**1. Понятие экземпляра и схемы структуры на примере стека. Элементы базисного множества динамической структуры стек.**

Переменная – множество значений и имя, которому можно присвоить конкретное значение. Структура данных Sa=(Ма, R; pa, p1), которая соответствует рассмотрению структуры как переменной величины, называется схемой структуры.Структура данных Sa\*=(Ма, R; pa, p1\*) с установленными значениями элементов, называется экземпляром. p1\* - отражает не отношения между элементами, а индивидуальные свойства элемента Структура данных Sa=(Ма, R; pa,p1), которая соответствует рассмотрению структуры как переменной величины, называется схемой структуры. Структуры с бинарными отношениями допускают случай графического изображения. Элементы множества изображаются точками или кружками; пары (ai,aj), для которых отношение истина, соединяются стрелкой от первого аргумента ко второму. Образ структуры с бинарными отношениями - ориентированный граф. Структуры, которым соответствует ориентированный граф с вершинами, лежащими на одной ломаной, называют линейными. Есть два особых элемента: начальный элемент a1: ("j) p(aj,a1)=л не имеет предшествующего элемента; конечный элемент an: ("j) p(an,aj)=л не имеет следующего элемента. Структуры данных являются операндами операций обработки.. Текущий набор адресов – линейная структура Sn=(a1 a2 … an). Пусть: P1 - отношение следования, порождаемое операцией вставки, P2 - отношение следования, порождаемое операцией исключения. Тогда стек есть структура S=(Mi,P1, P2), в которой каждый элемент – структура, в любой момент существует только один конкретный элемент из M, элементы частично упорядочены по включению. Динамическая структура есть математическая структура, которой соответствует частично-упорядоченное (по включению) базовое множество M, элементы которого являются структурами данных. При этом отношения включения индуцируются операциями преобразования структуры данных.

**2. Реализация списка на языке программирования высокого уровня.**

Звено списка представляется в виде объекта класса TLink. Образ памяти, выделенной для хранения стека, определяется в виде массива звеньев-объектов. Все свободные звенья объединяются в один список свободных звеньев. Звенья этого списка используются при необходимости свободной памяти, в этот список звенья должны возвращаться после освобождения. Вставка звена в список свободных звеньев: новое звено включается в начало списка свободных. Выборка звена из списка

свободных звеньев: для выделения используется первое звено списка свободных. Структура хранения стека – линейный список (начало списка – вершина стека). Вставка в стек: звено для нового значения берется в списке свободных. Выборка из стека: исключаемое звено из стека должно оказаться в списке свободных. Схемы работы со стеком и со списком свободных звеньев совпадают. Список свободных звеньев есть стек.

**2. Алгоритм обхода иерархического списка (итератор).**

Печать текста: схема обхода – текст текущей строки, текст подуровня, текст следующего раздела текста того же уровня (top-down-next). while (1) { if ( pLink != NULL ) { cout << pLink->Str;// обработка звена, St.push(pLink); // запись в стек, pLink = pLink->pDown;//переход на подуровень} else if ( St.empty() ) break; else {pLink = St.top(); St.pop();//выборка из стека, pLink = pLink->pNext; //переход по тому же уровню}. Ввод текста из файла: уровень текста в файле можно выделить строками специального вида (например, скобками '{' и '}'). Общая схема алгоритма: повторить: ввод строки, ЕСЛИ '}' ТО Завершить, ЕСЛИ '{' ТО Выполнить рекурсивно Ввод\_текста, Добавить строку на том же уровне.

**1. Общая характеристика стандартной библиотеки шаблонов.**

В стандарте языка С++ предусматривается наличие в среде программирования стандартной библиотеки шаблонов (Standard Template Library, STL).

Основные понятия библиотеки STL. Библиотека включает в свой состав большое количество контейнеров, представляющих собой структуры данных, в которых могут храниться объекты. В числе имеющихся контейнеров: vector<T> - вектор переменного размера, list<T> - двусвязный список, queue<T> - очередь, stack<T> - стек, deque<T> - дек, priority\_queue<T> - приоритетная очередь, set<T> - множество, multiset<T> - множество с повторением элементов, map<key,val> - ассоциативный массив (таблица), multimap<key,val> - ассоциативный массив с повторением ключей. Для быстрого и эффективного построения вычислительных процедур, библиотека обеспечивает итераторы для всех видов контейнеров, которые представляют унифицированный механизм последовательного доступа к элементам контейнеров.

Общая схема: <класс-контейнер>::iterator Iter; - объявление итератора, Iter = <объект-контейнер>.begin(); - установка на первый элемент, Iter != <объект-контейнер>.end(); - проверка на завершение, ++Iter – переход к следующему элементу.

В зависимости от типа контейнера, итератор может обеспечивать прямой доступ, быть одно- или дву- направленным, предназначенным только для чтения или записи и др. Библиотека содержит для контейнеров большое количество реализованных обобщенных алгоритмов. В числе таких алгоритмов: for\_each() - вызвать функцию для каждого элемента, find() - найти первое вхождение элемента, find\_if() - найти первое соответствие условию, count() - подсчитать число вхождений элемента, count\_if() - подсчитать число соответствий условию, replace() - заменить элемент новым значением, copy() - скопировать элементы, unique\_copy() - скопировать только различные элементы, sort() - отсортировать элементы, merge() - объединить отсортированные последовательности и др.

**2. Пример использования стеков: преобразование арифметических выражений в польскую форму записи.** Формат записи выражения. Выражение синтаксически правильно (без ошибок). Допускаются только однобуквенные идентификаторы для операндов. В записи выражения нет пробелов; выражение заканчивается знаком '=': A+(B-C)\*D-F/(G+H)=. Алгоритм: 1) Для операций вводится приоритет: '\*' '/' (3), '+' '-' (2), '(' (1), '=' (0), 2) Для хранения данных используется 2 стека (1 – для результата, 2 – для операций), 3) Исходное выражение просматривается слева направо, 4) Операнды по мере их появления помещаются в стек 1 , 5) Символы операций и левые скобки помещаются в стек 2, 6) При появлении правой скобки последовательно изымаются элементы из стека 2 и переносятся в стек 1. Данные действия продолжаются либо до опустошения стека 2 либо до попадания в стеке 2 на левую скобку, 7) Если текущая операция, выделенная при обходе выражения, имеет меньший (более низкий) приоритет, чем операция на вершине стека 2, то такие операции из стека 2 переписываются в стек 1.

**2. Представление многочленов от нескольких переменных. Исключение хранения мономов с нулевыми коэффициентами.** Полиномы как формальный объект хорошо изучены в математике. Математическая модель – алгебра полиномов. Под многочленом понимается выражение из нескольких термов, соединенных знаками сложения или вычитания. Терм включает коэффициент и моном, содержащий одну или несколько переменных, каждая из которых может иметь степень P = S Coeff\*XAYBZC. В число возможных вычислительных процедур над полиномами входят действия по вычислению значений полинома при заданных значениях переменных, а также большинство известных математических операций (сложение, вычитание, вычисление частных производных, интегрирование и т.п.). Возможный вариант структуры хранения – стеки. Одна из наиболее частных операций – приведение подобных мономов. Для ускорения поиска подобных элементов целесообразно ввести какое-либо правило упорядочения мономов (отношение следования). Возможный подход – организация порядка по аналогии с упорядочением слов в словарях (лексикографический порядок). Установим старшинство переменных в порядке XYZ. Тогда XA1YB1ZC1> XA2YB2ZC2 ÛУстановленный порядок является линейным Þ правило следования может быть сформулировано в более простом виде. Пусть область возможных значений степеней переменных имеет вид 0 £ A, B, C < 10. Тогда можно установить взаимно-однозначное соответствие троек (A, B, C) и целых чисел следующим образом (A, B, C) ~ ABC =A\*100+B\*10+C Обратное соответствие определяется при помощи выражений A=E(ABC%100), B=E(ABC-A\*100)%10, C=ABC-A\*100-B\*10. Получаемые по степеням целые величины ABC (будем называть их далее свернутыми степенями или индексами) порождают тот же самый, ранее установленный, порядок следования мономов XA1YB1ZC1> XA2YB2ZC2 Û ABC1>ABC2. Приведенные выражения представляют собой правила позиционной системы счисления (где N=10 есть основание системы). Таким образом, на множестве мономов определено отношение следования и многочлен может быть рассмотрен как линейная структура, элементами которой являются термы. В ходе вычислений количество мономов в полиноме может изменяться Þ полиному соответствует динамичес Структура хранения полинома, тождественно равного нулю, не содержит звеньев (список вырождается). Данная ситуация может отражаться установкой нулевого значения указателю на список, но тогда все программы для полиномов должны включать специальные действия по обнаружению и обработке этого уникального состояния полинома. Возможное решение проблемы может состоять во введении дополнительного служебного звена, размещаемого в начале списке (звено-заголовок). Звено-заголовок маркируется логически-недопустимым значением индекса монома. Аналогичным образом можно уйти от проверки нулевого указателя последнего звена, установив в последнем звене в качестве следующего звена первый элемент списка (звено-заголовок). Данная модификация приводит к использованию в качестве структуры хранения циклический список. Структура хранения нулевого полинома имеет вид:



**2. Сравнение структур хранения линейных и динамических структур данных.**

|  |  |
| --- | --- |
| Линейные структуры | Динамические структуры |
| Базисное множество – множество элементов | Элементы базисного множества являются структурами ("ДС – структуры структур") |
| Базисное отношение – отношение следования | Базисное отношение – отношение следования |
| В структуре хранения хранятся все элементы структуры | В структуре хранения хранится только текущий элемент структуры |
| Отношение следования реализуется при помощи адресной арифметики | Отношение включения реализуется при помощи программ |

**1. Таблицы с вычислимым входом. Запись и**

**поиск при переполнении (способ открытого перемешивания).**

Функция преобразования значения ключа к номеру (адресу) строки памяти для хранения записи H: K → L (L={0,…,M-1}) называется функцией (хеширования, перемешивания, рассеивания) расстановки (hash - мешанина, путаница).Таблицы, представление которых организуется при использовании функции расстановки, называются таблицы с вычислимыми адресами (хеш-таблицы, перемешиваемые таблицы). Ситуация, когда для расположения записи функцией расстановки определяется уже занятая строка памяти, называется относительным переполнением (коллизией). Уменьшение эффекта сгущений может быть достигнуто при применении способа открытого или линейного перемешивания s' = (s+p)modM (1≤p<M). Возможное решение состоит в выборе взаимно-простых значений для M и p. В более общем виде правило разрешения коллизии может быть представлено как функция вторичного перемешивания s'=h'(s). Теорема. Алгоритм открытого перемешивания при взаимно-простых M и p гарантирует нахождение свободных строк структуры хранения таблицы. Доказательство. Рассмотрим множество G={0,1,…,M-1} c операцией a⊕b=(a+b)mod M. Свойства операции: G замкнута относительно ⊕, операция ассоциативна и коммутативна, ∃ нулевой и обратные элементы ⇒ Множество G с операцией ⊕ является группой. Выделим подмножество G'={0, a, a ⊕ a,…}. Такое множество G' с операцией ⊕ тоже является группой (такие группы называются циклическими). Обозначим a ⊕ a ⊕…⊕ a через na (a - число повторений). Если n>0, то минимальное значение n, при котором na=0, называется порядком элемента a и обозначается |a|(т.е. порядок определяет количество итераций открытого перемешивания, после которого начнется повторение строк). Целое значение в операции (n a) / M получится только при n=M (т.к. a<M и для взаимно простых а и M). Но это означает также, что na=0, и, тем самым, |a|=M. Отсюда следует G=G'. При разрешении коллизии просматриваемые строки могут рассматриваться как список, в котором порядок следования определяется при помощи алгоритмического правила. Тем самым, удаление записи в середине подобного списка не должно разрушать связность записей. Это может быть достигнуто специальной маркировкой строк с удаленными записями. Строка структуры хранения имеет три возможных состояния – свободное, занятое, пустое (пустое состояние строки возникает после удаления хранимой в строке записи). Вставка (окончательный вариант). 1) Если n==M, ТО { Переполнение; Останов }, 2) f=-1 // f – номер первой найденной пустой строки, 3) s = h(key) // применение функции расстановки, 4) ЕСЛИ s занята и K[s]==key, ТО {Дублир.; Останов }, 5) ЕСЛИ s пустая и (f<0), ТО { f = s }, 6) ЕСЛИ s свободна и (f < 0), ТО { K[s]=key; Останов }, 7) ЕСЛИ s свободна и (f >-1), ТО { K[f]=key; Останов }, 8) (**!**) Коллизия {s = (s+p) mod M и переход к п. 4 }. Поиск. 1) f=-1 // f – номер первой найденной пустой строки, 2) s = h(key) // применение функции расстановки, 3) ЕСЛИ s занята и K[s]==key, ТО { Останов }, 4) ЕСЛИ s пустая и (f<0), ТО { f = s }, 5) ЕСЛИ s свободна, ТО { Останов }, 6) (**!**) Коллизия { s = (s+p) mod M и переход к п. 3 }.

**2**

**. Понятие линейного списка.**

Необходимость перепаковки для обеспечения динамического распределения памяти возникает в силу принятого способа реализации отношения следования - следующий элемент структуры располагается в следующем элементе памяти (с адресом, большим на 1). Устранение перепаковки возможно только при кардинальном изменении способа реализации основных отношений – необходимо допустить размещение следующих элементов структуры в произвольных элементах памяти (там, где имеется свободные области памяти). Возможность такого подхода может быть обеспечена запоминанием для каждого текущего элемента структуры адреса памяти, где хранится следующий элемент. Интерпретация содержимого элемента памяти (значение или адрес следующего элемента) в самом простом варианте может быть обеспечена фиксированным форматом используемых участков памяти. Под квантом памяти понимается последовательность элементов памяти с последовательно-возрастающими адресами. Именем (адресом) этой группы считается адрес первого слова кванта. Элементы кванта называются полями. В общем случае, набор элементов памяти, связанных с одним именем, называют звеном. Далее будут использоваться двухэлементные звенья памяти, в которых первое поле будет использоваться для хранения значений, а второе поле – для запоминания адресов. Способ задания отношения следования, в котором фиксация месторасположения следующего элемента производится путем запоминания соответствующего адреса памяти, называется сцеплением (пары, хранящие ai и ai+1, сцеплены адресными указателями). Для изображения структуры хранения с использованием сцепления звенья памяти рисуются в виде прямоугольников, а сцепление звеньев показывается в виде стрелок. Индикация последнего звена в списке обычно производится записью в поле адреса некоторого барьера – фиктивного (неадресного) значения (как правило, 0 или -1). Для доступа к звеньям списка должен быть известен адрес первого звена списка. Указатель, в котором этот адрес запоминается, называется переменной связи. Структура хранения данного типа (звенья, сцепление, барьер, переменная связи) называется линейным или односвязным списком.

**1. Алгоритм обхода плекса.**

Алгоритм 1. // общая схема алгоритма обхода для плекса: // переход на крайнюю левую точку, while ( pN Ï TChartPoint ) { St.push(pN); pN = pN->GetFirstPoint();}//подъем по плексу и рисование, pF = pN; while ( !St.Empty() ) {pN = St.top(); St.Pop(); pL = pN->GetLastPoint();//рисование линии <pN,pF,pL> pF = pL;}. Переходим на крайнюю левую точку ,поднимаемся по плексу (пока стек не пуст, достаем точки траектории и рисуем линии). Алгоритм 2. Рекурсивный вариант (общая схема): TChartPoint \*Show ( TChart \*pN ) {if (pN != NULL) pL = NULL; else if (pN Î TChartPoint) pL = pN; else {pF = Show(pN->GetFirstPoint()); pL = Show(pN->GetLastPoint());//рисование линии <pN,pF,pL>} return pN;}. Линии, определяемые при обходе плекса, помещаются в стек. При линии, извлекаемой из стека, последовательно определяются начальная и конечная точки. Для определения начальной точки используется метод GetFirstPoint линии. Если получаемый указатель указывает на линию, обработка текущей линии откладывается (линия помещается в стек) и начинается анализ новой линии, данная процедура выполняется итеративно до получения линии, с которой начали.

**2. Линейные структуры данных**

Структуры с бинарными отношениями допускают случай графического изображения. Элементы множества изображаются точками или кружками; пары (ai,aj), для которых отношение истина, соединяются стрелкой от первого аргумента ко второму. Образ структуры с бинарными отношениями - ориентированный граф. Структуры, которым соответствует ориентированный граф с вершинами, лежащими на одной ломаной, называют линейными. Есть два особых элемента:

начальный элемент a1: ("j) p(aj,a1)=л не имеет предшествующего элемента; конечный элемент an: ("j) p(an,aj)=л не имеет следующего элемента.

**1. Изменение структуры текста (вставка и удаление строк)**

void InsNextLine(char \*s) //вставка строки в том же уровне {if (pCurr != NULL){Zveno \*tmp; tmp = new Zveno(s, pCurr->pNext, NULL); pCurr->pNext = tmp; }}

void InsNextSection(char \*s) //вставка раздела в том же уровне {if (pCurr != NULL) { Zveno \*tmp; tmp = new Zveno(s, NULL, pCurr->pNext); pCurr->pNext = tmp;}}

void InsDownLine(char \*s) //вставка строки в подуровке {if (pCurr != NULL){ Zveno \*tmp; tmp = new Zveno(s, pCurr->pDown, NULL); pCurr->pDown = tmp;}}

void InsDownSection(char \*s) //вставка раздела в подуровне {if (pCurr != NULL) { Zveno \*tmp;

tmp = new Zveno(s, NULL, pCurr->pDown); pCurr->pDown = tmp; }}

void DelNext()//удаление строки {if (pCurr->pNext != NULL){Zveno \*tmp; tmp = pCurr->pNext;pCurr->pNext = pCurr->pNext->pNext;}}

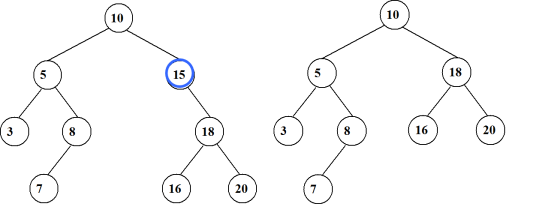
void DelDown()//удаление раздела{ if (pCurr->pDown != NULL) { Zveno \*tmp; tmp = pCurr->pDown; pCurr->pDown = pCurr->pDown->pNext;} }

**2. Деревья поиска. Алгоритм удаления.**

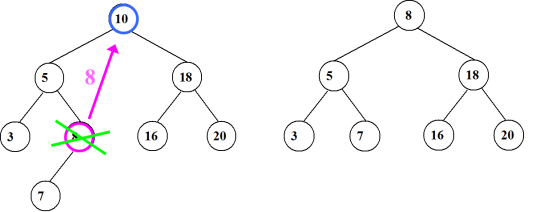
Сложность вставки и удаления в упорядоченных таблицах вызвана использованием непрерывной памяти, что приводит, как следствие, к необходимости перепаковок данных. Устранение перепаковок возможно только при использовании списковой памяти, но в этом случае теряется возможность прямого доступа к данным. Достижение эффективности двоичного поиска при списковой структуре хранения требует прямого доступа к звену со средней записью таблицы; из этого звена должна существовать возможность доступа (указатели) к средним звеньям левой и правой частей таблицы и т.д.

"Вычисленная" структура хранения является деревом поиска.

Возможные варианты: удаление листа, удаление узла с одним потомком.



Удаление узла с двумя потомками.



void TTreeTable :: DelRecord ( TKey k ) { // удалить запись

if ( FindRecord(k) == NULL ) SetRetCode(TabNoRec); // SKIP\_ON

else {

SetRetCode(TabOK);

PTTreeNode pNode = \*ppRef;

if ( pNode->pRight == NULL ) \*ppRef = pNode->pLeft; // один потомок слева

else if ( pNode->pLeft == NULL ) \*ppRef = pNode->pRight; // один потомок справа

else { // два потомка - поиск крайнего справа у левого поддерева

PTTreeNode pN = pNode->pLeft, \*ppR = &pNode->pLeft;

while ( pN->pRight != NULL ) {

ppR = &pN->pRight; pN = \*ppR;

} // вместо удаления pNode удается pN

pNode->pValue = pN->pValue; // значение в pNode

pNode->Key = pN->Key;

pNode = pN; \*ppR = pN->pLeft; // обход удаляемого pN

}delete pNode; } }

**Упорядоченные таблицы. Алгоритм быстрой сортировки.**

Таблицы, в которых записи располагаются в порядке возрастания (или убывания) ключей, называются сортированными (упорядоченными). Действия, связанные с размещением записей в порядке возрастания (или убывания) ключей называют сортировкой. Идея похода (Hoare C.A.R.)– использование процедуры разделения упорядочиваемого набора на две части, в одной из которых располагаются значения, меньшие некоторого порогового (ведущего) элемента массива, в другой – соответственно большие значения. Подобный способ разделения может быть выполнен без привлечения дополнительной памяти. При наличии процедуры разделения алгоритм сортировки может быть определен рекурсивно – необходимо разбить упорядочиваемый набор на два блока с меньшими и большими значениями соответственно и затем последовательно отсортировать полученные блоки.

key=k[0];//

left=1;right=n-1;

//пока блоки не пересекаются

while(left<=right)

// пока левый<=ведущего, то вправо

while(left<n && (k[left]<=key)) left++;

// пока правый элемент, ведущего,то влево

while (k[right]>key) right--;

//перестановка

if(right>left){temp=k[left]; k[left]=k[right]; k[right]=temp;}

//установка ведущего между блоками

k[0] =k[right]; k[right]=key; i=right;

рекурсивно применить первые шаги к обоим массивам

Оценка сложности: Tmin = N log2N, Tmax = N2.





Оценка сложности: Tср= 1.4(N+1) log*2* N.

**1.Организация доступа по имени. Просматриваемые таблицы.**

Для чтения или записи значения необходимо указать адрес элемента памяти. Для человека более привычный способ указания данных – имя. Пусть K - множество имен, A - множество адресов, тогда отношение "иметь имя" есть функция f: K → A. Возможный способ – табличное задание функции. Таблица - последовательность строк (записей). Запись может состоять из нескольких полей. Одно из полей должно задавать имя записи (ключ), остальные поля образуют тело записи. Операции с таблицей: поиск записи по ключу, вставка новой записи, удаление записи. Таблица – динамическая структура данных. Базисное множество – семейство линейных структур из записей, базисное отношение включения определяется операциями вставки и удаления записей. Пусть K - множество имен, A - множество значений, A*\*=* A*+*æ - расширенное множество (æ∉A), Z = K×A*\** - множество записей (z =<k,a>∈Z), *Z*- множество всех подмножеств Z. Тогда таблица есть структура T = <*Z*, p>, где p - базисное отношение включения, tnk∈T - текущий элемент (состояние) из k записей (n – размер памяти). Основные операции: поиск по ключу, вставка записи, исключение записи. Дополнительные операции: создание таблицы, предикат проверки пустоты, проверка переполнения памяти. При завершении операции формируется код выполнения δ

=1 – таблица пуста, =2 – таблица заполнена, =3 – записи в таблице нет, =4 – запись в таблице уже есть. Аксиомы: созданная таблица является пустой, таблица после операции вставки не пуста, таблица после исключения записи не полна, после вставки записи в таблице поиск этой же записи должен быть успешным, после удалении записи в таблице данная запись должна отсутствовать, результат поиска не зависит от операций вставки и удаления записей с другими ключами. Теоретико-множественные операции: пересечение, объединение, вычитание таблиц. Поиск – линейный (последовательный) просмотр. Вставка – добавление в конец списка. Удаление – исключение (стирание) с уплотнением (ex., путем переписыванием последней записи в удаляемую строку). Сложность операций просматриваемых таблиц: Поиск: Tmin=1, Tmax=N, Tср=N/2 (поиск имеющихся в таблице записей), =p(N/2)+(1-p)N (p-вероятность наличия записи). Вставка: Tср=p(N/2)+(1-p)(N+1)=N+1 (вставка без дублирования). Удаление: Tср=p(N/2+1)+(1-p)N

**1. Деревья поиска как способ организации таблицы. Алгоритмы обхода.**

Связный граф без циклов называется деревом. Структура типа дерева (древовидная структура) с базовым типом T – это либо пустая структура, либо узел (вершина) со значением типа T, с которым связано конечное число древовидных структур (поддеревьев) с базовым типом T. Если для любой вершины бинарного дерева значения в левом потомке меньше значения узла, а значение в правом потомке больше значения узла, то такое дерево называется деревом поиска. Обработка дерева – выполнение необходимой операции для каждой узла дерева. Реализация подобного типа действий предполагает умение обхода (обхода) дерева

Представим дерево в общем виде: T – top (корень), L – left (левое поддерево), R – right (правое поддерево).

Тогда возможны следующие варианты обхода: T, L, R – сверху вниз; L, T, R – слева направо; L, R, T – снизу вверх; T, R, L – сверху вниз; R, T, L – справа налево; R, L, T – снизу вверх. Пример: представление арифметического выражения в виде бинарного дерева. Печать значений дерева поиска (схема LTR, рекурсия). void TTreeTable :: PrintTreeTab(ostream &os, PTTreeNode pNode) { if ( pNode != NULL ) { // печать дерева с вершиной pNode, PrintTreeTab(os,pNode->pLeft); pNode->Print(os); os << endl; PrintTreeTab(os,pNode->pRight);}}

**2. Сравнение непрерывной и списковой структур хранения**

|  |  |
| --- | --- |
| Непрерывная память | Списки |
| Перепаковка для динамического распределения памяти | Динамическое распределение памяти эффективно реализуется при помощи списка свободных звеньев |
| В структуре хранения хранятся только данные | В структуре хранения хранятся данные и указатели |
| К элементам структуры данных обеспечивается прямой доступ | К элементам структуры данных обеспечивается последовательный доступ |

**2. Структуры хранения динамических структур типа стек и очередь**

стек, |S| - количество элементов в стеке, Snk - стек, n – размер памяти, k – количество значений, d - код выполнения операции (=1 – стек пуст, =2 – стек полон). Операции: S(n) ® Sn0 - создание стека, S<<x ® S' – вставка в стека (при k=n Þ S' = S и d=2), S>>x ® S' - исключение из стека (при k=0 Þ S' = S и d=1), a0(S) - предикат проверки пустоты стека (a 0(S)=1 при k=0), a(S) - предикат проверки переполнения памяти (a(S)=1 при k=n). Аксиомы: S=S(n) Þ S = Sn0 и a 0(S)=1 – созданный стек является пустым, S'=S<<x Þ a0(S)=0 – стек, в который выполнена вставка, не пуст, S'=S>>x Þ a(S) =0 – стек после операции исключения не полон, S'=(S<<x)>>y Þ x=y – при выборке значения из стека извлекается последнее вставленное значение, S'=S>>x, a0(S)=1 Þ S' = S и d=1 – выборка значения из пустого стека устанавливает код завершения d=1, S'=S<<x, a(S)=1 Þ S' = S и d=2 – вставка значения в полный стек устанавливает код завершения d=2. Для хранения элементов базисного множества выделяется вектор памяти размера, достаточного для хранения максимально возможного количества значения в стеке. Хранение значений в памяти осуществляется последовательно от младшего элемента вектора к старшему. Для запоминания количества хранимых в стеке значений используется индекс последнего занятого элемента в векторе.

#define MemSize 20

class TStack{protected:int Mem[MemSize]; //память для структуры данных

int Top;//индекс вершины стека

public: TStack(){Top=-1;} int IsEmpty(void)const{Top==-1} int IsFull(void)const {Top==MemSize-1;} void Put(const int Val) {Mem[++Top]=Val;} TData Get9void0{return Mem[Top--];} Очередь (вставка в конец очереди, исключение из начала – дисциплина FIFO – first in, first out). Вставка значений происходит в начало очереди, исключение значений – с конца очереди Þ для индикации начала и конца очереди требуется два индекса. Li – конец, Hi – начало. Выборка из элемента с индексом Li и увеличение Li Вставка - увеличение Hi и запись в элемент с индексом Hi. В ходе вычислений может возникнуть ситуация Li= Hi=n-1. Тогда вставка нового значения невозможна, а Emem»0. Способы достижения полного использования памяти: сдвиг значений очереди после выборки очередного значения (т.е. обеспечение Li=0) – возрастание накладных расходов, использование левого участка свободной области при достижении Hi=n-1 (т.е. при отсутствии свободной памяти справа). Структура хранения, получаемая из вектора расширением отношения следования парой p(an,a1), называется циклическим или кольцевым буфером. Реализация кольцевого буфера логически может быть обеспечена переходом индексов Li и Hi при достижении граничного значения MemSize-1 на индекс первого элемента вектора памяти.

Class TQueue: public TStack {protected: int Start; virtual int GetNextIndex(int Index) {return ++index%MemSize;} public: TWueue(it size=DerMemSize): TStack(size){size=0;} xirtual TData Get(void){TData tmp=-1; if(pMem==NULL) SetRetCode; else if (IsEmpty()) SetRetCode();else {tmp=pMem[i]; Start=GetNextInsex(Start);DataCount--;}return tmp;}

**2. Сравнение структур хранения линейных и динамических структур данных.**

|  |  |
| --- | --- |
| Линейные структуры | Динамические структуры |
| Базисное множество – множество элементов | Элементы базисного множества являются структурами ("ДС – структуры структур") |
| Базисное отношение – отношение следования | Базисное отношение – отношение следования |
| В структуре хранения хранятся все элементы структуры | В структуре хранения хранится только текущий элемент структуры |
| Отношение следования реализуется при помощи адресной арифметики | Отношение включения реализуется при помощи программ |

**1. Комбинирование геометрических объектов (плексы).**

геометрический объект может быть образован при помощи сборки существующих объектов - рассмотрим данны способ построения новых объектов на примере рисунков (чертежей), образованных только их объектов двух базовых типов: точек и линий.узел структуры хранения представляет линию чертежа. указатель на начальную точку линии может также указывать на линию, т.е. конечная точка предыдущей линии является начальной точкой следующей линии. структура хранения данного типа называется плексом(содержит элементы разного типа) повторяющаяся точка на чертеже должна быть представлена обним и тем же объектом. разработанная структура хранения позволяет обеспечить представление чертежей, которые можно нарисовать без отрыва карандаша от бумаги.общая схема алгоритма обхода для плекса. Алгоритм 1. // общая схема алгоритма обхода для плекса: // переход на крайнюю левую точку, while ( pN Ï TChartPoint ) { St.push(pN); pN = pN->GetFirstPoint();}//подъем по плексу и рисование, pF = pN; while ( !St.Empty() ) {pN = St.top(); St.Pop(); pL = pN->GetLastPoint();//рисование линии <pN,pF,pL> pF = pL;}. Переходим на крайнюю левую точку ,поднимаемся по плексу (пока стек не пуст, достаем точки траектории и рисуем линии). Алгоритм 2. Рекурсивный вариант (общая схема): TChartPoint \*Show ( TChart \*pN ) {if (pN != NULL) pL = NULL; else if (pN Î TChartPoint) pL = pN; else {pF = Show(pN->GetFirstPoint()); pL = Show(pN->GetLastPoint());//рисование линии <pN,pF,pL>} return pN;}. Линии, определяемые при обходе плекса, помещаются в стек. При линии, извлекаемой из стека, последовательно определяются начальная и конечная точки. Для определения начальной точки используется метод GetFirstPoint линии. Если получаемый указатель указывает на линию, обработка текущей линии откладывается (линия помещается в стек) и начинается анализ новой линии, данная процедура выполняется итеративно до получения линии, с которой начали.

**2. Роль гипотез о росте структур при разработке систем управления памятью путем перепаковки. Понятие смешанной стратегии распределения свободной памяти**

распределение памяти до начала процесса вычислений называется статическим. Распределение памяти в ходе выполнения программы называется динамическим распределением памяти. Процедура динамического перераспределения памяти путем перезаписи части хранимых значений в другую область памяти называется перепаковкой памяти иди перепаковкой. перепаковка обеспечивает эффективное использование одного ресурса ЭВМ(памяти) за счет другого ресурса(времени). Для выполнении требудется разработка управляющих программ

Выполнение функций анализа свободной памяти, планирование размещения структур,переписывание структур называется управление памятью. Комплекс программ реализующих управление памятью назвается системой управления памятью.

Гипотезы о поведении структур служат основой для принятия решений о распределении памяти. Формирование гипотез происходит в результате теоретического анализа модели решаемой задачи или может быть выполнено на основе статистических данных, получаемых в ходе вычислительных экспериментов с проектируемой программной системой. Гипотеза 1: Стеки используются с одинаковой интенсивностью, память разделяется между стеками поровну. Гипотеза 2: Интенсивность использования стеков различается. Конструктивное предположение о характере такой неравномерности может состоять в гипотезе сохранения локальных тенденций роста стеков, т.е. в каждый момент времени использование стеков на последующих шагах вычислений характеризуется точно таким же поведением, что и на предшествующих этапах обработки данных. Сохранение локальных тенденций роста: показатель роста стека δi = max ( 0, DataCount' i – DataCount i), суммарный показатель роста Δ = Σ δi, 0 ≤ i < N, правило распределение памяти для стеков в соответствии с их показателями роста Li'k=Li'k-1+ (Hik-1-Lik-1+1) + F\*(δi / Δ), 1 ≤ k < N. Гипотеза 3: Использование вероятностных предположений о поведении стеков. Пусть есть θ, 0≤θ≤1, вероятность выполнимости гипотезы сохранения локальных тенденций роста. Тогда Li'k=Li'k-1+ (Hik-1-Lik-1+1) + (θ)\*(F/N)+(1-θ)\*F\*(δi / Δ), 1 ≤ k < N.

(θ)\*(F/N) - распределение памяти поровну между стеками

(1-θ)\*F\*(δi / Δ) - распределение памяти пропорционально показателям роста

**Оценка параметров модели в ходе выполнения программ (адаптация).**

распределение памяти до начала процесса вычислений называется статическим. Распределение памяти в ходе выполнения программы называется динамическим распределением памяти. Процедура динамического перераспределения памяти путем перезаписи части хранимых значений в другую область памяти называется перепаковкой памяти иди перепаковкой. перепаковка обеспечивает эффективное использование одного ресурса ЭВМ(памяти) за счет другого ресурса(времени). Для выполнении требудется разработка управляющих программ

Выполнение функций анализа свободной памяти, планирование размещения структур,переписывание структур называется управление памятью. Комплекс программ реализующих управление памятью назвается системой управления памятью.

Гипотезы о поведении структур служат основой для принятия решений о распределении памяти. Формирование гипотез происходит в результате теоретического анализа модели решаемой задачи или может быть выполнено на основе статистических данных, получаемых в ходе вычислительных экспериментов с проектируемой программной системой. Гипотеза 1: Стеки используются с одинаковой интенсивностью, память разделяется между стеками поровну. Гипотеза 2: Интенсивность использования стеков различается. Конструктивное предположение о характере такой неравномерности может состоять в гипотезе сохранения локальных тенденций роста стеков, т.е. в каждый момент времени использование стеков на последующих шагах вычислений характеризуется точно таким же поведением, что и на предшествующих этапах обработки данных. Сохранение локальных тенденций роста: показатель роста стека δi = max ( 0, DataCount' i – DataCount i), суммарный показатель роста Δ = Σ δi, 0 ≤ i < N, правило распределение памяти для стеков в соответствии с их показателями роста Li'k=Li'k-1+ (Hik-1-Lik-1+1) + F\*(δi / Δ), 1 ≤ k < N. Гипотеза 3: Использование вероятностных предположений о поведении стеков. Пусть есть θ, 0≤θ≤1, вероятность выполнимости гипотезы сохранения локальных тенденций роста. Тогда Li'k=Li'k-1+ (Hik-1-Lik-1+1) + (θ)\*(F/N)+(1-θ)\*F\*(δi / Δ), 1 ≤ k < N.

(θ)\*(F/N) - распределение памяти поровну между стеками

(1-θ)\*F\*(δi / Δ) - распределение памяти пропорционально показателям роста

Пусть σ есть количество выполненных перепаковок памяти за некоторый отрезок времени Δt. Величина σ зависит от значения θ, и для повышения эффективности функционирования системы следует определить такое θ, чтобы количество перепаковок было минимальным, т.е. min σ(θ). Возможная схема определения оптимального значения θ может состоять в следующем: выполним оценку величины σ на последовательных друг за другом отрезках времени Δt и определим величину изменения количества выполненных перепаковок: Δ σ = σ'- σ, примем следующее правило корректировки значения θ: θ' = θ + Δθ, если Δσ ≤ 0, θ' = θ - Δθ, если Δσ > 0, где Δθ есть параметр схемы адаптации.

**2. Реализация структуры хранения нескольких стеков с использованием списков на языке высокого уровня**

Звено списка представляется в виде объекта класса TLink. Образ памяти, выделенной для хранения стека, определяется в виде массива звеньев-объектов. Все свободные звенья объединяются в один список свободных звеньев. Звенья этого списка используются при необходимости свободной памяти, в этот список звенья должны возвращаться после освобождения. Вставка звена в список свободных звеньев: новое звено включается в начало списка свободных. Выборка звена из списка свободных звеньев: для выделения используется первое звено списка свободных. Структура хранения стека – линейный список (начало списка – вершина стека). Вставка в стек: звено для нового значения берется в списке свободных. Выборка из стека: исключаемое звено из стека должно оказаться в списке свободных. Схемы работы со стеком и со списком свободных звеньев совпадают. Список свободных звеньев есть стек. class TMultiStack:public TMultiRoot{ protected: TStack \*pStack[stacknum+1]; //стеки-память выделяется из стекмем

int freememsize; protected:

telem \*pstackmem[stacknum+1];//базовые адреса для памяти стеков

int relocationcout;

int stackrecocation(int nst){ int is, ns, ks, k, res=0;  
pStack[nst]~>DataCount++; // захват памяти для переполненного стека  
// оценка свободной памяти int temp = FreeMemSize; FreeMemSize = GetFreeMemSize();  
if ( FreeMemSize > -1 ) {  
res = 1; // свободная память есть-перепаковка  
RelocationCount++;  
// определение нового расположения стеков  
SetStackLocation { pStackMem );  
// перемещение стеков  
for ( ns=0; ns<stackNum; ns++ )  
if ( pstackmem[ns]<pstack[ns]->pmem){ //смещение вправо  
for ( k=0; k<pStack[ns]~>Datacount;k++)  
pStackMem[ns][k] = pStack[ns]->pmem[k];  
pStack[ns]->SetMem(pStackMem[ns], pstackmem[ns+1]-pstackmem[ns];);}else if ( pStackMem[ns] > pStack[ns]->pmem  
for ( ks=ns; pStackMem[ks+1]->pmem;ks++);  
// CTeK (ks+1) первый справа,сдвигаемый влево  
for ( is=ks; is>=ns; is~~ ) {  
for ( k=pStack[is]—>DataCount-1;k>=0;k--)  
pStackMem[is][k} = pStack[is]->pmem[k];  
pStack[is}~>SetMem(pStackMem[is], pstackmem[is+1]-pstackmem[is];);}}

else // стек не перемешивается  
pStack[ns]~>SetMem<pStackMem[ns], pstackmem[ns+1]-pstackmem[ns];);}

pstack[nst]->datacount--;//возврат захваченной памяти

freememsize++;return res;}

int getfreememsize(void);

virtual void setstacklocation(telem \*pstackmem[]);{pstackmem[0]=&mem[0]; for (int i=0;i<=stacknum;i++) pstackmem[i]=pstackmem[i-1]+pstack[i-1]->datacount + freememsize/stacknem; pstackmem[stacknum]=pstack[stacknum]->pmem}

public: tmultistack(){int stacksize=memlimit/stacknum;

for (int i=0,pos=0;i<=stacknum;i++,pos+=stacksize){pstack[i]=new tstack(0);

if (i==stacknum-1) pstack[i]->setmem9&mem[pos], memlimit-pos);

if(i==stacknum) pstack->setmem(&mem[memlimit],0);

else pstack[i]->setmem(&mem[pos],stacksize);}

reloationcount=0;

freememsize=memlimit;

retcode=dataok;}

~tmultistack(){for (int i=0;i<=stacknum;i++)

delete pstack[i];}

int isempty(int ns) const{return pstack[ns]->isempty();}

int isfull( int ns) const{return freememsize==0;}

virtual void put(int ns, const tdata &val){setretcode(dataok);if (pstack[ns]->isfull()){if(!stackrelocation(ns)) setretcode(datanomem); }

if(retcode==dataok){pstack[ns]->put(val); int code=pstack[ns]->getretcode(); if(code==dataok) freememsize--; setretcode(code);}}

virtual tdata get(int ns)

int getrelocationcount{return relocationcount;}

void paint(int y, int x1, int x2)

1. Динамические структуры: свойства и применение.

Структуры данных являются операндами операций обработки.

Результаты вычислений также являются структурами, модель которых может как совпадать, так и отличаться от структуры исходных данных.

Пример. Организация последовательного вызова подпрограмм.

Для возврата в точку вызова необходимо запоминать адрес возврата.

При завершении вызываемой подпрограммы для возврата используется последний запомненный адрес.

В ходе вызова подпрограмм количество запоминаемых адресов постоянно изменяется (увеличивается при вызове очередной подпрограммы и уменьшается после завершения работы текущей подпрограммы).

Отличительная особенность – структура исходных и результирующих данных являются близкими.

Выполним анализ примера.

Текущий набор адресов – линейная структура Sn=(a1 a2 … an).

Пусть T – операция исключения последнего адреса.

Тогда T(a1 a2 … an an+1) = (a1 a2 … an). Пусть P – операция добавления нового адреса. Тогда P(an+1; (a1 a2 … an)) = (a1 a2 … anan+1).

Орграф результата является подорграфом орграфа операнда или включает его. Последовательное применение операций T и P позволяет получить набор состояний стека адресов. Пусть: P1 - отношение следования, порождаемое операцией вставки, P2  - отношение следования, порождаемое операцией исключения.

Тогда *стек* есть структура S=(Mi,P1, P2), в которой каждый элемент – структура, в любой момент существует только один конкретный элемент из M, элементы частично упорядочены по включению.

**Динамическая структура есть математическая структура, которой соответствует частично-упорядоченное (по включению) базовое множество M, элементы которого являются структурами данных. При этом отношения включения индуцируются операциями преобразования структуры данных.**

Пример. 1. очередь (FIFO) вставка в конец очереди исключение из начала

пример 2. Дек - вставка и исключение из начала и конца дека - FOLIFOLO

программы реализующие отношения включения, называются средствами поддержки динамической структуры.

**ИНТЕРНЕТ:**

**Динамические структуры данных – это структуры данных, память под которые выделяется и освобождается по мере необходимости.**

**Динамическая структура данных характеризуется тем что (СВОЙСТВА):**

* **ей выделяется память в процессе выполнения программы;**
* **количество элементов структуры может не фиксироваться;**
* **отсутствует физическая смежность элементов структуры в памяти;**
* **размерность структуры может меняться в процессе выполнения программы;**
* **в процессе выполнения программы может меняться характер взаимосвязи между элементами структуры**

**Необходимость в динамических структурах данных обычно возникает в следующих случаях(ПРИМЕНЕНИЕ):**

* **Используются переменные, имеющие довольно большой размер (например, массивы большой размерности), необходимые в одних частях программы и совершенно не нужные в других.**
* **В процессе работы программы нужен массив, список или иная структура, размер которой изменяется в широких пределах и трудно предсказуем.**
* **Когда размер данных, обрабатываемых в программе, превышает объем сегмента данных**

1. Реализация списка на языке программирования высокого уровня.

Подход 1:

Для имитации звеньев могут быть использованы два массива один из которых используется для хранения значений, другой- для хранения индексов следующих элементов. В этом случае, звено есть элемент массивов с одинаковым индексом, адрес (имя) звена – индекс массивов.

Подход 2:

С использованием ООП звено может быть представлено в виде объекта. Образ памяти, выделенной для хранения структур данных, в это случае будет представлять массив звеньев-объектов.

Class TLink

{

Public:

Intvalue;//значение

Intnext;//индекс следующего звена

Protected:

TLink();

};

TLinkMem[MemLimit];

А код хз какой вставить ибо,там тип всю лабу со списками?

**Билет№4:**

1. Реализация множества как набора номеров элементов универса. Оценка сложности по памяти. Оценка сложности по времени.

Множество – набор элементов.

Для множества определены операции:

проверка наличия элемента a∈A,

Получить значение бита n, при условии, если бит больше -1 и меньше количества битов, в противном случае возвращается 0:

intTBitField::GetBit(constint n) const

{

if ((n>-1) && (n <BitLen))

return (pMem[(GetMemIndex(n))] & (GetMemMask(n)));

else return(0);

}

добавлениеэлементаA+a,

Установление бита происходит в позицию n том случае, если бит больше -1 и меньше количества битов.

voidTBitField::SetBit(constint n)

{ if ((n>-1) && (n <BitLen))

pMem[(GetMemIndex(n))] |= GetMemMask(n);}

удаление элемента A –a.

Удалить бит в позиции n при условии, если бит больше -1 и меньше количества битов:

voidTBitField::ClrBit(constint n)

{ if ((n>-1) && (n <BitLen))

pMem[(GetMemIndex(n))] &= ~GetMemMask(n);}

Теоретико-множественные операции:

объединение A∪B,

Операция "или" для двух битовых полей, осуществляется с помощью сравнивания двух полей, причем, результирующее будет с длиной наибольшего из них. Затем последовательно выполняется дизъюнкция для i-ых элементов двух массивов:

TBitFieldTBitField::operator|(constTBitField&bf)

{

inti, len;

if (BitLen>bf.BitLen)

len = BitLen;

elselen = bf.BitLen;

TBitFieldtemp(len);

for (i = 0; i<MemLen; i++)

temp.pMem[i] = pMem[i];

for (i = 0; i<bf.MemLen; i++)

temp.pMem[i] |= bf.pMem[i];

return temp;

}

пересечение A∩B,

Операция "и" для двух битовых полей, осуществляется с помощью сравнивания двух полей, результирующее будет с длиной наименьшего из них. Затем последовательно выполняется конъюнкция для i-ых элементов двух массивов:

TBitFieldTBitField::operator&(constTBitField&bf)

{

inti, len;

if (BitLen<bf.BitLen)

len = BitLen;

elselen = bf.BitLen;

TBitFieldtemp(len);

for (i = 0; i<MemLen; i++)

temp.pMem[i] = pMem[i];

for (i = 0; i<bf.MemLen; i++)

temp.pMem[i] &= bf.pMem[i];

returntemp;

}

вычитание A\B

Для отрицания последовательно выполняется инверсия для каждого бита массива:

TBitFieldTBitField::operator~(void)

{

TBitFieldtemp(BitLen);

for (inti = 0; i<MemLen; i++)

temp.pMem[i] = ~pMem[i];

returntemp;

}

УниверсU – множество всех элементов.

Конкретизация (допущения и ограничения): элементы множества проиндексированы (каждому элементу соответствует уникальный индекс), множество индексов элементов составляют непрерывный диапазон целых значений. Тогда любое множество A⊂U может быть описано характеристическим вектором α=(α1 α2…αn), ,

если иначе. Множество → битовая строка → массив битовых элементов → оперативная память (обратный порядок хранения). Нумерация бит в битовой строке – слева направо. Нумерация элементов в массиве – слева направо, биты элемента – справа налево. Байты двухбайтового элемента располагаются в ОП в обратном порядке (сначала байт с младшими битами, затем байт со старшими битами).

Оценка сложности по времени:

битовое поле представляет из себя массив int .все операции с конкретным int это константа, т.к. там везде битовые операции они грубо говоря все О(1).остаётся только найти этот intмы там делили нацело, потом брали ещё остаток как смещение и шагали по массиву, а потом поint в этом массиве. это всё грубо говоря О(1) - очень быстро.

Оценка сложности по памяти:

мы делили на 32 (типоint 4 байта 1 байт 8 битов) значит в int 32 бита

будет [n/32] + 1

* (+1 ,потому что если элементов меньше 32, то деление даст 0) типо 5/32 = 0 деление же целочисленное значит для хранения 5 элементов множества тебе нужно 5/32 + 1 = 1 элемент в массиве это размер чтобы получить индекс мы делали n » 5то есть по факту делили на 32 без остатка, только делая это битовым сдвигом.битовые операции просто быстрее работают

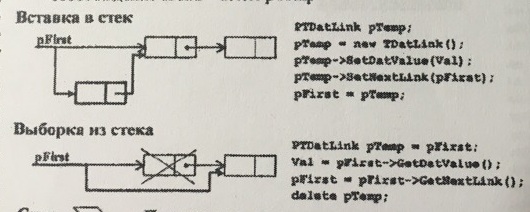
2. Линейный список как динамическая структура. Базисное множество. Неоднозначность операций вставки и удаления.

Среда выполнения обеспечивает динамически –распределяемую область памяти :

-звено

-выделение звена PTemp=newTDatLink()

-освобождение звена –delete pTemp



базисное множество - структуры

неоднозначность операций в том что можем вставлять как в голову так и в хвост и после текущего и удалять можно тоже по-разному. Для показа неоднозначности взять простой односвязный линейный список с головой

и двусвязный с хвостом и головой.

**Билет№6:**

1. Управление памятью путем перепаковки структур хранения на примере реализации N стеков на одном массиве.

Процедура динамического перераспределения памяти путем переписи части хранимых значений в другую область памяти называется перепаковкой памяти или просто перепаковкой.

Перепаковка обеспечивает эффективное использование одного ресурса ЭВМ (памяти) за счет другого ресурса (времени).

Для выполнения перепаковки требуется разработка управляющих программ.

Выполнение функций анализа свободной памяти, планирование размещения структур, переписывание структур называется управление памятью. Комплекс программ, реализующих управление памятью, называется системой управления памятью. Необходимость перепаковки обуславливается принятым способом реализации отношений следования.

**.**N – количество стеков, m – размер памяти. Свойства: Li0=0 - условие неподвижности 1 стека, Hik=Lik-1 - условие пустоты, Hik<Lik+1 - условие неперекрытия, Hik=Lik+1-1 - условие переполнения. Для выполнимости последних двух условий для всех стеков введем фиктивный стек N, для которого LiN=m.Будем предполагать, что все стеки используются с одинаковой интенсивностью ⇒ память распределяется всем стекам поровну: Li0=0, LiN=m, Lik=Lik-1+ m / N, 1 ≤k<N. Выполняется при попытке вставки нового значения в стек s, у которого отсутствует свободная память: F = Σ (Hik-Lik-1-1),1≤k<N. F =0 – свободной памяти нет, F =1 – свободен 1 элемент памяти и его следует отдать стеку s, F>1 – необходимо перераспределить свободную память. Для гарантированного выделения свободной памяти стеку s при наличии только одного свободного элемента памяти (случай 2), выполним: His=His+1 – перед началом процедуры оценки свободной памяти, His=His-1 – после завершения перепаковки. Снова предположим, что все стеки используются с одинаковой интенсивностью ⇒ свободная память должна распределиться всем стекам поровну: Li'0= Li0, Li'N= LiN, Li'k=Li'k-1+ (Hik-1-Lik-1+1) + F / N, 1≤k<N. На равномерности распределения памяти может сказаться целочисленность операции деления.??????

2. Структуры хранения для матриц специального вида

Ленточные матрицы.

Для хранения элементов можно выделить непрерывный вектор памяти размера 3\*n-2. Адрес (aij) = α + 3\*(i-1) + (j-i).

Треугольные матрицы.

Подход 1. Матрицы подобного вида можно представить как матрицы общего вида и использовать для хранения двухиндексные массивы. Используется память Vисп= n2, необходимая память Vнеоб= n (n+1)/2. Эффективность использования памяти Emem≈ 0.5.

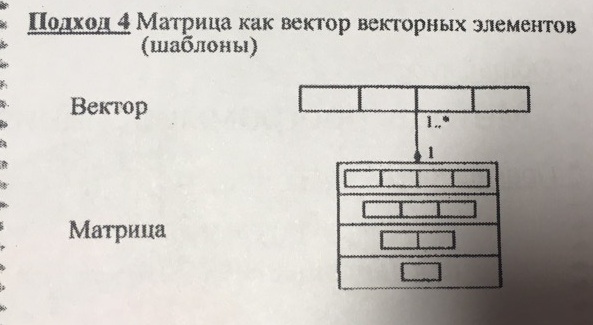
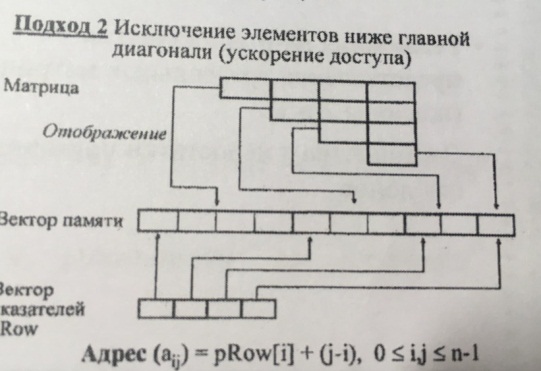
Подход 2. Исключение хранения элементов ниже главной диагонали (построчная запись в массив). Адрес **(**aij) = α + i\*n – i\*(i-1)/2 + (j-i), 0 ≤i,j≤n-1. Ускорение доступа – использование вектора указателей на первые элементы каждой строки. Адрес **(**aij) = pRow[i] + (j-i), 0 ≤i,j≤ n-1.

Подход 3. Представление матрицы в виде набора векторов.

Подход 4. Матрица как вектор векторных элементов (шаблоны).

**Билет№7:**

1. Матрица как вектор векторов и матрица как вектор указателей. Оценки сложности по памяти и времени.



по памяти :это же треугольные матрицы, в матрице нхн н квадрат ячееки нулевые мы не храним

по времени поиск будет О(1)т.к. мы просто индекс считаем грубо говоря,если с матрицами типо A + B,где а и б матрицы, то нам нужно каждый элемент с каждым сложитьсоотв. это будет такой же O как и у памятивычитание аналогично

а как скалярное произведение матриц-сумма попарных произведений?

2. Статическое и динамическое распределение памяти.

Распределение памяти до начала процесса вычислений называется статическим. Распределение памяти в ходе выполнения программы называется динамическим распределением памяти.

Процедура динамического перераспределения памяти путем переписи части хранимых значений в другую область памяти называется перепаковкой памяти или просто перепаковкой.

**Билет№8:**

1. Роль гипотез о росте структур при разработке систем управления памятью путем перепаковки на примере работы с N стеками.

Гипотезы о поведении структур служат основой для принятия решений о распределении памяти. Формирование гипотез происходит в результате теоретического анализа модели решаемой задачи или может быть выполнено на основе статистических данных, получаемых в ходе вычислительных экспериментов с проектируемой программной системой.

Гипотеза 1: Стеки используются с одинаковой интенсивностью, память разделяется между стеками поровну.

Гипотеза 2: Интенсивность использования стеков различается. Конструктивное предположение о характере такой неравномерности может состоять в гипотезе сохранения локальных тенденций роста стеков, т.е. в каждый момент времени использование стеков на последующих шагах вычислений характеризуется точно таким же поведением, что и на предшествующих этапах обработки данных. Сохранение локальных тенденций роста: показатель роста стека δi = max( 0, DataCount' i – DataCounti), суммарный показатель роста Δ = Σδi, 0 ≤i<N, правило распределение памяти для стеков в соответствии с их показателями роста Li'k=Li'k-1+ (Hik-1-Lik-1+1) + F\*(δi / Δ), 1 ≤k<N.

Гипотеза 3: Использование вероятностных предположений о поведении стеков. Пусть есть θ, 0≤θ≤1, вероятность выполнимости гипотезы сохранения локальных тенденций роста. Тогда Li'k=Li'k-1+ (Hik-1-Lik-1+1) + (1-θ)\*(F/N)+θ\*F\*(δi / Δ), 1 ≤k<N.

2. Структура хранения множеств.

Множество – набор элементов.

Для множества определены операции:

проверка наличия элемента a∈A,

Получить значение бита n, при условии, если бит больше -1 и меньше количества битов, в противном случае возвращается 0:

intTBitField::GetBit(constint n) const

{

if ((n>-1) && (n <BitLen))

return (pMem[(GetMemIndex(n))] & (GetMemMask(n)));

else return(0);

}

добавлениеэлементаA+a,

Установление бита происходит в позицию n том случае, если бит больше -1 и меньше количества битов.

voidTBitField::SetBit(constint n)

{ if ((n>-1) && (n <BitLen))

pMem[(GetMemIndex(n))] |= GetMemMask(n);}

удаление элемента A –a.

Удалить бит в позиции n при условии, если бит больше -1 и меньше количества битов:

voidTBitField::ClrBit(constint n)

{ if ((n>-1) && (n <BitLen))

pMem[(GetMemIndex(n))] &= ~GetMemMask(n);}

Теоретико-множественные операции:

объединение A∪B,

Операция "или" для двух битовых полей, осуществляется с помощью сравнивания двух полей, причем, результирующее будет с длиной наибольшего из них. Затем последовательно выполняется дизъюнкция для i-ых элементов двух массивов:

TBitFieldTBitField::operator|(constTBitField&bf)

{ inti, len;

if (BitLen>bf.BitLen)

len = BitLen;

elselen = bf.BitLen;

TBitFieldtemp(len);

for (i = 0; i<MemLen; i++)

temp.pMem[i] = pMem[i];

for (i = 0; i<bf.MemLen; i++)

temp.pMem[i] |= bf.pMem[i];

return temp;

}

пересечение A∩B,

Операция "и" для двух битовых полей, осуществляется с помощью сравнивания двух полей, результирующее будет с длиной наименьшего из них. Затем последовательно выполняется конъюнкция для i-ых элементов двух массивов:

TBitFieldTBitField::operator&(constTBitField&bf)

{ int i, len;

if (BitLen<bf.BitLen)

len = BitLen;

elselen = bf.BitLen;

TBitFieldtemp(len);

for (i = 0; i <MemLen; i++)

temp.pMem[i] = pMem[i];

for (i = 0; i <bf.MemLen; i++)

temp.pMem[i] &= bf.pMem[i];

returntemp;

}

вычитание A\B

Для отрицания последовательно выполняется инверсия для каждого бита массива:

TBitFieldTBitField::operator~(void)

{ TBitFieldtemp(BitLen);

for (int i = 0; i <MemLen; i++)

temp.pMem[i] = ~pMem[i];

returntemp;

}

УниверсU – множество всех элементов.

Конкретизация (допущения и ограничения): элементы множества проиндексированы (каждому элементу соответствует уникальный индекс), множество индексов элементов составляют непрерывный диапазон целых значений. Тогда любое множество A⊂U может быть описано характеристическим вектором α=(α1 α2…αn), ,

если иначе. Множество → битовая строка → массив битовых элементов → оперативная память (обратный порядок хранения). Нумерация бит в битовой строке – слева направо. Нумерация элементов в массиве – слева направо, биты элемента – справа налево. Байты двухбайтового элемента располагаются в ОП в обратном порядке (сначала байт с младшими битами, затем байт со старшими битами).

**Билет№9:**

1. Управление свободной памятью при использовании сцепления.
2. Алгоритм сложения многочленов от N переменных.

Основные алгоритмические моменты метода сложения полиномов operator+ состоят в следующем: результат сложения запоминается в объекте первого операнда; операция сложения сводится к последовательной обработке мономов полиномов-операндов p и q: если моном p меньше монома q, то моном q добавляется в полином p; и текущая позиция в q сдвигается вправо; если моном p старше монома q, то текущая позиция в p сдвигается вправо; если моном p равен моному q, то коэффициенты мономов складываются и запоминаются в p; далее если результат сложения равен 0, то моном в p удаляется и текущая позиция   
в q сдвигается вправо; если же результат сложения ненулевой, то текущая позиция сдвигается вправо и в p и в q.

TList TList::plus(TList list)

{pLink tmp, ptr, temp, prev1, prev2, cnt, n;

tmp = prev1 = cnt = Head; ptr = prev2 = temp = n = list.Head; size = this->GetSize(); int flag = 1; while ((tmp->pNext->data != -1) && (ptr->pNext->data != -1)) if ((ptr->data == -1) && (tmp->data == -1)) { ptr = ptr->pNext; tmp = tmp->pNext; temp = temp->pNext; cnt = cnt->pNext; } if ((tmp->data > ptr->data) && (flag == 1)){ptr = ptr->pNext; this->insCurrent(temp->data, temp->data2); list.deleteCurrent(temp->data); temp = ptr; flag = 0; }if ((tmp->data < ptr->data) && (flag == 1)){ptr = ptr->pNext; tmp = tmp->pNext; this->insCurrent(temp->data, temp->data2); list.deleteCurrent(temp->data);temp = ptr; flag = 0; }if ((tmp->data == ptr->data) && (flag == 1)) {this->insCurrent(tmp->data, tmp->data2 + ptr->data2); if ((tmp->data2 + ptr->data2) == 0){while ((prev1->pNext != tmp) && (n->pNext != ptr)) {prev1 = prev1->pNext; n = n->pNext; } tmp = tmp->pNext; ptr = ptr->pNext;

this->deleteCurrent(cnt->data); list.deleteCurrent(temp->data); cnt = tmp; temp = ptr; }flag = 0; }flag = 1; }if ((ptr->data == tmp->data) && (ptr->data != -1) && (tmp->data != -1)) {this->insCurrent(ptr->data, tmp->data2 + ptr->data2); if ((tmp->data2 + ptr->data2) == 0)

{while ((prev1->pNext != tmp) && (n->pNext != ptr)) {prev1 = prev1->pNext; n = n->pNext; }

tmp = tmp->pNext; ptr = ptr->pNext; this->deleteCurrent(cnt->data); list.deleteCurrent(temp->data); list.size--;

cnt = tmp; temp = ptr; }}if (ptr->data != tmp->data){if (tmp->data == -1) while (ptr->pNext != list.Head) { this->insCurrent(ptr->data, ptr->data2);ptr = ptr->pNext;list.size--;}}if ((ptr->data != -1) && (tmp->data != -1)){this->insCurrent(ptr->data, ptr->data2); list.deleteCurrent(ptr->data);

}}return \*this;}