1) **procedure AddElem(var stek1:List;znach1:TInf)** - процедура добавление элемента в стек.  
2) **procedure Print(stek1:List)** - вывод стека.  
3) **Procedure FreeStek(stek1:List)** - освобождение памяти использованного под стек.  
4) **Function SearchElemZnach(stek1:List;znach1:TInf):List** - поиск в стеке по значению, функция возвращает адрес найденного элемента.  
5) **Procedure DelElem(var stek1:List;tmp:List)** - процедура удаления из стека элемента с адресом **tmp**.  
6) **procedure DelElemZnach(var Stek1:List;znach1:TInf)** - удаление из стека элемента со значением **znach1**.  
7) **Procedure DelElemPos(var stek1:List;posi:integer)** - удаление из стека элемента с порядковым номером **posi**.  
8) **procedure SortBublInf(nach:list)** - сортировка стека "пузырьком" (самый простой вариант), с обменом данными между элементами.  
9) **procedure SortBublLink(nach:List)**- сортировка стека "пузырьком" (самый простой вариант), с изменением лишь указателей на элементы.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289 | **Program** Stek;  **uses**    crt; *{Для использования readkey и clrscr}*  **type**    Tinf=**integer**; *{тип данных, который будет храниться в элементе стека}*    List=^TList;  *{Указатель на элемент типа TList}*    TList=**record** *{А это наименование нашего типа "запись" обычно динамические структуры описываются через запись}*      data:TInf;  *{данные, хранимые в элементе}*      next:List;   *{указатель на следующий элемент}*  **end**;    *{Процедура добавляющая элемент в стек}*  **procedure** AddElem(**var** stek1:List;znach1:TInf);  **var**    tmp:List;  **begin**    GetMem(tmp,sizeof(TList)); *{выделяем в памяти место для нового элемента}*    tmp^.next:=stek1;  *{указатель на следующий элемент "направляем" на вершину стека}*    tmp^.data:=znach1; *{добавляем к элементу данные}*    stek1:=tmp; *{вершина стека изменилась, надо перенести и указатели на неё}*  **end**;    *{Процедура вывода стека}*  **procedure** Print(stek1:List);  **begin**  **if** stek1=**nil** **then** *{проверка на пустоту стека}*  **begin**      writeln('‘вҐЄ Їгбв.');      exit;  **end**;  **while** stek1<>nil **do** *{пока указатель stek1 не станет указывать в пустоту}*  **begin**   *{а это произойдёт как только он перейдёт по ссылке последнего элемента}*      Write(stek1^.data, ' '); *{выводить данне}*      stek1:=stek1^.next  *{и переносить указатель вглубь по стеку}*  **end**;  **end**;    *{Процедура освобождения памяти занятой стеком}*  **Procedure** FreeStek(stek1:List);  **var**    tmp:List;  **begin**  **while** stek1<>nil **do** *{пока stek1 не станет указывать в "пустоту" делать}*  **begin**      tmp:=stek1; *{указатель tmp направим на вершину стека}*      stek1:=stek1^.next; *{вершину стека перенесём на следующий за данной вершиной элемент}*      FreeMem(tmp,SizeOf(Tlist)); *{освободим память занятую под старую вершину}*  **end**;  **end**;    *{Поиск элемента в стеке по значению}*  **Function** SearchElemZnach(stek1:List;znach1:TInf):List;  **begin**  **if** stek1<>nil **then** *{если стек не пуст, то}*  **while** (Stek1<>nil) and (znach1<>stek1^.data) **do** *{пока stek1 не укажет в "пустоту" или пока мы не нашли нужный нам элемент}*        stek1:=stek1^.next; *{переносить указатель}*    SearchElemZnach:=stek1;*{функция возвращает указатель на найденный элемент}*  **end**;         *{в случае если элемент не найден, она вернёт nil}*    *{Процедура удаления элемента по указателю}*  **Procedure** DelElem(**var** stek1:List;tmp:List);  **var**    tmpi:List;  **begin**  **if** (stek1=**nil**) **or** (tmp=**nil**) **then** *{если стек пуст или указатель никуда не указывает, то выходим}*      exit;  **if** tmp=stek1 **then** *{если мы удаляем элемент который является вершиной стека, то}*  **begin**      stek1:=tmp^.next;*{следует перенести вершину и}*      FreeMem(tmp,SizeOf(TList)); *{высвободить память из под элемента}*  **end**  **else** *{в случае, если удаляемый элемент не вершина стека, то}*  **begin**      tmpi:=stek1; *{ставим указатель на вершину стека}*  **while** tmpi^.next<>tmp **do** *{доходим до элемента стоящего "перед" тем, который нам следует удалить}*        tmpi:=tmpi^.next;      tmpi^.next:=tmp^.next; *{указатель элемента переносим на следующий элемент за удаляемым}*      FreeMem(tmp,sizeof(TList)); *{удаляем элемент}*  **end**;  **end**;    *{Процедура удаления элемента по значению}*  **procedure** DelElemZnach(**var** Stek1:List;znach1:TInf);  **var**    tmp:List;  **begin**  **if** Stek1=**nil** **then** *{Если стек пуст, то выводим сообщение и выходим}*  **begin**      Writeln('‘вҐЄ Їгбв');      exit;  **end**;    tmp:=SearchElemZnach(stek1,znach1); *{tmp указывает на удаляемый элемент}*  **if** tmp=**nil** **then** *{если элемент не был найден, то выводим сообщение и выходим}*  **begin**      writeln('ќ«Ґ¬Ґ\*в б ЁбЄ®¬л¬ §\*\*зҐ\*ЁҐ¬ ' ,znach1, ' ®вбгвбвўгҐв ў бвҐЄҐ.');      exit;  **end**;    DelElem(stek1,tmp); *{удаляем элемент из стека }*    Writeln('ќ«Ґ¬Ґ\*в г¤\*«с\*.'); *{сообщаем о выполнении действия}*  **end**;    *{Удаление элемента по порядковому номеру (вершина имеет номер 1)}*  **Procedure** DelElemPos(**var** stek1:List;posi:**integer**);  **var**    i:**integer**;    tmp:List;  **begin**  **if** posi<1 **then** *{проверка на ввод информации}*      exit;  **if** stek1=**nil** **then** *{если стек пуст}*  **begin**      Write('‘вҐЄ Їгбв');      exit  **end**;    i:=1; *{будет считать позиции}*    tmp:=stek1;  **while** (tmp<>nil) and (i<>posi) **do** *{пока tmp не укажет в "пустоту" или мы не найдём искомый элемент}*  **begin**      tmp:=tmp^.next; *{переходим на следующий элемент}*      inc(i)   *{увеличиваем значение счётчика}*  **end**;  **if** tmp=**nil** **then** *{если элемента нет выводим соответствующие сообщения и выходим}*  **begin**      Writeln('ќ«Ґ¬Ґ\*в\* б Ї®ап¤Є®ўл¬ \*®¬Ґа®¬ ' ,posi, ' \*Ґв ў бвҐЄҐ.');      writeln('‚ бвҐЄҐ ' ,i-1, ' н«Ґ¬Ґ\*в\*(®ў).');      exit  **end**;    DelElem(stek1,tmp); *{если мы не вышли, то элемент есть и его следует удалить}*    Writeln('ќ«Ґ¬Ґ\*в г¤\*«с\*.'); *{сообщаем о выполнении действия}*  **end**;    *{Процедура сортировки "пузырьком" с изменением только данных}*  **procedure** SortBublInf(nach:list);  **var**    tmp,rab:List;    tmps:Tinf;  **begin**    GetMem(tmp,SizeOf(Tlist)); *{выделяем память для рабочего "буфера" обмена}*    rab:=nach; *{рабочая ссылка, становимся на вершину стека}*  **while** rab<>nil **do** *{пока мы не дойдём до конца стека делать}*  **begin**      tmp:=rab^.next; *{перейдём на следующий элемент}*  **while** tmp<>nil **do** *{пока не конец стека делать}*  **begin**  **if** tmp^.data<rab^.data **then** *{проверяем следует ли менять элементы}*  **begin**          tmps:=tmp^.data; *{стандартная замена в 3 операции}*          tmp^.data:=rab^.data;          rab^.data:=tmps  **end**;        tmp:=tmp^.next *{переход к следующему элементу}*  **end**;      rab:=rab^.next *{переход к следующему элементу}*  **end**  **end**;    *{Процедура сортировки "пузырьком" с изменением только адресов}*  **procedure** SortBublLink(nach:List);  **var**    tmp,pered,pered1,pocle,rab:List; *{все рабочие ссылки}*  **begin**    rab:=nach; *{становимся на вершину стека}*  **while** rab<>nil **do***{пока не конец стека делать}*  **begin**      tmp:=rab^.next; *{переходим к следующему за сортируемым элементу}*  **while** tmp<>nil **do** *{пока не конец стека делать}*  **begin**  **if** tmp^.data<rab^.data **then** *{если следует произвести замену, то}*  **begin**          pered:=nach; *{становимся в вершину стека}*          pered1:=nach; *{становимся в вершину стека}*  **if** rab<>nach **then** *{если мы не стоим на изменяемом элементе, то}*  **while** pered^.next<>rab **do** pered:=pered^.next; *{станем на элементе перед изменяемым}*  **while** pered1^.next<>tmp **do** pered1:=pered1^.next; *{станем на элементе перед изменяемым, который находится за*  *первым изменяемым}*          pocle:=tmp^.next; *{запоминаем адрес элемента после второго изменяемого}*  **if** rab^.next=tmp **then** *{если элементы "соседи", то}*  **begin**            tmp^.next:=rab; *{меняем ссылки, тут если не понятно рисуйте на листочке}*            rab^.next:=pocle  **end**  **else** *{в случае если элементы не соседи, то}*  **begin**            tmp^.next:=rab^.next;*{меняем ссылки, тут если не понятно рисуйте на листочке}*            rab^.next:=pocle;  **end**;  **if** pered1<>rab **then***{советую просмотреть на листочке}*            pered1^.next:=rab;  **if** rab<>nach **then***{советую просмотреть на листочке}*            pered^.next:=tmp  **else***{всё советую просмотреть на листочке}*            nach:=tmp;          pered1:=tmp;*{советую просмотреть на листочке}*          tmp:=rab;*{советую просмотреть на листочке}*          rab:=pered1;*{советую просмотреть на листочке}*  **end**;        tmp:=tmp^.next; *{переходим на следующий элемент}*  **end**;      rab:=rab^.next;*{переходим на следующий элемент}*  **end**;  **end**;    **var**    Stk, *{переменная, которая всегда будет указывать на "вершину" стека}*    tmpl:List; *{рабочая переменная}*    znach:Tinf; *{данные вводимые пользователем}*    ch:**char**; *{для работы менюшки}*  **begin**    Stk:=**nil**;  **repeat** *{цикл для нашего меню}*      clrscr; *{очистка экрана, далее идёт вывод самого меню}*      Write('Џа®Ја\*¬¬\* ¤«п а\*Ў®вл б® ');      Textcolor(4);      Writeln('бвҐЄ®¬.');      Textcolor(7);      Writeln('‚лЎҐаЁвҐ ¦Ґ«\*Ґ¬®Ґ ¤Ґ©бвўЁҐ:');      Writeln('1) „®Ў\*ўЁвм н«Ґ¬Ґ\*в.');      Writeln('2) ‚лў®¤ бвҐЄ\*.');      Writeln('3) “¤\*«Ґ\*ЁҐ н«Ґ¬Ґ\*в\* Ї® §\*\*зҐ\*Ёо.');      Writeln('4) “¤\*«Ґ\*ЁҐ н«Ґ¬Ґ\*в\* Ї® Ї®ап¤Є®ў®¬г \*®¬Ґаг.');      Writeln('5) Џ®ЁбЄ н«Ґ¬Ґ\*в\* Ї® §\*\*зҐ\*Ёо');      Writeln('6) ‘®авЁа®ўЄ\* бвҐЄ\* ¬Ґв®¤®¬ "Џг§ламЄ\*", ¬Ґ\*пп в®«мЄ® ¤\*\*\*лҐ.');      Writeln('7) ‘®авЁа®ўЄ\* бвҐЄ\* б Ё§¬Ґ\*Ґ\*ЁҐ¬ \*¤аҐб®ў.');      Writeln('8) ‚ле®¤.');      writeln;      ch:=readkey; *{ожидаем нажатия клавиши}*  **case** ch **of** *{выбираем клавишу}*        '1':**begin**              write('‚ўҐ¤ЁвҐ §\*\*зҐ\*ЁҐ ¤®Ў\*ў«пҐ¬®Ј® н«Ґ¬Ґ\*в\* ');              readln(znach); *{считываем значение добавляемого нового элемент}*              AddElem(Stk,znach);  **end**;        '2':**begin**              clrscr; *{очистка экрана}*              Print(Stk); *{вызов процедуры вывода}*              readkey; *{ожидаем нажатия клавиши}*  **end**;        '3':**begin**              Write('‚ўҐ¤ЁвҐ §\*\*зҐ\*ЁҐ г¤\*«пҐ¬®Ј® н«Ґ¬Ґ\*в\* ');              readln(znach); *{ввод значения удаляемого элемента}*              DelElemZnach(Stk,znach); *{вызов процедуры удаления элемента по значению}*              readkey;*{ожидаем нажатия клавиши}*  **end**;        '4':**begin**              Write('‚ўҐ¤ЁвҐ Ї®ап¤Є®ўл© \*®¬Ґа г¤\*«пҐ¬®Ј® н«Ґ¬Ґ\*в\* ');              readln(znach); *{ввод позиции удаляемого файла}*              DelElemPos(Stk,znach);*{вызов процедуры удаления элемента по значению}*              readkey;*{ожидаем нажатия клавиши}*  **end**;        '5':**begin**              write('‚ўҐ¤ЁвҐ §\*\*зҐ\*ЁҐ ЁбЄ®¬®Ј® н«Ґ¬Ґ\*в\* ');              readln(znach); *{ввод искомого значения}*              tmpl:=SearchElemZnach(Stk,znach); *{вызываем процедуру поиска элемента по значению}*  **if** tmpl=**nil** **then** *{проверяем найден ли элемент и выводим соответствующие сообщения}*                write('?бЄ®¬л© н«Ґ¬Ґ\*в ®вбгвбвўгҐв ў бвҐЄҐ')  **else**                write('ќ«Ґ¬Ґ\*в ',tmpl^.data,' \*\*©¤Ґ\*');              readkey;*{ожидаем нажатия клавиши}*  **end**;        '6':**begin**  **if** Stk=**nil** **then** *{проверяем не пустой ли стек}*  **begin**                Write('‘вҐЄ Їгбв.');                readkey*{ожидаем нажатия клавиши}*  **end**  **else**  **begin**                SortBublInf(Stk);*{вызов процедуры сортировки стека с изменением данных}*                Write('‘вҐЄ ®вб®авЁа®ў\*\*.');                readkey;*{ожидаем нажатия клавиши}*  **end**  **end**;        '7':**begin**  **if** Stk=**nil** **then***{проверяем не пустой ли стек}*  **begin**                Write('‘вҐЄ Їгбв.');                readkey*{ожидаем нажатия клавиши}*  **end**  **else**  **begin**                SortBublLink(Stk);*{вызов процедуры сортировки стека с изменением адресов}*                Write('‘вҐЄ ®вб®авЁа®ў\*\*.');                readkey;*{ожидаем нажатия клавиши}*  **end**  **end**;  **end**;  **until** ch='8';    FreeStek(Stk); *{освобождаем память занятую стеком}*  **end**. |

1)**procedure AddToTree (var Tree:PNode;x:integer);** - добавление элемента в дерево  
2)**function Search(Tree:PNode;x:integer):PNode;** - функция поиска в дереве  
3)**procedure Lkp(Tree:PNode);** - обход дерева по принципу "Левле поддерево, корень, правое поддерево"  
4)**procedure DeleteTree(var Tree1:PNode )** - процедура удаления дерева

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112 | **uses** crt;  **type**  PNode=^Node;  *{Указатель на узел}*   Node=**record**  *{Тип запись в котором будет храниться информация}*     data:**integer**;     left,right:PNode; *{Ссылки на левого и правого сыновей}*  **end**;    **var**    Tree,p1:PNode; *{Tree адрес корня дерева, p1-вспомогательная переменная}*    n,x,i:**integer**;    ch:**char**; *{для работы менюшки}*    *{Процедура добавления элемента }*  **procedure** AddToTree (**var** Tree:PNode;x:**integer**); *{Входные параметры - адрес корня дерева и добавл элемент }*  **begin**  **if** Tree=**nil** **then**  *{Если дерево пустое то создаём его корень}*  **begin**       New(Tree);   *{Выделяем память }*       Tree^.data:=x;     *{Добавляем данные }*       Tree^.left:=**nil**;     *{Зануляем указатели на левого }*       Tree^.right:=**nil**;  *{и правого сыновей }*        exit;  **end**;  **if** x < Tree^.data **then**   *{Доб к левому или правому поддереву это завсит от вводимого элемента}*       AddToTree(Tree^.left,x)  *{если меньше корня то в левое поддерево }*  **else**      AddToTree(Tree^.right,x);  *{если больше то в правое}*  **end**;    *{Функция поиска по дереву}*  **function** Search(Tree:PNode;x:**integer**):PNode; *{Возвращает адрес искомого элемента, nil если не найден}*  **var**  p:PNode;   *{вспомог переменнная }*  **begin**  **if** Tree=**nil** **then**   *{если дерево пустое то}*  **begin**         Search:=**nil**;  *{присваеваем функции результат нил}*         exit; *{и выходим }*  **end**;  **if** x=Tree^.data **then**  *{если искомый элемент равен корню дерева то }*      p:=Tree  *{Пзапоминаем его адрес }*  **else**   *{иначе}*  **if** x < Tree^.data **then** *{если вводимый элемент менньше корня то }*            p:=Search(Tree^.left,x) *{то ищем его в левом поддереве}*  **else**     *{иначе }*           p:=Search(Tree^.right,x);  *{ищем в правом поддереве }*    Search:=p; *{присваеваем переменной с именем фугкции результат работы}*  **end**;    *{Обход дерева по принципу Левый-корень-правый }*  **procedure** Lkp(Tree:PNode);  **begin**  **if** Tree=**nil** **then**  *{Если дерево пустое }*     exit;      *{то выход }*    Lkp(Tree^.left);  *{иначе начинаем обход с левого подднрева}*    write('  ',Tree^.data); *{выводим данные }*    Lkp(Tree^.right);  *{обходим правое поддерево}*  **end**;    *{Процедура удаления дерева}*  **procedure** DeleteTree(**var** Tree1:PNode );  **begin**  **if** Tree1 <> **nil** **then**  **begin**              DeleteTree (Tree1^.LEFT);              DeleteTree (Tree1^.RIGHT);              Dispose(Tree1);  **end**;  **end**;  *{основная пограмма}*  **begin**   Tree:=**nil**;  **repeat** *{цикл для нашего меню}*        Writeln('Выберете действие ');      Textcolor(2);      Writeln('Доступны следующие пункты:');      Writeln('1) Создание дерева поиска');      Writeln('2) Поиск элемента в дереве');      Writeln('3) Вывод дерева');      Writeln('4) ‚Выход');      writeln;      readln(ch); *{ожидаем нажатия клавиши}*  **case** ch **of** *{выбираем клавишу}*        '1': **begin**              writeln(' kolv elementov');              readln(n);  **for** i:=1 **to** n **do**  **begin**                    writeln('Введите число');                    readln(x);                    AddToTree(Tree,x);  **end**;  **end**;         '2': **begin**                 writeln('Элемент для поиска');                 readln(x);                 p1:=Search(Tree,x);  **if** p1 <> **nil** **then**                      writeln('Найден')  **else** writeln('Такого элемента нет!');  **end**;          '3': **begin**                writeln('Само дерево');                Lkp(Tree);                writeln;  **end**;  **end**;  **until** ch='4';     DeleteTree(Tree);  **end**. |