для составления варианта задания обмена. Реализовать обмен с помощью регистров и стека, по возможности используя другие команды обмена и возможно большее количество режимов адресации.

Самостоятельная работа №3

Цель работы: Изучение арифметических и команд сдвига языка ассемблер с выводом сообщений на экран.

Команды арифметические включают в себя действия со знаковыми и беззнаковыми данными такие как сложить, вычесть, умножить, разделить и дополняются командами сдвигов и логическими, что даёт возможность вычислить произвольное выражение. Команды десятичной арифметики расширяют набор действий над данными, представленными в символьной форме или в упакованном формате. Для сохранения промежуточных результатов и исходных данных широко применяются стековые операции и операции пересылки.

Ниже предлагается вычислить выражение над данными, которые нужно ввести в диалоге с клавиатуры и вывести результат. Если операции нет соответствия общеупотребительного обозначения, то в таблице стоит мнемоника кода, в операциях сдвига предполагается сдвиг на единицу. Рабочее поле, как ранее, обозначено через Pole.

Pole

A	B	C	D	Вспомогательное поле ввода-вывода

Условно рабочие ячейки в Pole поименованы буквами A, B, C, D, их тип и требуемое выражение даны ниже в таблице. При вычислении нужно привести типы данных к более длинному формату.

№ варианта	Требуемое выражение	Тип исходных данных
0	(A/B-C)*D<AND>B<ROL>	Byte, word, word, byte
1	(AmodB-C)*D<OR>A<ROR>	Word, byte, word, byte
2	(A/B<XOR>C)*D-A<RCL>	Word, byte, word, byte
3	(<NOT>A-B/C)*D<RCR>	Word, word, byte, byte
4	(A/B<XOR>C-C)*D<SAR>	Word, word, byte, byte
5	(A/B<OR>D-C)*D<SHL>	Word, byte, byte, word
6	(A/B-C<AND>C)*D<SAL>	Byte, word, word, byte
7	(A/B-C)*D<OR>C<SHR>	Byte, word, byte, word
8	(A/B-C)*D<XOR>C<ROL>	Byte, word, byte, word
9	(AmodB<AND>C-C)*D<ROR>	Byte, byte, word, word

Самостоятельная работа №4

Цель работы: Изучение команд условных переходов и циклов языка ассемблер с выводом сообщений на экран.

Команды условных переходов и циклы широко применяются при программировании ветвящихся и циклических процессов. Мнемоника условных переходов задаётся сокращением слова JMP с добавлением условия перехода. Все команды имеют short формат передачи управления в пределах текущей страницы размером в 256 байт и при необходимости можно применять их совместно с командой безусловного перехода, имеющей полный набор передачи управления. Команды условных переходов представлены ниже в таблице.

Таблица команд условных переходов

Мнемоника	Отношение	Условие
JA/JNBE	>	CF|ZF=0
JAE/JNB/JNC	>=	CF=0
JB/JNAE/JC	<	CF=1
JBE/JNA	<=	CF|ZF=1
JE/JZ	=	ZF=1
JG/JNLE	>	(SF<XOR>OF)|ZF=0
JGE/JNL	>=	SF<XOR>OF=0
JL/JNGE	<	SF<XOR>OF=1
JLE/JNG	<=	(SF<XOR>OF)|ZF=1
JNE/JNZ	!=	ZF=0
JNO	Нет переполнения	OF=0
JNP/JPO	Нет паритета	PF=0
JNS	Нет знака	SF=0
JO	Есть переполнение	OF=1
JP/JPE	Есть паритет	PF=1
JS	Есть знак	SF=1
JCXZ	(CX)=0	(CX)=0

Команда цикла LOOP реализована по счетчику CX, уменьшающегося на 1 на каждом проходе, и, если (CX)!=0, то передача управления на метку. Модификации цикла дополнительно включают в условие бит ZF, так что
LOOPZ передаёт управление, если (CX)!=0<AND>ZF=1, а LOOPNZ, если (CX)!=0<AND>ZF=0.
Для управления флагами во всех случаях широко используются результаты предыдущих операций, если они устанавливают флаги, и команды сравнения.

В работе предлагается по введенной строке символов, считая байты как двоичные числа со знаком, найти сумму требуемых чисел.

№ варианта	Подсчитать сумму элементов строки
0	До первого отрицательного

1	Меньших по модулю заданного
2	Больше по модулю заданного
3	Всех отрицательных
4	Всех положительных
5	До первого положительного
6	До первого меньшего по модулю заданного
7	До первого большего по модулю заданного
8	N первых знаков, N – вводится с клавиатуры
9	N последних знаков, N – вводится с клавиатуры

Самостоятельная работа №5

Цель работы: Изучение команд языка ассемблер для работы с десятичными данными с выводом сообщений на экран.

Команды десятичной арифметики расширяют набор действий над данными, представленными в символьной форме ASCII или в упакованном BCD- формате. В символьном формате в шестнадцатиричном представлении цифры представлены как <3ц>, а в упакованном <цц> или же с нулевой зоной как в операциях умножения <0ц>. После выполнения обычных операций сложения, вычитания и умножения проводится десятичная коррекция, а для команды деления коррекция проводится до выполнения деления. Команды коррекции DAA, DAS, AAM, AAD при работе в BCD-формате; AAA, AAS для сложения и вычитания в ASCII-формате.

Варианты выражений с десятичными данными

№ варианта	Требуемое выражение	Формат исходных Данных
0	(A/B-C)*D+B	Byte, ASCII, ASCII, BCD
1	(A+B-C)*D	ASCII, BCD, ASCII, BCD
2	(A/B+C)*D-A	ASCII, BCD, ASCII, BCD
3	(A-B/C)*D	ASCII, ASCII, BCD, BCD
4	(A/B -C)*D	ASCII, ASCII, BCD, BCD
5	(A/B+D-C)*D	ASCII, BCD, BCD, ASCII
6	(A/B-C)*D+A	BCD, ASCII, ASCII, BCD
7	(A/B-C)*D+C	BCD, ASCII, BCD, ASCII
8	(A/B-C)*D+C	BCD, ASCII, BCD, ASCII
9	(A/B-C)*D+A	BCD, BCD, ASCII, ASCII

Самостоятельная работа №6

Цель работы: Изучение команд пересылки цепочек данных и перекодирования языка ассемблер с выводом сообщений на экран.

Команды перекодирования и цепочечные широко используются при работе с текстовыми данными как непосредственно для кодирования-декодирования с целью ‘сокрытия’ данных, так и для различных форм поиска и подсчета в тексте. Использование префикса rep расширяет возможности команд.

Основные команды цепочек MOVS, CMPS, SCAS, LODS, STOS требуют подготовки регистров источника и получателя, установки бита направления DF, а также указания типа данных B(yte)или W(ord).

Команда перекодирования XLAT предварительно требует установки регистра BX на начало таблицы, а перекодируемый символ в AL, здесь же и результат из таблицы.

В работе требуется ввести строку символов с разделителями и выполнить требуемый подсчет из таблицы вариантов.

№ варианта	Подсчитать сумму количества элементов строки

0	Согласных
1	Гласных
2	Разделителей
3	Слов
4	Предложений
5	Шипящих согласных
6	Твердых и мягких знаков
7	Знаков меньших по величине заданного
8	Знаков больших по величине заданного
9	Знаков без разделителей

Самостоятельная работа №7

Цель работы: Изучение принципов построения подпрограмм и функций на языке ассемблера

Подпрограммы и функции являются основным средством структурного программирования. В языке ассемблера отличия подпрограмм и функций нет, все они вызываются с помощью команды Call и могут быть вызваны также командой Jmp при предварительной подготовке стека. В других языках программирования подпрограммы и функции отличаются способом вызова, например, язык Paskal, а в языке Си определены только функции. Подпрограмме/функции передаются параметры, которые в теле процедуры используются как аргументы.

Способов передачи параметров несколько:

-использование глобальных переменных;

-использование регистов процессора;

-копирование значений параметров (передача по значению) в стек.

Глобальные переменные снижают гибкость программы и не рекомендуются для применения. Передача через регистры процессора наиболее быстрый и эффективный способ и поэтому применяется в системных программах обработки прерываний, однако небольшое количество регистров процессора ограничивает количество параметров Передача по значению через стек получила наибольшее распространение в языках программирования. Значением может быть некоторая константа или адрес, в последнем случае имеется возможность изменить значения по месту расположения параметра, что обычно используется в подпрограммах и может использоваться в функции как побочный эффект.

Процедура должна вернуть признак безошибочного выполнения установкой флага процессора CF в нуль, в противном случае в регистре AX возвращается код ошибки, используемый в точке вызова для идентификации ошибки и принятия решения. Если вид ошибки единственный, а возвращаемое значение позволяет идентифицировать ошибку, то признак ошибки может содержаться в возвращаемом значении функции, например, -1, если нормальное знчение >=0.

При написании на ассемблере подпрограмм/функций для использования в языках программирования нужно знать имена секций, что легко определить по листингу при трансляции любого модуля, написанного на нужном языке. Например, для процедур на языке Си сегмент кода имеет имя _TEXT, сегмент данных _DATE.

Отличие в способе развертывания строки параметров: в Си слева-направо, в Paskal справа-налево. При возврате из процедуры в паскальподобных языках стек очищается при возврате из процедуры, в Си стек очищается от параметров в точке вызова, что в принципе даёт возможность использования возвращаемых значений и в стеке, однако не рекомендуется, так как операция коррекции add sp, <n> вставляется сразу в точке возврата, а прерывания процессора не запрещены, и, следовательно, при наступлении прерывания значения в стеке будут потеряны. Для программ на ассемблере место очистки и способ использования стека не принципиальны, однако лучше придерживаться используемой в языках программирования идеологии.

Ниже в таблице предлагается реализовать на ассемблере функцию для работы со строками данных с передачей параметров по значению через стек, для четных вариантов использовать стиль Paskal, для нечетных – Си. Использование функции протестировать.

Используемые обозначения: char * - указатель (адрес) на строку символов, Int – данные целого типа, str – строка символов, tbl – ссылка на таблицу, char – символ, inpos – позиция поиска слева, inposr- позиция поиска справа, kld- количество удаляемых (замещаемых) символов, str_ins, strin, stradd- дополнительные строки. Названия функций смысловые.

№ варианта	Функции, аргументы и их тип	Возвращаемое значение
0	Char * Lower(str)	DS:BX
1	Char * Revers(str)	ES:DX
2	Int Lenstr(str).	AX
3	Char * stuff(str, inpos, kld, str_ins)	ES:DX
4	Int pos(str, inpos, strin)	AX
5	Char * addstr(str, stradd)	DS:DX
6	Char * strprkd(str, tbl)	ES:BX
7	Int At(str, inpos, char)	AX
8	Int Rat(str, inposr, char)	AX
9	Char * Upper(str)	DS:BX

Самостоятельная работа №8

Цель работы: Изучение принципов построения драйверов.

Драйвер устройства устанавливается путем включения имени готовой программы в файл конфигурации системы. Для установки пробной программы поместите в файл CONFIG. SYS строку DEVICE = DEVICE12. COM. Затем перезагрузите систему для установки драйвера. Если машина не будет загружаться, то скорее всего имеется ошибка в коде инициализации драйвера.

После того как драйвер установлен, для доступа к нему используются обычные функции MS DOS прерывания 21H. Какие функции можно использовать зависит от того, заменяет ли устройство стандартное устройство DOS (как в приведенном примере) или оно добавляется как совершенно новое устройство.

Для замены стандартного последовательного устройства, назовите драйвер AUX, после чего функции 3 и 4 прерывания 21H будут осуществлять соответственно ввод и вывод. Если устройство параллельное, то назовите его PRN, после чего функция 5 будет выводить данные на ‘принтер’. Другой возможностью является использование функции 3FH для ввода и 4FH для вывода. В этом случае используйте номер файла 3 - для последовательного устройства и 4 - для

параллельного. Напоминаем, что при использовании предопределенных номеров файла нет необходимости открывать устройство.

Если устройство не заменяет одно из стандартных устройств MS DOS (т. е. если оно не названо одним из резервных слов, таким как PRN, AUX и т. д.), то Вы можете открыть устройство с помощью одной из функций для открытия файла. Вы можете использовать как метод доступа с помощью управляющего блока файла, так и метод дескриптора файла, хотя последний предпочтительнее. Чтобы быть уверенным, что Вы по ошибке не откроете дисковый файл, поместите номер файла в BX, 0 - в AL, посде чего выполните функцию 44H прерывания 21H. Это функция IOCTL и если бит 7 значения, возвращаемого в DL установлен, то драйвер устройства загружен.

IOCTL требует, чтобы в байте атрибутов драйвера была соот ветствующая установка битов и чтобы по крайней мере основы процедуры обработки IOCTL имелись в процедуре обработчика прерывания драйвера. Функция IOCTL имеет 8 подфункций, пронумерованных от 0 до 7, при этом соответствующий кодовый номер помещается в AL при вызове функции:

0 Возвратить информацию об устройстве в DX
1 Установить информацию об устройстве, используя DL (DH=0)
2 Считать CX байтов от драйвера устройства через управля щий канал и поместить их начиная с DS:DX
3 Записать CX байтов в драйвер устройства через управляющий канал, взяв их начиная с DS:DX
4 То же, что и 2, но использовать номер накопителя в BL, где 0 = накопитель по умолчанию, 1 = A и т. д.
5 То же, что и 3, но использовать номер накопителя как в 5
6 Получить статус ввода
7 Получить статус вывода

В ответ возвращается различная информация, в зависимости от того, какая функция вызвана. Для подфункций 0 и 1 значение битов регистра DX следующее (при условии, что бит 7 = 1, что означает, что доступ получен к устройству, а не к файлу):

0 1 = устройство консольного ввода
1 1 = устройство консольного вывода
2 1 = нулевое устройство
3 1 = устройство часы
4 резерв
5 1 = нет проверки на Ctrl-Z, 0 = есть проверка на Ctrl-Z
6 1 = не конец файла, 0 = конец файла
7 1 = устройство, 0 = дисковый файл
8-13 резерв
14 1 = если можно использовать подфункции 2 и 3, 0 = нельзя
15 резерв

Подфункции 2-5 позволяют программе и устройству обмениваться произвольными управляющими строками. Это позволяет передавать управляющие сообщения отдельно от основного потока данных, что существенно упрощает дело. При возврате AX будет содержать число переданных байтов. Подфункции 6-7 позволяют программе проверить, готово ли устройство для ввода или вывода. Для устройств в AL возвращается FF, если устройство готово и 0, если нет. При использовании с открытым файлом (бит 7 = 0) в AL возвращается FF до тех пор, пока не будет достигнут конец файла.

Начальный код для драйвера определяет его как COM программу.

Драйвер устройства должен начинаться с заголовка драйвера. Он имеет длину 18 байтов, разделенных на 5 полей.

```
;устанавливаем кодовый сегмент
CSEG SEGMENT PUBLIC 'CODE'
ORG 0 ;эта строка необязательна
ASSUME CS:CSEG, DS:CSEG, ES:CSEG
DEVICE12 PROC FAR ;драйвер это далекая процедура
DD 0FFFFFFFFH ;адрес следующего драйвера
DW 8000H ;атрибуты драйвера
DW DEV_STRATEGY ;адрес процедуры стратегии
DW DEV_INTERRUPT ;адрес процедуры прерывания
DB 'AUX     ' ;имя устройство (дополненное пробелами до 8)
```

Первое поле (DD) всегда содержит значение -1 (FFFFFFFFH), и когда MS DOS загружает драйвер, то оно заменяется на стартовый адрес следующего драйвера. Таким образом, система может искать следующий драйвер по цепочке. У последнего загруженного драйвера в этом поле остается значение -1.

В слове атрибутов имеют значение только 7 битов:

бит
15 1 = символьное устройство, 0 = блочное устройство
14 1 = поддерживает IOCTL, 0 = не поддерживает IOCTL
13 1 = формат блоков IBM, 0 = другой формат блоков
3 1 = часы, 0 = не часы
2 1 = нулевое устройство, 0 = не нулевое устройство
1 1 = устройство стандартного вывода, 0 = нет
0 1 = устройство стандартного ввода, 0 = нет

Обычно установлен только бит 15, или биты 15 и 14, если устройство поддерживает IOCTL. Бит 13 устанавливается только для блочных устройств. Остальные биты используются для замены устройств, используемых MS DOS по умолчанию. Устройствами стандартного ввода и вывода являются клавиатура и видеодисплей; устройство часов объединяет часы реального времени с часами времени суток BIOS, а нулевое устройство (NULL) - это псевдоустройство, используемое для тестовых целей.

Последнее поле содержит имя устройства. Для замены существующих в DOS устройств, таких как LPT1 или COM1, используйте то же имя устройства, как в данном примере.

Процедура стратегии устройства требует только пяти строк.

Когда система загружает устройство, то она создает блок данных, называемый заголовком запроса. Он имеет две функции. Во-первых, он служит областью данных для внутренних операций системы. Более важно то, что заголовок запроса служит областью, через которую происходит обмен информацией между драйвером и вызывающей его программой. Например, когда драйвер выводит данные, то ему дается адрес данных через заголовок запроса. Когда же драйвер завершает свою работу, то он устанавливает в заголовке запроса байт статуса, который доступен вызывающей программе, тем самым давая возможность ей узнать об ошибке.

MS DOS создает заголовок запроса при установке драйвера устройства (когда система загружается). Процедура стратегии устройства выполняется только один раз в этот момент. При этом ES:BX указывают на вновь созданный заголовок запроса и процедуре нужно просто скопировать их, чтобы впоследствии он мог быть обнаружен при обращении к драйверу. Адреса смещения и сегмента заголовка помещаются в две переменные. В следующем разделе Вы увидите, что при обращении к драйверу, первое что он делает - восстанавливает значения ES:BX, чтобы можно было получить информацию из заголовка запроса.

Размер заголовка запроса может меняться, в зависимости от типа сделанного запроса к драйверу (напр. инициализация, вывод данных или возврат статуса). Однако первые 13 байт заголовка всегда одни и те же. Их формат таков:

1. Длина заголовка запроса (DB).
2. Код устройства (DB). Определяет номер для блочных устройств.
3. Код команды (DB). Здесь хранится номер последней посланной драйверу команды.
4. Статус (DW). Статус устанавливается каждый раз при вызове драйвера. Если установлен бит 15, то в младших восьми битах находится код ошибки.
5. Резервная область (8 байтов). Используется MS DOS.
6. Данные необходимые для работы драйвера (переменной длины).

Вот 5 строк процедуры стратегии устройства. Отмечаем, что две словные переменные, хранящие значения ES и BX, следуют за инструкцией RET, как и положено в формате COM.

```
DEV_STRATEGY: MOV CS:KEEP_ES,ES
MOV CS:KEEP_BX,BX
RET
KEEP_ES DW ?
KEEP_BX DW ?
```

Создание обработчика прерывания устройства.

Драйвер устройства начинается с двух порций кода, приведенных в предыдущих разделах. За ними должна следовать соответствующая процедура обработки прерывания. На самом деле, это неверно, называть эту процедуру процедурой обработки прерывания, так как она вовсе не обслуживает прерывание и завершается обычной инструкцией RET.

Имеется 13 типов функций, которые может выполнять устанавливаемый драйвер устройства. Когда драйвер вызывается функцией DOS (скажем функцией 3FH прерывания 21H, которая читает данные из файла или устройства), то функция помещает кодовый номер от 1 до 13 в однобайтное поле по смещению 2 в заголовке запроса (для ввода - кодовый номер 5). Затем управление передается процедуре обработки прерывания драйвера, адрес которой определяется при просмотре заголовка драйвера. Эта процедура в первую очередь восстанавливает ES:BX, с тем чтобы они указывали на заголовок запроса, а затем читает кодовый номер команды. По этому коду процедура обработки прерывания вызывает нужную процедуру, которая выполнит требуемую функцию. Процедура ищется с помощью 13-словной таблицы, содержащей смещения для 13 типов функций.

Функции всегда перечисляются в следующем порядке:

1. INITIALIZE (инициализация)
2. CHECK_MEDIA (проверка носителя)
3. MAKE_BPB
4. IOCTL_IN
5. INPUT_DATA (ввод данных)
6. NONDESTRUCT_IN
7. INPUT_STATUS (статус ввода)
8. CLEAR_INPUT (очистка ввода)
9. OUTPUT_DATA (вывод данных)
10. OUTPUT_VERIFY (проверка вывода)
11. OUTPUT_STATUS (статус вывода)
12. CLEAR_OUTPUT (очистка вывода)
13. IOCTL_OUT

После завершения процедуры, процедура обработки прерывания завершается инструкцией RET и управление возвращается в вызывающую программу. Драйвер устройства может включать код для обработки только некоторых функций, в зависимости от устройства и требуемой степени контроля ошибок и управления устройством. Номера функций, для которых не написаны процедуры, должны завершаться выходом из драйвера без выполнения чего-либо. В этом случае надо только перед выходом установить биты 15, 8, 1 и 0 в заголовке запроса, чтобы информировать вызывающую задачу, что была затребована несуществующая функция (бит 15 индицирует ошибку, бит 8 показывает, что драйвер работает нормально, а биты 0 и 1 дают код ошибки 3, что соответствует "неизвестной команде").

Но одна функция должна присутствовать во всех драйверах устройств, и это функция номер 1 - инициализация. Эта функция автоматически выполняется при загрузке драйвера, а затем нет. Одна из важных задач, выполняемая этой процедурой, состоит установке адреса конца драйвера в четырех байтах, начинающихся со смещения 14 в заголовке запроса. В нижеприведенном примере конец программы отмечен меткой eop:. Кроме этой задачи, процедура инициализации должна также выполнить всю необходимую для данного устройства инициализацию. На рис. показана структура драйвера устройства.

Какие из оставшихся функций будут включены в драйвер устройства зависит от того, что драйвер должен делать. Некоторые, такие как CHECK_MEDIA и MAKE_BPB, относятся только к блочным устройствам (они устанавливают тип диска, размер секторов и т. д.). Для символьных устройств наиболее важными являются две функции:

INPUT_DATA и OUTPUT_DATA (отметим, что эти имена несущественны - важна позиция в таблице функций, которая неизменна).

В обоих случаях заголовок запроса имеет следующую структуру:

13 байтов - стандартный формат заголовка запроса,
1 байт - байт описания среды (только для блочных устройств),
4 байта - смещение/сегмент буфера обмена данных,
2 байта - число байтов, которое надо передать,
2 байта - стартовый номер сектора (только для блочных устройств).

В нижеприведенном примере используется функция вывода. Процедура, выполняющая вывод получает из заголовка запроса адрес буфера, в котором находятся выводимые данные (смещение 14). Она также считывает число байтов, которое надо вывести (смещение 18). Когда процедура завершит вывод данных, то она установит слово статуса в заголовке запроса (смещение 3) и возвратит управление. Если операция успешна, то надо установить бит 8 слова статуса. Другие возможности будут обсуждены позднее.

В данном примере приведена общая форма процедуры обработки прерывания, не включая реального кода, управляющего устройством.

```
;---инициализация обработчика прерывания устройства
DEV_INTERRUPT: PUSH ES ;сохраняем регистры
PUSH DS
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH SI
PUSH DI
PUSH BP
```

```
MOV AX, CS:KEEP_ES ;ES:BX указатель на заголовок запроса
MOV ES, AX ;
MOV BX, CS:KEEP_BX ;
MOV AL, ES:[BX]+2 ;получаем код команды из заголовка
SHL AL, 1 ;умножаем на 2 (т. к. таблица словная)
SUB AH, AH ;обнуляем AH
LEA DI, FUNCTIONS ;DI указывает на смещение до таблицы
ADD DI, AX ;добавляем смещение в таблице
JMP WORD PTR [DI] ;переход на адрес из таблицы

FUNCTIONS LABEL WORD ;это таблица функций
DW INITIALIZE
DW CHECK_MEDIA
DW MAKE_BPB
DW IOCTL_IN
DW INPUT_DATA
DW NONDESTRUCT_IN
DW INPUT_STATUS
DW CLEAR_INPUT
DW OUTPUT_DATA
DW OUTPUT_VERIFY
DW OUTPUT_STATUS
DW CLEAR_OUTPUT
DW IOCTL_OUT

;---выход из драйвера, если функция не поддерживается
CHECK_MEDIA:
MAKE_BPB:
IOCTL_IN:
INPUT_DATA:
NONDESTRUCT_IN:
INPUT_STATUS:
CLEAR_INPUT:
OUTPUT_VERIFY:
OUTPUT_STATUS:
CLEAR_OUTPUT:
IOCTL_OUT:
OR ES:WORD PTR [BX]+3, 8103H ;модифицируем статус
JMP QUIT;---выход из драйвера

;---процедуры для двух поддерживаемых кодов
INITIALIZE: LEA AX, E_O_P ;смещение конца программы
 MOV ES:WORD PTR [BX]+14, AX ;помещаем его в заголовок
 MOV ES:WORD PTR [BX]+16, CS ;
; .
; (здесь идет инициализация устройства)
; .
JMP QUIT;---выход из драйвера
;
OUTPUT_DATA: MOV CX, ES:[BX]+18 ; число символов в CX
MOV AX, ES:[BX]+16 ;получаем адрес буфера данных
```

```
MOV DS, AX ;
MOV DX, ES:[BX]+14 ;
; .
; (здесь идут операции по выводу)
;
; call far ptr initpr
MOV BX, DX
DEC BX
L3: mov si, cx
MOV AL, [bx+si]
MOV AH, 0eh
INT 10H
LOOP L3
inc bx
JMP QUIT

;---выход с модификацией байт статуса в заголовке запроса
QUIT: OR ES:WORD PTR [BX]+3, 100H ;устанавливаем бит 8
POP BP ;восстанавливаем регистры
POP DI ;
POP SI ;
POP DX ;
POP CX ;
POP BX ;
POP AX ;
POP DS ;
POP ES ;
RET
E_O_P: ;метка конца программы
DEVICE12 ENDP
```

Исполняемый файл в современных системах может быть нескольких видов:. SYS, . COM и даже. EXE.

Чтобы получить из приведенного примера подключаемый драйвер вначале с помощью LINK.EXE получаем. EXE файл, далее воспользовавшись прогаммой EXE2BIN преобразуем файл в требуемый формат через. BIN файл, переименовав его далее в. COM или. SYS, можно получить требуемое расширение и сразу, указав полное имя выходного файла.

Интересный вариант драйвера получается при использовании формата. EXE, в этом случае можно подключить перекрываемую секцию с выводом вспомогательной информации как показано ниже.

INIT=1; Для формирования. exe файла

```
IF INIT EQ 1
INITPR PROC far
PUSH DS
MOV AX, 0
PUSH AX
ASSUME DS:cseg, cs:cseg
MOV AX, CSEG
mov ds, ax
comvzv: JMP SHORT L1
msg db 'Davydov', 0ah, 0dh, '$'
```

```
L1:
LEA DX, MSG
MOV AH, 09h
INT 21H
MOV BP, SP
MOV AX, [BP]
CMP AX, 0
JNZ L5
ret near
L5: ret far
INITPR ENDP
ENDIF
CSEG ENDS
END initpr; DEVICE12;inlbl
;
```

Перед возвратом драйвер устанавливает слово статуса в заголовке запроса. В данном примере это делается в двух местах, в зависимости от того вызывалась функция обеспечиваемая драйвером или нет. Эти строки выглядят так: OR ES:WORD PTR [BX]+3, XXXXH.
Значение битов XXXX следующее:

биты 0-7 код ошибки (если бит 15 = 1)
бит 8 устанавливается в 1, когда функция завершена
бит 9 устанавливается в 1, когда драйвер занят
биты 10-14 зарезервированы MS DOS
бит 15 устанавливается при возникновении ошибки

Если установлен бит 15, индицирующий ошибку, младший байт этого слова содержит следующие коды ошибок:

0 - попытка записи на защищенное от записи устройство,
1 - неизвестное устройство,
2 - устройство не готово,
3 - неизвестная команда,
4 - ошибка проверки по контрольной сумме,
5 - неверная длина запроса к устройству,
6 - ошибка поиска,
7 - неизвестный носитель,
8 - сектор не найден,
9 - нет бумаги в принтере,
A - ошибка записи,
B - ошибка чтения,
C - общая ошибка.

Реализовать драйвер с минимальным набором функций, обеспечивающего работу с программой тестирования, написанной на любом языке программирования.

№ варианта	Имя устройства	Тип файла
0	PRN	SYS
1	LPT1	COM
2	AUX	EXE
3	MAIN	EXE

4	LABDR	SYS
5	STDIN1	COM
6	STDOUT1	SYS
7	INOUT	EXE
8	OUTINC	COM
9	EXO16	SYS

Самостоятельная работа №9

Цель работы: Изучение основных типов данных и операций языка Си.

К основным типам данных относятся скалярные переменные, включая производный тип - указатель. Любые данные перед их использованием в программе должны быть объявлены:

Класс_памяти область_видимости тип имя;

В зависимости от места объявления переменные могут быть глобальными, если место их определения вне тела функции, и локальные, они описаны в блоках тела функции. Глобальные переменные имеют статическую область памяти и доступны и из других файлов, если они объявлены с ключевым словом extern. Локальные переменные обычно размещаются в стеке и известны внутри своего блока, если локальная переменная объявлена с ключевым словом static, то она размещается в статической области, и сохраняет своё значение после выхода из тела функции. Любые переменные имеют значение и адрес, за исключением переменных, объявленных с ключевым словом register, они не имеют адреса, и выделение регистра для них не гарантировано.

Основными типами являются символьные данные (char), данные целого типа (int) с модификаторами длины (short, long, double). Эти типы могут трактоваться как беззнаковые (unsigned). Данные с плавающей точкой используютя в инженерных расчетах, имеют тип float с модификаторами длины (long, double). Например:

```
extern unsigned char perekl; /* внешняя переменная из другого файла */
char * pperekl; /* указатель на переменную типа char, область действия и тип памяти
определяется местом объявления */
int i; // переменная целого типа.
```

Из переменных составляется выражение, которое состоит из операций, одна из них операция присвоить (=). В выражение может входить несколько операций присвоения. Если выражение закончено знаком (;), то оно образует оператор следования. Операции присвоения могут быть явными, включающими знак присвоить и неявными (++ и --). Явные операции могут быть составными <знак_операции>=, например, записи

<переменная>=<переменная> <операция><выражение>

соответствует более короткая запись в стиле языка Си

<переменная><операция>=<выражение>,

где допустимыми могут быть следующие опреации, перечисленные через запятую : *, /, %, +, -, <<, >>, &, ^, | (умножение, деление, Операция по модулю, суммирование, вычитание, сдвиг влево, сдвиг вправо, побитовая операция И, побитовая исключающая ИЛИ, побитовая ИЛИ).

Операции объединяются по группам и для каждой группы устанавливается приоритет. Порядок вычислений в выражении определяет приоритет операции.

Ниже предлагаются варианты для вычисления выражений. Тип переменных задан для (a, b, c). Начальные значения вводятся в диалоге и выводятся после вычисления выражения совместно с их адресами.

№ варианта	Выражение	Тип
0	A*=b>>3|c++-c	Unsigned char, int, short int
1	A^=b++==3?--c:b---c	Unsigned int, int, short int
2	A&=b<<3+~++c	Unsigned int, short int, int
3	A|=--b%c++	Unsigned int, int, short int
4	A%=b+++--c%b	Int, int, short int
5	A/=++b*--c>>3	Int, int, short int
6	A>>=b++%--c	Unsigned char, int, short int
7	A<<=~b*--c	Unsigned char, int, short int
8	A-=((b++>>2!=c)+1)|++c	Int, int, unsigned char
9	A+=b+=c*=b>>2	Long int, int, short int

Самостоятельная работа №10

Цель работы: изучение операторов цикла языка “Си”. Реализовать задачу циклами for, while и do while.

For, while, do, break, continue (зарезервированные слова). Break завершает цикл по некоторому условию в теле, continue продолжает цикл, передавая управление в конец цикла, что позволяет реализовать сложные циклы без использования оператора безусловного перехода goto и дополнительных меток. Тело цикла может состоять из одного или нескольких операторов, в последнем случае группа операторов объединяется в блок с помощью фигурных скобок {}, обычно тело цикла принято сокращённо называть оператором.

№ варианта	Постановка задачи
0	Найти сумму целых чисел от m до n.
1	Найти произведение целых чисел от m до n.
2	Найти сумму чисел от m до n, кратных к.
3	В банк положили N у. е. под P процентов в год. Сколько денег будет в банке через L лет?
4	Найти !n, n<=16.
5	Найти произведение целых чисел от m до n, кратных к.
6	В банк положили N у. е. под P процентов в год. Через сколько лет в банке будет S у. е. ?
7	На сколько дней студенту хватит стипендии S, если он тратит N р. в день и его расходы каждый день увеличиваются на X р. ?
8	Найти сумму четных чисел от m до n.
9	На сколько дней студенту хватит стипендии S, если он тратит N р. в день и его расходы каждый день увеличиваются в X раз?

Оператор цикла While содержит выражение, которое управляет повторением выполнения тела цикла.

Синтаксис:
While (выражение) оператор

Выражение вычисляется до выполнения оператора, и трактуется как строка бит, если строка бит содержит хотя бы одну единицу, то выражение считается истинным и выполняется тело цикла. Для избежания зацикливания переменные входящие в выражение должны изменяться в теле цикла.

Пример:
```
While ((Ch =getchar())!=EOF) name[cnt++]=ch;

While (*string!=’\0’)
{
putchar(*string++);
putchar(‘\n’);
}
```
Оператор цикла do While содержит выражение, которое вычисляется после выполнения тела цикла.
Синтаксис:

Do оператор While (выражение);

Тело цикла выполняется хотя бы раз и повторяется до тех пор пока выражение истинно.

Пример:

```
Do
Printf("%d\n", x);
While(++x<=7);
```

Оператор цикла for является наиболее сложным.

Синтаксис:

For (выражение1;выражение2;выражение3) оператор

Выражение1 и выражение3 могут состоять из нескольких операций, объединённых операцией запятая (перечисленных через запятую), выражение2 задаёт условие цикла и если истина, то выполняется оператор (тело) цикла, иначе завершение цикла.

Выражение1 вычисляется до входа в цикл и описывает инициализацию переменных цикла, выражение3 вычисляется после тела цикла и выполняет модификацию цикла.

Любое из выражений может быть опущено, пустое выражение считается истинным значением условия.

Пример:

```
For(;;); задаёт бесконечный цикл,
For (i=0, j=n-1;j<n;i++, j--) a[i]=a[j];
For (i=0, j=n-1; a[i]=a[j], j<n;i++, j--);
```

Самостоятельная работа №11

Цель работы: изучение операторов принятия решений. Реализовать задачу, используя switch и IF.

If, else, switch, case, default, breck -зарезервированные слова

Условный оператор if применяется для программирования неравновероятных процессов, а оператор-переключатель switch для структуированной записи вычислений.

Оператор switch состоит из выражения (селектора) и списка операторов, отмеченных метками-константами так, как показано ниже. В отличие от других языков программирования метка-константа определяет точку входа в последовательность case и при необходимости выхода из последовательности применяется оператор разрыва break. Тип метки-константы должен совпадать с типом выражения селектора, а в случае несовпадения значения ни с одной меткой-константой управление передаётся на метку с именем default.

№ варианта	Постановка задачи
0	Дан одномерный массив чисел. Умножить на 3 его положительные элементы, отрицательные разделить на 2, а равные нулю – заменить числом 5.
1	Даны отрезки a, b, c. Определить, можно ли из них составить треугольник. Вывести сообщения типа "можно", "нельзя", "треугольник превращается в отрезок".
2	Дан одномерный массив чисел. Если его максимальный элемент >10, найти произведение элементов массива, <=10, но >=5 – найти сумму элементов, <5 – обнулить массив и вывести соответствующее сообщение.

3	Дана точка (x;y) и круг радиуса R с центром в точке (0;0). Определить, где находится точка: внутри, на границе, или за пределами круга.
4	Дана матрица [mxn]. Если число строк больше числа столбцов, удвоить все элементы матрицы и утроить их в обратном случае. Если матрица квадратная, обнулить все ее элементы.
5	Напечатать значение 0<К<10 римскими цифрами.
6	Для целого числа 14<К<26 написать фразу “мне К лет”, учитывая форму числительных (год, года).
7	Вывести название месяца по его номеру.
8	Напечатать словесное название числа 0<К<10
9	Вывести название дня недели по его номеру.

Синтаксис оператора-переключатель:

```
Switch(выражение)
{
case const1: оператор[ы];
...
case constn: оператор[ы];
default: оператор[ы];
}
```

Пример:

```
Switch(ch)
{
case 'A':case 'a': puts(“Буква”); break;
case '0': puts(”Цифра=”);
default: puts(“0”)
}
```

Самостоятельная работа №12

Цель работы: научиться работать с двумерными массивами и вложенными операторами цикла.

Массив применяется для задания однородных данных с одинаковым размером элементов. Описание массива даёт коллективное имя для входящих в него элементов. Для обращения к элементу обычно используется индексация, которая всегда начинается с нуля. Однако иногда удобней обращаться к элементу непосредственно по вычисляемому адресу, в этом случае необходимо помнить, что имя массива есть адрес-константа и, следовательно, ему нельзя присвоить значение.

Если с массивом связать указатель_на_элемент_массива, то с этой переменной можно работать как с обычной скалярной переменной, не забывая о допустимых операциях с адресами.

Синтаксис:

```
Память тип имя_массива [размер1]…[размерn]; /*это массив*/
Память тип * имя_ptr; /*это указатель на тип*/
```

Элементы массива могут иметь любой тип, а тип индексов должен быть целым.

Пример:

```
/* статический одномерный массив из двух элементов типа int*/
Static int w[2];
```

```
/* автоматическая память, двумерный массив элементов символов*/
char m[7][50];
/* автоматический одномерный массив указателей на тип char */
char * mptr[7];
int * wptr; /*скалярный указатель на тип int */.
```

№ варианта	Постановка задачи
0	Составить одномерный массив из максимальных элементов строк матрицы [MxN].
1	Определить число элементов матрицы [MxN] кратных A.
2	Перемножить 2 матрицы [MxN] и [NxK].
3	Вывести все элементы двумерного массива [MxN] кратные A.
4	Найти сумму элементов матрицы [MxN], расположенных ниже побочной диагонали.
5	Найти число отрицательных элементов матрицы [MxN], расположенных в столбцах с номером, кратным A.
6	Найти произведение элементов матрицы [MxN], расположенных выше главной диагонали.
7	Поменять местами элементы j и k столбцов матрицы [MxN].
8	Определить количество элементов матрицы [MxN] больших A и расположенных в четных строках и столбцах.
9	Найти минимальный элемент главной диагонали матрицы [MxN].

Самостоятельная работа №13

Цель работы: изучение комбинированного типа данных.

Запись содержит несколько компонентов, или полей, которые могут иметь различные типы. Обычно такие агрегаты данных в языке Си называются структурами и описывают с помощью ключевого слова struct. Различают структуру-шаблон, которая задаёт схему расположения элементов, и структурную переменную, размещаемую в памяти. Структуры, как и любой другой агрегат данных, можно размещать в массиве.

Синтаксис:

```
Struct имя_шаблона /* Задание расположения элементов структуры */
{
Описания_элементов;
...
Описания_элементов;
};
Struct имя_шаблона имя_переменной; /* Объявление структуры*/
```

Обращение к элементам структуры возможно по составному имени элемента (используется операция точка):

Имя_структуры. имя_элемента_структуры…

Если со структурой связан указатель на структуру, то для доступа к данным можно использовать имя, квалифицированное указателем:

Имя_указателя_на_структуру -> имя_элемента_структуры.

Необходимо помнить что изменение указателя_на_структуру изменяет значение адреса на длину записи структуры.

№ варианта	Постановка задачи
0	Ввести информацию о ценах на процессоры в массив записей и вывести ее в виде таблицы.
1	Ввести имена и телефоны сотрудников организации в массив записей и вывести их в виде таблицы.
2	Ввести информацию о ценах на мониторы в массив записей и вывести ее в виде таблицы.
3	Ввести фамилии и адреса сотрудников организации в массив записей и вывести их в виде таблицы.
4	Ввести характеристики монитора (разрешение, частота вертикальной развертки) в массив записей и вывести их в виде таблицы.
5	Ввести информацию о количестве продукции на складе в массив записей и вывести ее в виде таблицы.
6	Ввести информацию о ценах на жесткие диски в массив записей и вывести ее в виде таблицы.
7	Ввести информацию о ценах на компьютеры в массив записей и вывести ее в виде таблицы.
8	Ввести информацию о ценах на автомобили в массив записей и вывести ее в виде таблицы.
9	Ввести информацию о специальностях сотрудников организации в массив записей и вывести ее в виде таблицы.

Самостоятельная работа №14

Цель работы: Изучение динамического распределения памяти.

В зависимости от модели памяти доступны для управления near-куча, far-куча или только far-куча, что предопределяет использование указателей и функций соответствующего вида (см. \\include\alloc. h).

Широко используется динамическое управление памятью при работе со списковыми структурами.

Замечания:

Не забывайте освобождать занятую вами память, т. к. память, занятая в far-куче, системой не освобождается.

№ варианта	Постановка задачи
0	Ввести информацию о количестве продукции на складе и ценах на нее в массив записей и вывести ее в виде таблицы.
1	Ввести информацию о ценах на жесткие диски, их количестве на складе в массив записей и вывести ее в виде таблицы.
2	Ввести информацию о ценах на компьютеры, их количестве на складе в массив записей и вывести ее в виде таблицы.
3	Ввести информацию о ценах на автомобили, их количестве на складе в массив записей и вывести ее в виде таблицы.
4	Ввести информацию о специальностях сотрудников организации, их зарплате в массив записей и вывести ее в виде таблицы.
5	Ввести информацию о ценах на процессоры, их количестве на складе в массив записей и вывести ее в виде таблицы.
6	Ввести имена и телефоны, адреса сотрудников организации в массив записей и вывести их в виде таблицы.

7	Ввести информацию о ценах на мониторы, их количестве на складе в массив записей и вывести ее в виде таблицы.
8	Ввести фамилии, имена и адреса сотрудников организации в массив записей и вывести их в виде таблицы.
9	Ввести информацию о названии композиций, их исполнителях, продолжительности звучания в массив записей и вывести ее в виде таблицы.

Пример:

```
#include <stdlib. h>
/* Объявление шаблона структуры однонаправленного списка с инициализацией первого элемента*/
 struct CLIENT
 {
 char avto[30]; // Марка автомобиля
 char nom[20]; // Номер автомобиля
 struct CLIENT *p; // Указатель на следующий автомобиль
 } first={"", "", NULL};
struct CLIENT *p_f=NULL;
/*Функция добавления элемента*/
void vAddAvto(void)
{
struct CLIENT *p_t=NULL;
 p_t=&first;
 while (p_t->p!=NULL) p_t=p_t->p; /*Поиск свободного*/
 if ((p_t->p=malloc(sizeof(struct CLIENT)))==NULL)
 {
 printf("Недостаточно памяти для работы программы");
 exit(0);
 }
 printf("\nВведите марку автомобиля:");
 scanf("%s", p_t->avto);
 printf("\nВведите номер автомобиля:");
 scanf("%s", p_t->nom);
}
```

Самостоятельная работа №15

Цель работы: Изучение возможностей команды перекодирования данных и команд работы со строками.

Строковые операции широко применяются в различных приложениях и фактически являются подпрограммами, требующими подготовки регистров в качестве входных параметров.

Основные команды:
Xlat, Movs, Scas

Замечания:
Реализовать ввод текста в диалоге с пользователем.

№ варианта	Постановка задачи

0	Дан текст. Определить, каких букв – гласных или согласных – больше в этом тексте.
1	Дан текст. Вывести все его гласные буквы.
2	Определить количество звонких согласных в тексте.
3	Дан текст. Вывести все его согласные буквы
4	Дан текст. Определить количество гласных букв в нем.
5	Дан текст. Определить количество согласных букв в нем.
6	Определить количество глухих согласных в тексте.
7	Дан текст. Вывести все его звонкие согласные буквы.
8	Дан текст. Вывести все его глухие согласные буквы.
9	Подсчитать число гласных и согласных букв в тексте.

Самостоятельная работа №16

Цель работы: Изучение файлов.

File (зарезервированное слово)

Файловый тип состоит из линейной последовательности компонентов любого типа, кроме файлового.

Синтаксис:

```
FILE * fptr;
```

или

```
int handle;
```

Замечания:

Тип FILE предполагает работу с операциями файлового типа (см. файл \\include\STDIO. h), используя указатель на FCB-блок, а тип int используется для работы через номер в таблице файлов и часто называется низкоуровневым доступом (см. файл \\include\IO. h).

Пример:

```
(* Объявления файловых типов *)
#include <io. h>
…
int handle;
handle=open(“xmp. txt”, O_CREAT|O_TRUNC|O_BINARY, S_IREAD); /* Создаёт двоичный файл для чтения и если он существует, то его длина принимает значение 0 */
#include <stdio. h>
…
FILE * Stream;
Stream=fopen(“xmp. txt”, ”rt”); /* Открывает существующий файл для чтения в текстовом режиме */
```

№ варианта	Постановка задачи
1 – 10	Ввести информацию из Л. Р. №4 в массив записей. Вывести ее из массива записей в набор данных. Прочитать из набора данных и распечатать эту информацию, не используя массив записей. Применить текстовый и двоичный режим обработки.

Самостоятельная работа №17

Цель работы: изучение функций языка С

В языке Си есть только понятие функции, которые расположены в одной области видимости. Функции могут быть сосредоточены в одном программном файле или в нескольких, объединяемых через файл Проект, а также в библиотеках языка Си и личных библиотеках пользователя. Перед обращением к функции необходимо иметь объявление функции полное или частичное. Объявления могут быть вынесены в отдельные файлы пользователя или системные, которые сосредоточены в файлах \\include*. h. Файлы описаний включаются оператором препроцессора #include … обычно в начале программного файла.

Синтаксис объявления:

Возвращаемый_тип идентификатор_функции([необязательный список описаний параметров]);

Одна из функций должна иметь имя main, с неё начинается выполнение программы.

Функция начинается с заголовка функции с указанием типов и имён формальных параметров, а тело функции заканчивается оператором

Return(выражение);

Функцию можно вызвать по имени или по адресу с помощью указателя на функцию совпадающего типа в любом месте выражения.

Пример:

```
(* Объявление и описание функции *)
char* UpCaseStr(char* S) ;
{
 int I ;

 For (I=1;I<= strlen(S);)
 If (S+I>='a') &&(S+I<='Z')
 Dec((S+I), 32);
 Return(S);
 };
```

№ варианта	Постановка задачи
1 - 10	Выполнить задания из Л. Р. №7, с использованием функций.

Самостоятельная работа №18

Цель работы: изучение резидентных программ на языке С

Резидентные программы в системном программировании находят широкое распространение как с целью придания свойств многозадачности однозадачным операционным

усторойствам так и для выполнения специфических действий. Обычно резидентная программа будучи загружена в память остается там до конца сеанса, однако она может быть снята или деактивирована и в любой другой момент либо с помощью собственных средств (пункт меню, горячая клавиша) либо через повторный запуск программы с подходящим ключом. Для того, чтобы повторный запуск не оставлял программу резидентной должны быть предусмотрены средства контроля наличия программы в памяти.

Обычно как средство связи с резидентом используется мультиплексное прерывание Int 2fh, либо Int 2dh при соблюдении стандарта AMIS (альтернативная спецификация мультиплексорного прерывания). Int 2fh использует жёсткие идентификаторы резидентов, а Int 2dh определяет идентификаторы во время загрузки динамически и позволяет увеличить число одновременно загруженных резидентов с 64 до 255 и исключает проблему идентификации.

Int 2dh на входе требует наличия идентификатора в AH, а в AL номер функции для резидента:

0 - проверка наличия в памяти
1 - получить адрес точки входа
2 - деинсталляция (деактивация)
3 - запрос на активизацию (для всплывающих программ)
4 - получить список перехваченных прерываний
5 - получить список перехваченных клавиш
6 - получить информацию о драйвере (для драйверов устройств)
7 - 0Fh – резерв для AMIS
1Fh-0FFh – свои дополнительные функции для каждой программы.

На выходе резидент должен вернуть в AL код 0, если функция не поддерживается. Обязательные функции для реализации: 0, 2, 3, 4, 5.

Стандартное начало обработчика прерываний в стандарте IBM ISP должно выглядеть так:

Hw_reset2D: retf
00h: 0EB10 (jmp short на байт после блока)
02h: dword –адрес предыдущего обработчика (save)
06h: 424Bh – сигнатура ISP блока
08h: 80h, если это первичный обработчик аппаратного прерывания (т. е. он посылает контроллеру прерываний сигнал EOI=20h) или 00h, если это обработчик программного или дополнительный обработчик аппаратного прерывания.
09h: jmp short hw_reset2D (на начало подпрограммы аппаратного сброса, обычно состоит из одной команды iret или retf)
0Bh: 7 байт – резерв ISP.

При установке инсталляционная часть резидента олжна проверить, нет ли её копии, просканировав все идентификаторы от 00h до FFh, и если нет, установить свой на первый свободный идентификатор мультиплексного прерывания.

В конце резидента обычно перед секцией инициализации включаются требуемые AMIS таблицы :

```
; amis сигнатура для резидентной программы
amis_sign db '8 байт – имя автора'
          db '8байт – имя программы'
          ; далее комментарий <=64 байт
          db 'comment…', 0; ASCIZ
; amis список перехваченных прерываний
amis_hooklist db 09 h; номер прерывания
              dw offset int09h_handler
              ; перечень всех обработчиков
              ; последним обработчик int 2Dh
              db 2Dh
              dw offset int2Dh_handler
; amis список горячих клавиш
amis_hotkeys  db 01h ; проверка после стандартного обработчика
              db 1 ; число клавиш
              db 1Eh ; скан_код клавиши 'A'
              dw 08h ; требуемые флаги (любая Alt)
              dw 0 ; запрещенные флажки
              db 1 ; клавиша проглатывается.
```

Как правило, процесс включения резидента не вызывает особых трудностей. При программировании на ассемблере используются прерывания

int 27, в dx адрес первого свободного байта,

int 31, в dx размер программы в параграфах,

а при программировании на C используют функцию tsr(0, size_paragraf).

При реализации резидентных частей требуется учитывать много рабочих моментов. Нужно всегда помнить, что передача управления происходит из 'чужой ' программы и, следовательно, при необходимости нужно сохранить 'чужую среду' и переключиться на 'свою среду', в первую очередь это касается регистров, стека, PSP, при возврате управления по iret 'чужая' среда должна быть предварительно восстановлена. Эти же моменты нужно учитывать и при 'выгрузке' резидента, если восстановить среду полностью не удается, например, нельзя восстановить векторы прерывания, то вместо выгрузки из памяти нужно деактивизировать резидент. Собственно выгрузка резидента проводится по стандартной схеме.

```
; выгрузка резидента
mov ah, 51h
int 21h; сегментный адрес PSP прерванного процесса в bx
mov word ptr cs:[16h], bx; адрес PSP прерванного процесса в наше PSP
; восстановление адреса возврата из стека
pop dx; сегмент
pop bx; смещение
mov word ptr cs:[0Ah], bx
mov word ptr cs:[0Ch], dx; адрес возврата в предок
puch cs
pop bx
mov ah, 50h
```

int21h; установить текущий PSP
mov ax, 4CFFh
int 21h; завершить программу с возвратом в предок.

Обычно завершение программы выполняется через запуск этой же программы с дополнительным ключом, однако можно предесмотреть завершение и из самого резидента либо через горячую клавишу либо через меню.

Один из возможных вариантов завершения программы непосредственная запись кода ‘свободен’ в MCB –блок.

```
push ds
mov ax, cs ; вычислить адрес MCB
dec ax
mov ds, ax
mov word ptr ds:[1], 0 ; Объявляем MCB как не занятый
pop ds
;….
iret
```

При программировании на языке C (CPP) нужно учесть следующее.

Функции_обработчики прерываний должны быть объявлены как

```
void interrupt имя()
```

Если требуется возврат значений в регистрах, то их нужно менять в стеке, например так, как показано ниже.

```
/*Файл int24err. cpp*/
/*-----------------------------------------------
 Заглушка прерывания 24h -критическая ошибка DOS
-----------------------------------------------*/
#ifdef __cplusplus
#define __CPPARGS...
#else
#define __CPPARGS
#endif
/*
void interrupt ISR_int24(unsigned bp, unsigned di,
 unsigned si, unsigned ds,
 unsigned es, unsigned dx,
 unsigned cx, unsigned bx,
 unsigned out_ax,
 unsigned ip, unsigned cs,
 unsigned out_flags )
 */
void interrupt ISR_int24(__CPPARGS)
{
 // out_ax=3; //3 - Fail, 2 - Abort, 1 - Retry
 asm mov word ptr [bp+16h], 3;
}
```

Доступ к данным, расположенным в кодовом сегменте, осуществляется с помощью регистра CS, например через указатель на структуру, где описаны все данные, так

```
struct VIDMEM _cs register * pmem ;
pmem=(struct VIDMEM _cs *)dvidmem;
```

где dvidmem глобальное имя в секции кода (например, подпрограмма на ассемблере с описаниями_данных структуры).

Описание структуры приведено ниже.

```
/* Файл vidmem. h */
#define MEMCS
#ifdef __cplusplus
#define __CPPARGS...
#else
#define __CPPARGS
#endif
#ifdef MEMCS
#define cstods() asm push ds;\
 _DS=_CS;
#define poptods() asm pop ds
struct VIDMEM
{/*Память для векторов прерываний*/
void interrupt (far * old_int5)(__CPPARGS);
void interrupt (far * old_int9)(__CPPARGS);
void interrupt (far * old_int13)(__CPPARGS);
void interrupt (far * old_int1C)(__CPPARGS);
void interrupt (far * old_int24)(__CPPARGS);
void interrupt (far * old_int28)(__CPPARGS);
void interrupt (far * old_int2F)(__CPPARGS);
/*Стек для TSR*/
unsigned TSR_sp, TSR_ss, TSR_screen_sp, TSR_screen_ss;
unsigned TSR_stack[512], TSR_screen_stack[512];
unsigned old_sp, old_ss, old_screen_sp, old_screen_ss;
/*PID*/
unsigned old_PSP;
/*DTA*/
char far * old_DTA;
char far * TC_DTA;
/*Адрес флага повторной входимости в DOS*/
char far * inside_DOS_ptr;
/*Флаги собственной TSR*/
unsigned inside_BIOS, busy, request, unload, inside;
int old_cbrk;
unsigned str0, str1, col0, col1;
unsigned vid_memory, start_adr, max_str, max_stolb;
char ftitle[21];//="Имя файла назначения";
char fkey[24];//="Esc <Cancel> Enter <Ok>";
char style[8];//={'\xC9', '\xCD', '\xBB', '\xBA', '\xBA', '\xC8', '\xCD', '\xBC'};
char filename[40], data[4100];
unsigned buf[120];
};
extern "C" void dvidmem(void);
#endif
```

Доступ к глобальным данным возможен через CS в модели tyni.

Варианты заданий

№ варианта	Прерывание	Функция/Горяч. клавиша
0	09h	<alt/w>
1	05h	Print Screen
2	13h	Обращение к дисководу
3	1Ch	Timer
4	21h	39h – Mkdir
5	21h	3Ah – Rmdir
6	21h	3Bh – Chdir
7	21h	30h – ver
8	21h	2Ah – date
9	21h	2Ch – time

Экономия памяти получается за счет подходящего расположения перекрываемых данными модулей, используемых при инициализации.

Заметим, что порядок включения модулей определяется их положением в файле проекта имя. prj, а необходимые данные для освобождения памяти можно определить из файла имя. map

В самостоятельной работе следует реализовать один из обработчиков с обозначением его присутствия выводом контрольного сообщения на определенном месте экрана. Завершение программы по <имя> /u.