Начинающие программисты, особенно студенты, часто пишут программы так:

Получив задание, тут же садятся за компьютер и начинают кодировать те фрагменты алгоритма, которые им удается придумать сразу. Переменным дают первые попавшиеся имена типа а, b, с или другие, более отражающие словарный запас автора, чем содержание величин. Когда компьютер зависает, безжалостно убивая первый порыв энтузиазма, делается перерыв, после которого написанные фрагменты стираются, и все повторяется заново.

В процессе работы несколько раз изменяются структуры данных, функции нервно удаляются и кодируются заново, а разбиение на модули делается только тогда, когда листать программу становится утомительно. Нечего говорить, что комментарии к программе не пишутся, а ее текст никак не форматируется. Периодически высказываются сомнения в правильности работы компилятора, компьютера и операционной системы.

Когда программа впервые доходит до стадии выполнения, в нее вводятся произвольные значения, после чего экран на некоторое время становится объектом пристального удивленного изучения. «Работает» такая программа обычно только в бережных руках хозяина на одном наборе исходных данных, а внесение даже небольших изменений может привести автора к потере веры в себя и ненависти к процессу программирования.

**1 этап.** *Постановка задачи*.

**2 этап.** *Разработка внутренних структур данных.* Большинство алгоритмов зависит от того, каким образом организованы данные. Структуры данных могут быть статическими и динамическими.

**3 этап.** *Проектирование* – определение общей структуры и взаимодействия модулей.

**4 этап.** *Программировани*е.

**5 этап.** *Тестирование*.

**Выбор структур данных должен предшествовать созданию алгоритмов! 3. Динамические структуры данных**

*Способы организации данных*:

- основные типы

- составные типы

*Память под данные выделяется*:

- на этапе компиляции

- во время выполнения программы с помощью операции **new** или функции **mallос**

В обоих случаях *выделяется**непрерывный участок памяти.*

Вопрос: что делать, если до начала работы с данными невозможно определить, сколько памяти потребуется для их хранения?

Решение: выделять память по мере необходимости отдельными блоками.

Вопрос: как между собой «связать» эти блоки?

Решение: с помощью указателей!

Получили ***динамическую структуру данных***- размер ***изменяется*** во время выполнения программы.

Динамическая структура ***может занимать несмежные участки*** оперативной памяти.

Динамические структуры данных ***различаются***

- способом связи блоков (будем называть их ***элементами***)

- допустимыми операциями.

Элемент любой динамической структуры данных содержит по крайней мере два поля:

1) поле для хранения данных

2) поле для указателя

В данном параграфе (кроме п.3.5) будем рассматривать реализацию динамических структур данных при помощи **структур** (**struct**)

**typedef int Data;**

**//типы - с заглавной буквы, переменные - со строчной**

**struct Elem{**

**Data key;**

**Elem\* ptr;**

**};**

***Линейный список***

- ***однонаправленный* (*односвязный*) *список***

- ***двунаправленный* (*двусвязный*) *список***

- ***кольцевой список***

***Операции****:*

- создать (добавить первый)

добавить

- добавить в конец

- добавить в заданное место

- найти

- удалить

- упорядочить ***и др***.

**3.1. Стек**

***Операции***:

- добавить

- просмотреть вершину

выбрать

- удалить

**struct Elem{**

**int key;**

**Elem\* ptr;**

**};**

**//добавить элемент в стек (с ошибкой)**

**int push(Elem\* top, int d){**

**Elem\* ph = new Elem;**

**ph->key = d;**

**ph->ptr = top;**

**top = ph;**

**return 0;**

**}**

**//добавить элемент в стек**

**int push(Elem\* \*top, int d){**

**Elem\* ph = new Elem;**

**ph->key = d;**

**ph->ptr = \*top;**

**\*top = ph;**

**return 0;**

**}**

**//выбрать элемент из стека, стек не пуст**

**int pop(Elem\* \*top){**

**int d = (\*top)->key;**

**Elem\* ph = \*top;**

**\*top = (\*top)->ptr;**

**delete ph;**

**return d;**

**}**

**int main(){**

**Elem\* top = 0;**

**for(int i=1; i<=10; i++)**

**push(&top,i);**

**while(top)**

**printf("%i\n",pop(&top));**

**return 0;**

**}**

**3.2. Очередь**

***Операции***:

- добавить

- выбрать

**struct Elem{**

**int key;**

**Elem\* ptr;**

**};**

**//добавить элемент в конец очередь**

**int add(Elem\* \*pbeg, Elem\* \*pend, int d){**

**Elem\* ph = new Elem;**

**ph->key = d;**

**ph->ptr = 0;**

**if(\*pend) //если очередь не пуста**

**(\*pend)->ptr = ph;**

**else //если очередь пуста**

**\*pbeg = ph;**

**\*pend = ph;**

**return 0;**

**}**

**//выбрать элемент из начала очереди, очередь не пуста**

**int del(Elem\* \*beg, Elem\* \*pend,){**

**int d = (\*beg)->key;**

**Elem\* ph = \*beg;**

**\*beg = (\*beg)->ptr;**

**if(!\*pbeg) //если очередь стала пустой**

**\*pend = 0;**

**delete ph;**

**return d;**

**}**

**int main(){**

**Elem\* pbeg = 0;**

**Elem\* pend = 0;**

**for(int i=1; i<=10; i++)**

**add(&pbeg,&pend,i);**

**while(pbeg)**

**printf("%i\n",del(&pbeg,&pend));**

**return 0;**

**}**

**3.3. Двунаправленный линейный список**

***Операции****:*

- создать (добавить первый)

- добавить в конец

- найти

- удалить

**typedef int Data;**

**struct Elem{**

**Data key;**

**Elem\* pnext;**

**Elem\* pprev;**

**};**

**//начальное формирование списка**

**//(создание первого элемента)**

**int first\_elem(Elem\* \*pbeg, Elem\* \*pend, Data d){**

**Elem\* ph = new Elem;**

**ph->key = d;**

**ph->pnext = 0;**

**ph->pprev = 0;**

**\*pbeg = ph;**

**\*pend = ph;**

**return 0;**

**}**

**//добавление элемента в конец списка, список не пуст**

**int add\_elem(Elem\* \*pend, Data d){**

**Elem\* ph = new Elem;**

**ph->key = d;**

**ph->pnext = 0;**

**ph->pprev = \*pend;**

**(\*pend)->pnext = ph;**

**\*pend = ph;**

**return 0;**

**}**

**//поиск элемента с заданным ключом**

**Elem\* find\_elem(Elem\* const pbeg, Data d){**

**Elem\* ph = pbeg;**

**while((ph)&&(ph->key != d))**

**ph = ph->pnext;**

**return ph;**

**}**

**//удаление элемента с заданным ключом**

**int del\_elem(Elem\* \*pbeg, Elem\* \*pend, Data k){**

**Elem\* pk = find\_elem(\*pbeg,k);//ищем то, что надо**

**if (pk){ //нашли то, что надо**

**if(pk == \*pbeg){ //удаляем первый**

**if(\*pbeg == \*pend){ //он же последний**

**\*pbeg = 0; \*pend = 0;**

**}else{**

**\*pbeg = (\*pbeg)->pnext;**

**(\*pbeg)->pprev = 0;**

**}**

**}else if (pk == \*pend){ //удаляем последний**

**\*pend = (\*pend)->pprev;**

**(\*pend)->pnext = 0;**

**}else{ //удаляем из середины**

**(pk->pprev)->pnext = pk->pnext;**

**(pk->pnext)->pprev = pk->pprev;**

**}**

**delete pk;**

**return 1;**

**}else //не нашли то, что надо**

**return 0;**

**}**

**int main(){**

**Elem\* pbeg = 0; Elem\* pend = 0;**

**char ch, s[81];**

**Data d;**

**do{**

**printf("1 - add to end, 2 - delete: ");**

**gets(s);**

**ch = s[0];**

**switch(ch){**

**case '1':**

**printf("Enter element for add: ");**

**gets(s);**

**d = atoi(s);**

**if(!pbeg)**

**first\_elem(&pbeg,&pend,d);**

**else**

**add\_elem(&pend,d);**

**break;**

**case '2':**

**printf("Enter element for delete: ");**

**gets(s);**

**d = atoi(s);**

**if(!del\_elem(&pbeg,&pend,d))**

**printf("Elem %i not found.\n",d);**

**break;**

**default:**

**printf("Error I/O.\n");**

**}**

**printf("List: ");**

**Elem\* ph = pbeg;**

**while(ph){**

**printf("%i ",ph->key);**

**ph = ph->pnext;**

**}**

**printf("\n");**

**printf("Repeat (Y/N)? ");**

**gets(s);**

**ch = toupper(s[0]);**

**}while(ch == 'Y');**

**return 0;**

**}**

**3.4’. Однонаправленный линейный список**

***Операции****:*

- создать (добавить первый)

- добавить в конец

- найти

- удалить

**typedef int Data;**

**struct Elem{**

**Data key;**

**Elem\* pnext;**

**~~Elem\* pprev;~~**

**};**

**//начальное формирование списка**

**//(создание первого элемента)**

**int first\_elem(Elem\* \*pbeg, Elem\* \*pend, Data d){**

**Elem\* ph = new Elem;**

**ph->key = d;**

**ph->pnext = 0;**

**~~ph->pprev = 0;~~**

**\*pbeg = ph;**

**\*pend = ph;**

**return 0;**

**}**

**//добавление элемента в конец списка, список не пуст**

**int add\_elem(Elem\* \*pend, Data d){**

**Elem\* ph = new Elem;**

**ph->key = d;**

**ph->pnext = 0;**

**~~ph->pprev = \*pend;~~**

**(\*pend)->pnext = ph;**

**\*pend = ph;**

**return 0;**

**}**

**//поиск элемента с заданным ключом**

**Elem\* find\_elem(Elem\* const pbeg, Data d){**

**Elem\* ph = pbeg;**

**while((ph)&&(ph->key != d))**

**ph = ph->pnext;**

**return ph;**

**}**

**//удаление элемента с заданным ключом**

**int del\_elem(Elem\* \*pbeg, Elem\* \*pend, Data k){**

**Elem\* pk = find\_elem(\*pbeg,k);//ищем то, что надо**

**if (pk){ //нашли то, что надо**

**if(pk == \*pbeg){ //удаляем первый**

**if(\*pbeg == \*pend){ //он же последний**

**\*pbeg = 0; \*pend = 0;**

**}**

**else{ //первый не последний**

**\*pbeg = (\*pbeg)->pnext;**

**~~(\*pbeg)->pprev = 0;~~**

**}**

**}**

**else{**

**//в остальных случаях надо хранить адрес узла,**

**//предшествующего удаляемому!**

**//Чтобы дважды не ходить по списку, перепишем эту**

**//функцию!**

**//удаление элемента с заданным ключом**

**int del\_elem(Elem\* \*pbeg, Elem\* \*pend, Data k){**

**Elem\* pk;**

**Elem\* ph = \*pbeg;**

**if (!\*pbeg) return 0; //список пуст**

**if(\*pbeg->key == k){ //удаляем первый**

**if(\*pbeg == \*pend){ //он же последний**

**\*pbeg = 0; \*pend = 0; delete ph;**

**}**

**else{ //первый не последний**

**\*pbeg = (\*pbeg)->pnext; delete ph;**

**}**

**}**

**else{ //удаляем не первый**

**while((ph->pnext)&&(ph->pnext->key != k))**

**ph = ph->pnext;**

**pk = ph->pnext;**

**if (pk){ //нашли, что удалять**

**ph->pnext = pk->pnext;**

**delete pk;**

**}**

**else //не нашли то, что надо**

**return 0;**

**}**

**return 1;**

**}**

**I вариант:**

1. ОБЪЯВИТЬ функцию выделения памяти для двумерной   
таблицы.

тип элементов таблицы: char;

тип возвращаемого функцией значения: код ошибки;

параметры функции: x – число строк, y – число столбцов, T – таблица;

имя функции: new\_T.

2. Открыть файл t.txt для записи. Записать в него в столбик 123 целочисленных значения, сгенерированных датчиком случайных чисел в диапазоне от -5 до 7. Закрыть файл.

**II вариант:**

1. ОБЪЯВИТЬ функцию выделения памяти для двумерной   
таблицы.

тип элементов таблицы: double;

тип возвращаемого функцией значения: адрес таблицы;

параметры функции: a – число строк, b – число столбцов;

имя функции: Table\_new.

2. Открыть файл a.txt для записи. Записать в него в столбик 175 целочисленных значений, сгенерированных датчиком случайных чисел в диапазоне от -8 до 3. Закрыть файл.

**I вариант:**

**int new\_T(int x, int y, char\*\* \*T);**

**FILE \*f = fopen("t.txt","w");**

**srand(time(0));**

**for(int i = 1; i <= 123; i++){**

**printf("i\n",(rand() % (7-(-5)+1))-5);**

**}**

**fclose(f);**

**II вариант:**

**double\*\* Table\_new(int a, int b);**

**FILE \*f = fopen("a.txt","w");**

**srand(time(0));**

**for(int i = 1; i <= 175; i++){**

**printf("i\n",(rand() % (3-(-8)+1))-8);**

**}**

**fclose(f);**

**3.4. Бинарное дерево**

***Операции***:

- найти

- добавить (вставить)

- удалить

- обойти дерево

**struct Elem{**

**int key;**

**Elem\* pleft;**

**Elem\* pright;**

**};**

**//вставка узла**

**Elem\* tree\_insert(Elem\* \*root, int d){**

**Elem\* ptr = \*root;**

**if (!ptr){ //дерево пусто**

**Elem\* pnew = new Elem;**

**pnew->key = d;**

**pnew->pleft = 0;**

**pnew->pright = 0;**

**\*root = pnew;**

**ptr = pnew;**

**}**

**else{ //дерево не пусто**

**int key\_insert = 1;**

**while(key\_insert)**

**if (d < ptr->key)**

**if(!ptr->pleft){ //нет левого сына**

**Elem\* pnew = new Elem;**

**pnew->key = d;**

**pnew->pleft = 0;**

**pnew->pright = 0;**

**ptr->pleft = pnew;**

**ptr = pnew;**

**key\_insert = 0;**

**}else //поиск места д/вставки в левом поддереве**

**ptr = ptr->pleft;**

**else if (d > ptr->key)**

**if(!ptr->pright){//нет правого сына**

**Elem\* pnew = new Elem;**

**pnew->key = d;**

**pnew->pleft = 0;**

**pnew->pright = 0;**

**ptr->pright = pnew;**

**ptr = pnew;**

**key\_insert = 0;**

**}else //поиск места д/вставки в правом поддереве**

**ptr = ptr->pright;**

**else**

**key\_insert = 0;//такой узел уже есть**

**}**

**return ptr;**

**}**

**//удаление узла, необходимо помнить родителя удаляемого узла**

**int tree\_del(Elem\* \*root, int d){**

**Elem\* ptr = \*root; //ptr - узел для удаления**

**Elem\* pprev = 0; //pprev - родитель удаляемого узла**

**//поиск узла для удаления**

**int key\_find = 0;**

**while((ptr)&&(!key\_find))**

**if (d < ptr->key){//поиск в левом поддереве**

**pprev = ptr;**

**ptr = ptr->pleft;**

**}else if (d > ptr->key){//поиск в правом поддереве**

**pprev = ptr;**

**ptr = ptr->pright;**

**}else**

**key\_find = 1;//нашли**

**if (!ptr) return 1;//узел для удаления не найден**

**if(!ptr->pright){**

**//у удаляемого узла нет правого поддерева**

**if(pprev) //удаляем не корень**

**if(d < pprev->key) //удаляемый узел был левым сыном**

**pprev->pleft = ptr->pleft;**

**else //удаляемый узел был правым сыном**

**pprev->pright = ptr->pleft;**

**else //удаляем корень**

**\*root = ptr->pleft;**

**delete ptr;**

**}**

**else if((ptr->pright)&&(!ptr->pright->pleft)){**

**//у правого сына удаляемого узла нет левого поддерева**

**ptr->pright->pleft = ptr->pleft;**

**if(pprev) //удаляем не корень**

**if(d < pprev->key)//удаляемый узел был левым сыном**

**pprev->pleft = ptr->pright;**

**else //удаляемый узел был правым сыном**

**pprev->pright = ptr->pright;**

**else //удаляем корень**

**\*root = ptr->pright;**

**delete ptr;**

**}**

**else{**

**//у правого сына удаляемого узла есть левое поддерево**

**Elem\* ptr1 = ptr->pright;**

**Elem\* ptr2 = ptr;**

**while(ptr1->pleft){**

**ptr2 = ptr1;**

**ptr1 = ptr1->pleft;**

**}**

**ptr2->pleft = ptr1->pright;**

**ptr->key = ptr1->key;**

**delete ptr1;**

**}**

**return 0;**

**}**

**//обход дерева - печать - рекурсия**

**int tree\_print(Elem\* p, int level){**

**if(p){**

**tree\_print(p->pright,level+1); //правое поддерево**

**for(int i=0; i<level; i++)**

**printf(" ");**

**printf("%i\n",p->key); //корень**

**tree\_print(p->pleft,level+1); //левое поддерево**

**}**

**return 0;**

**}**

**int main(){**

**Elem\* root = 0;**

**int data[] = {20,18,44,36,5,9,3,33,22,4};**

**for(int i=0; i<10; i++)**

**tree\_insert(&root, data[i]);**

**printf("Tree:\n");**

**tree\_print(root,0);**

**for(int i=0; i<10; i++){**

**printf("\nDelete %i:\n",data[i]);**

**tree\_del(&root, data[i]);**

**tree\_print(root,0);**

**}**

**printf("OK!\n");**

**return 0;**

**}**

**I вариант:**

**// «Добавить элемент в непустой стек»**

**// Elem\* top; – указатель на вершину стека**

**// int d; – добавляемое значение**

**Elem\* ph = new Elem;**

**ph->key = d;**

**ph->ptr = top;**

**top = ph;**

**II вариант**

**// «Добавить элемент в конец непустой очереди»**

**// Elem\* pend; – указатель на конец очереди**

**// int d; – добавляемое значение**

**Elem\* ph = new Elem;**

**ph->key = d; ph->ptr = 0;**

**pend->ptr = ph;**

**pend = ph;**

**ЗАДАНИЕ: ДЛЯ КАЖДОЙ СТРОКИ КОДА НАРИСОВАТЬ ОТДЕЛЬНЫЙ РИСУНОК!**