Лабораторная работа дополнительная. СПОСОБЫ ПРЕДСТАВЛЕНИЯ В ПАМЯТИ И ОБРАБОТКИ СЛОЖНЫХ СТРУКТУР ДАННЫХ

Эффективность разрабатываемой программы во многом зависит от выбранных структур данных, поэтому важно сначала продумать какого типа переменные применить в программе, а это автоматически определяет способ их обработки. Среди сложных структур данных, размещаемых в оперативной памяти, выделяют: множества, массивы, записи, динамические и комбинированные типы данных.

Множества рассматриваются как последовательность битов, каждый бит которой показывает, принадлежит элемент с данным порядковым номером множеству или нет. В Turbo Pascal максимальное число элементов множества – 256, однако реальное количество элементов, под которые выделяется память, определяется из объявления. Число байт, занимаемых множеством, вычисляется по формуле

ByteSize = (Max div 8)-(Min div 8)+1,

где Min, Max – нижняя и верхняя границы базового типа множества.

Номер байта размещения конкретного элемента множества E вычисляется по формуле:

ByteNumber = (E div 8)-(Min div 8),

номер бита внутри этого байта:

BitNumber = E mod 8.

Элемент множества может быть любого перечисляемого типа.

Примеры

1. `var S1 : set of 2..7;`
   `S1 := [];`

```
 - - 7 6 5 4 3 2
|- - 0 0 0 0 0 0|
          +0
```

Min = 2, Max = 7, 1 байт

`var S2 : set of One..Ten;`
`S2 := [two, five..ten];`

```
 - - - - - - ten    five   one
|- - - - - - 1 1|1 1 1 1 0 0 1 0|
          +1            +0
```

Min = 1, Max = 10, 2 байта.

Массив хранится в виде непрерывной последовательности переменных. Для многомерного массива каждая из этих переменных имеет тип массив, причем элементы с наименьшими индексами находятся в младших адресах памяти. Многомерный массив представляется так, что правый индекс возрастает быстрее. Элементом массива может быть переменная любого типа, кроме файлового, а максимальный размер массива в Turbo Pascal 65520 байт (64 Кбайт без одного параграфа).

2. var M1: array[2..7] of byte;

```
  2  3  4  5  6  7
|*|*|*|*|*|*|
 +0 ...        +5
```

Элемент – байт, элементов – 6.

M2: array [boolean] of
array[-1..1] of integer;

```
  -1   0    1   -1   0    1
|**|**|**‖**|**|**|
+0 false      +6 true  +10
```

Элемент – слово, элементов – 6.

Как видно из примеров, развертка многомерной структуры в линейную последовательность приводит к заметному усложнению способа определения адреса индексируемого элемента по мере возрастания размерности массива.

Запись хранится как непрерывная последовательность переменных, соответствующих ее полям. Первое поле записи располагается в младших адресах памяти. Если в записи имеются вариантные части, то переменные каждого варианта считаются размещенными в памяти с одного и того же адреса.

3. var
R1:record
i: integer;
c1, c2, c3: char
end;

```
  i   c1  c2 c3
|* *|*|*|*|
+0+1 +2 +3 +4
```

адреса полей записи:
R1.i = baseR1+0;
R1.c1 = baseR1+2;
R1.c2 = baseR1+3;
R1.c3 = baseR1+4.

var
R2: record case boolean of
true: (i:integer);
false: (c1,c2,c3:char)
end;

```
  i
|c1c2|c3
|*|*|*|
+0 +1 +2
```

адреса полей записи:
R2.i = baseR2+0;
R2.c1 = baseR2+0;
R2.c2 = baseR2+1;
R2.c3 = baseR2+2.

Работа со структурами данных типа запись основывается на применении адресации с индексированием: известно количество байтов занимаемых каждым полем записи и их взаимное расположение в памяти, следовательно, доступ к конкретному полю выполняется прибавлением смещения поля к базовому адресу записи, как это показано выше. Ассемблер имеет удобные средства для работы с подобными типами данных, псевдокоманды TRUC и RECORD. Покажем, как в нашем случае применить псевдокоманду STRUC.

Описание структуры (STRUC) оформляется примерно так же, как и определение сегмента данных, поэтому условно каждую структуру можно считать еще одним сегментом данных в программе. Разница, однако, в том, что само описание STRUC не создает никаких данных. Для этого нужно вызвать структуру по имени (подобно макрообращению) в требуемом месте программы:

```
structR1  struc; описание структуры StructR1
i         dw  -1; поля записи: i (инициализировано)
c1        db   ? ;            c1 (неинициализировано)
c2        db  '0';            c2
c3        db   ? ;            c3
structR1  ends; конец описания структуры StructR1
```

Вызов структуры для создания записи

```
R1  structR1 <,'a','R',>
```

Поля, изменения которых не требуется, пропускаются, но запятые остаются. Здесь будет: i = -1, c1 = 'a', c2 = 'R', c3 = не определено.

Варианты обращения к полям структуры:

```
add  R1.i,7; прямая адресация с индексированием
lea  bx,R1 ; bx:=адрес записи R1
mov  al,[bx].c2; косвенная адресация
               ; с индексированием
```

Программа-ассемблер по выражениям вида R1.i или [bx].c2 автоматически вычисляет необходимое смещение поля записи и заносит его в код команды. Создавая описание структуры, сле-

дует учитывать, что при вызове структуры макроассемблер не разрешает изменять поля, содержащие определение более чем одного компонента. Например, поле вида c1c2 db '1', '2' переопределить нельзя, а поле c1c2 db '12' можно, так как строка рассматривается как один компонент. Несмотря на удобство псевдокоманды STRUC, записи с вариантами приходится обрабатывать обычным способом.

Динамический тип данных, по сути, сводится к использованию косвенной адресации для переменной любого типа, и в этом смысле все кажется очень простым. Например, приведенное ниже объявление задает переменную, являющуюся ссылкой (адресом) на динамически выделяемое слово памяти:

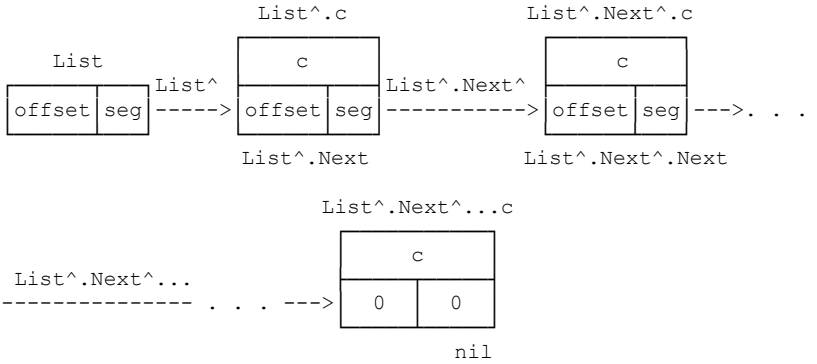
```
var WordPtr:^word;      указатель     ссылка     переменная
. . .                   WordPrt^       -->       | * | * |
new(WordPrt);                                     +0   +1
. . .
```

Тем не менее работа с таким типом данных на уровне ассемблера вызывает трудности из-за сложностей организации косвенных ссылок и, самое главное, необходимости строить специальный механизм управления динамической памятью [4]. Здесь будет рассмотрена только организация косвенных ссылок.

Предположим, требуется написать подпрограмму вывода на терминал списка List, по смыслу подобную процедуре PrintList:

```
type ListPrt=^ListNode;
     ListNode=record
                c: char;
                Next: ListPrt
                end
var  List: ListPrt;
procedure PrintList(List:ListPrt);
begin
      while List<>nil do
        begin
        write( List^.c);
        List:=List^.Next
        end;
end { PrintList } ;
```

Схематично представим расположение данных в памяти:



Из рисунка понятен процесс перемещения ссылки по списку: загрузить указатель List в регистры, например ds:si; ссылаясь через загруженный указатель, выполнить необходимую операцию с текущей вершиной, после чего загрузить ее поле Next в те же регистры. Процедура "обработка-загрузка" выполняется до появления ссылки nil. Текст подпрограммы на ассемблере:

```
; procedure PrintList(List: ListPrt);
;   выводит на экран список List.
PrintList proc  near
List  equ   dword ptr[bp+4]; адрес параметра List
      push  bp    ; сохранение bp
      mov   bp,sp ; настройка sp на вершину стека
      push  ds    ; сохранение ds
      lds   si,List; ds:si:=первая вершина
                  ;  списка (List^)
While: mov   ax,ds ; реализация "while List<>nil do"
       or    ax,si ; ds:si=nil ?
       jz    Exit  ; да, выход
       lodsb       ; нет, читать в al очередной
                   ;  List^.c
       mov   ah,2  ; реализация "write(List^.c);"
       mov   dl,al ; вывод al на экран через
       int   21h   ;  функцию DOS
       lds   si,dword ptr ds:[si]; ds:si:=адрес
                                  ; из List^.Next
       jmp   While  ; "while ...  end"
```

```
Exit:  pop   ds      ; восстановить ds
       pop   bp      ; восстановить bp
       ret   4       ; выход с удалением List
                     ;  из стека
PrintList endp
```

Аналогично можно рассуждать и над более сложными динамическими структурами. Без пояснений приведем текст рекурсивной подпрограммы вывода на экран бинарного дерева при инфиксном обходе:

```
type TreePtr=^TreeNode;
     TreeNode=record
                c:char;
                Left,
                Right:TreePtr
                end;
var  Tree:TreePtr;
procedure PrintTree(Tree:TreePtr);
begin
   if Tree<>nil then
       begin
       PrintTree(Tree^.Left);
       write(Tree^.c);
       PrintTree(Tree^.Right)
       end
end { PrintTree } ;

; procedure PrintTree(Tree:TreePtr);
;  выводит на экран двоичное дерево Tree
; в инфиксой форме

TreeNode  struc ; описание структуры вершины дерева
c         db   ?         ; место для символа
Left      dd   0         ; =nil
Right     dd   0         ; =nil
TreeNode  ends

PrintTree proc near
Tree    equ  dword ptr[bp+4] ; адрес параметра Tree
        push bp        ; сохранение bp
        mov  bp,sp  ; настройка bp на вершину стека
        push ds          ; сохранение ds
```

```
        lds  bx,Tree      ; ds:bx:=Tree^
        mov  ax,ds        ; "if Tree<>nil then "
        or   ax,bx        ; ds:bx=nil ?
        jz   Exit         ; да, выход
; подготовка  вызова  "PrintTree(Tree^.List);".
; Здесь  нужно сохранить offset Tree^ из bx ,
; т.к. вызов может его испортить (ds всегда
; сохраняется)
        push bx           ; сохранение offset Tree^
; загрузка в стек seg/offset указателя Tree^.Left
        push word ptr[bx+2].Left t
        push word ptr[bx].Left;
        call PrintTree    ; "PrintTree(Tree^.Left);"
; реализация "write(Tree^.c);"
        pop  bx ; восстановить offset Tree^ текущий
        mov  dl,[bx].c; вывод Tree^.c из dl
        mov  ah,2      ;  средствами DOS
        int  21h       ;  на экран
; подготовка вызова "PrintTree(Tree^.Right)",
; сохранения offset Tree^ не требуется
; загрузка в стек seg/offset указателя Tree^.Right
        push word ptr[bx+2].Right
        push word ptr[bx].Right
        call PrintTree; "PrintTree(Tree^.Right)"
Exit:   pop  ds        ; восстановить ds
        pop  bp        ; восстановить bp
        ret  4  ; выход с удалением Tree из стека
PrintTree endp

SSEG    segment para stack 'STACK'
        db      64 dup('STACKSEG')
SSEG    ends

DSEG      segment para 'DATA'    ; описание
                                 ; двоичного дерева
NodeA     TreeNode <'A',,>; листья: A (ссылки=nil)
NodeC     TreeNode <'C',,>; листья: C (ссылки=nil)
NodeE     TreeNode <'E',,>; листья: E (ссылки=nil)
NodeG     TreeNode <'G',,>; листья: G (ссылки=nil)
NodeB     TreeNode <'B',NodeA,NodeC>; ветви: B
                                   ; (ссылки=адреса)
NodeF     TreeNode <'F',NodeE,NodeG> ; ветви: F
```

```
                                    ; (ссылки=адреса)
NodeD     TreeNode <'D',NodeB,NodeF> ; корень: D
TreeOfs   dw    offset NodeD     ; =начальный адрес
DSEG      ends

include   print.lib

CSEG      segment para'CODE'
          assume  cs:CSEG, ds:DSEG, ss:SSEG
UseTree   proc  far
          mov   ax,seg DSEG; настройка ds на
          mov   dx,ax       ; сегмент данных DSEG
          push  ds          ; подготовка вызова
PrintTree:
          push  ds:TreeOfs ; seg/offset NodeD в стек
          call  PrintTree  ; печать дерева
          mov   ax,4c00h   ; завершение программы
          int   21h        ; код завершения 0
UseTree   endp
CSEG      ends
          end   UseTree
```

Принципы работы со структурами данных, являющихся комбинацией из рассмотренных выше структур, также строятся путем объединения механизмов обработки данных составляющих структуры. Если необходимо реализовать над структурой данных операцию поиска или сортировки, можно применить обычные алгоритмы, приведенные в любом учебнике по программированию, например [2, 4]. Имеющаяся аналогия между рассмотренными типами данных и данными языков программирования высокого уровня помогает сделать это с минимальными затратами сил.

Задание к лабораторной работе

Составить подпрограмму, реализующую действия, соответствующие варианту задания, приведенному ниже. Номер варианта выбирается по порядковому номеру студента в списке группы. Для отладки подпрограммы составляется основная программа, включающая определение структур данных и средства контрольного вывода результатов на экран. Числа и множества можно выводить в шестнадцатеричном виде.

1. Для списка List, содержащего сведения о книгах в библиотеке, вывести на экран названия всех книг заданного автора Author, изданных между годами Year1 и Year2:

```
type ListPtr=^ListNode;
ListNode=record
     Author: string[30];
BookName: string;
     Year: word;
     Next: ListPtr
     end;
var  List: ListPtr;
Author: string[30];
Year1, Year2: word;
```

2. В списке из задания 1 удалить вершины, в которых поля Author и BookName пусты либо значение поля Year находится вне диапазона задаваемого Year1 и Year2. Управление динамической памятью не учитывать.

3. В массиве, содержащем список телефонов Phone длиной Len, упорядочить записи по возрастанию номеров Number:

```
var  Phone: array[1..10] of record
                            Name: string;
                            Number: longint
                            end;
     Len:byte.
```

4. В массиве из задания 3 удалить записи, имеющие поле Number <= 0, либо пустое поле Name, откорректировав длину списка Len.

5. По символьной матрице с максимальным размером 10×10 получить транспонированную матрицу, поменяв местами строки и столбцы. Текущий размер матрицы находится в переменной N.

6. Дано произвольное множество A и переменная N, содержащая номер элемента множества. Написать подпрограммы, выполняющие: исключение N из A – procedure Off(var A; N: byte); включение N в A – procedure On(var A; N: byte); проверку вхождения N в A (возвращает true, если входит) – function Test(var A; N: byte): boolean. Если N превышает максимально

допустимое значение, то первые две процедуры ничего не должны выполнять, а функция возвращает false.

7. Для произвольных, но одинаковых по размеру множеств A, B с объявленным числом элементов M составить подпрограммы, выполняющие подсчет реального числа элементов (мощность множества) – function Card(var A; M: byte): byte; проверку включения множества A в множество B (возвращает true, если входит) – function SubSet(var A, B; M: byte): boolean; операции объединения (s=0), разности (s=3) множеств A, B с результатом в B – procedure Operate(var A, B; M, S: byte). Ошибка превышения M предельного значения не контролируется.

8. Вывести на экран в алфавитном порядке все русские буквы из массива Letters: array['a'..'h'] of string, входящие в него по одному разу. Индекс последней строки массива находится в переменной N: char.

9. В символьной матрице 20×20, первоначально содержащей пробелы, заполнить позиции, индексы i,j которых входят в множества SetI(строки) и SetJ(столбцы) типа set of 0..19. Код символа-наполнителя связан с индексами выражением Sym=i+j+32.

10. Для списка List из задания 1 написать подпрограмму реверсирования, переставляющую первую вершину с последней, вторую – с предпоследней и т.д.

11. В списке List из задания 1 упорядочить записи по возрастанию года издания Year.

12. Написать подпрограмму function SubSet (List, List1: ListPtr): boolean, возвращающую true, если список List (задание 1) включает все вершины списка List1.

13. Упорядочить по убыванию числа в столбцах массива A: array[-4..5, '0'..'7'] of longint. Текущие границы массива находятся в переменных Imax, Jmax.

14. Написать подпрограмму подстановки в список List (задание 1) вершин из списка List1, отсутствующих в List.

15. В упорядоченный по возрастанию номеров Number список Phone (задание 3) включить все отсутствующие в нем записи из неупорядоченного списка Phone1 длиной Len1, сохранив упорядоченность и соответственно изменив Len.

16. Для списка List из задания 1 осуществить поиск и вывод на экран сведений о книгах с годом издания Year1, имя автора

которых включает подстроку Substr типа string в произвольном месте поля Author.

17. Написать подпрограмму вывода на экран сведений из списка Phone (задание 3), в которых поле Number лежит в пределах определяемых переменными Number1, Number2, а поле Name не содержит букв из множества Letters: set of char.

18. Написать подпрограмму определяющую, является ли целочисленная матрица с максимальным размером 10×10 "магическим квадратом", суммы всех строк и столбцов которого одинаковы. Текущий размер матрицы задается переменной N.

19. В списке List из задания 1 удалить одинаковые вершины, не учитывая управление динамической памятью.

20. Определить, есть ли в произвольно упорядоченных списках Phone и Phone1 (задание 3), равной длины Len, одинаковые записи и вывести такие записи на экран вместе с индексами.

Порядок выполнения работы

1. Изучить основные сведения по работе, при необходимости просмотреть основные сведения по работе 3 и литературу по программированию на языке Паскаль.

2. Проанализировать варианты решения задачи и составить блок-схему или Паскаль-программу, реализующую задание. Составить подпрограмму и основную программу в соответствии с алгоритмом задания. Подготовить тестовые примеры для отладки программы.

3. Выполнить ввод, трансляцию, построение кода программы и получить результаты ее выполнения на тестовых примерах.

Содержание отчета

1. Цель работы.

2. Текст задания, схема реализации, блок-схема или программа на языке высокого уровня.

3. Тексты программы и подпрограммы.

4. Результаты работы программы на тестовых примерах.

5. Выводы по работе.

Контрольные вопросы

1. Каковы принципы работы с типом данных множество на языке ассемблера?

2. На чем основана работа с типом данных массив на языке ассемблера?

3. Каким образом обрабатывать переменные типа запись на языке ассемблера?

4. Какие сложности возникают при управлении динамической памятью?

5. Каков общий механизм работы со сложной структурой данных, являющейся комбинацией более простых типов?

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Брэй Б. Микропроцессоры Intel: 8086/8088…80486, Pentium: пер. с англ. СПб: BHV-Петербург, 2005. 1328 с
2. Вирт Н. Алгоритмы и структуры данных: пер. с англ. М.: ДМК-Пресс, 2011. 272 с.
3. Зубков С.В. Ассемблер для DOS, Windows и UNIX. М.: ДМК-Пресс, 2013. 638 с.
4. Кнут Д. Искусство программирования в 3 т. Т1. Основные алгоритмы: пер. с англ. М.: Вильямс, 2010. 720 с.
5. Юров В.И. Assembler: учебник для вузов. СПб.: Питер, 2011. 640 с.
6. Юров В.И. Assembler: практикум: учеб. Пособие для вузов. СПб.: Питер, 2007. 400 с.
7. Андреева А.А. Основы программирования персонального компьютера на языке ассемблера: лабораторный практикум. Чебоксары. Изд-во Чуваш. ун-та, 2013. 84 с.