7. Указатели и массивы. Динамические программные объекты

Любой байт памяти компьютера обладает **адресом**, через который может осуществляться доступ к этому байту. Адрес выражается, как правило, шестнадцатеричным числом. **Ячейка памяти** для хранения какой-либо величины может состоять из нескольких байт (размер определяется типом величины), в этом случае адресом ячейки является адрес первого составляющего её байта. Все обращения в программе к величине по её имени заменяются компилятором на адрес ячейки памяти, в которой хранится значение величины. Для явного взятия адреса величины имеется операции & (к примеру, &x означает адрес величины x), На языке C/C++ есть возможность определять собственные переменные для хранения адресов областей памяти. Такие переменные называются **указателями**. Указатель не является самостоятельным типом, он всегда связан с каким-либо другим конкретным типом.

В C++ различают три вида указателей: указатель на объект, указатель на функцию и указатель на void (подробности см. в [2]). В настоящем разделе будут рассмотрены указатели на объект, которые будем здесь называть просто указателями.

Указатель на объект содержит адрес области памяти, в которой хранятся данные определённого типа. Простейшее объявление указателя имеет вид:

*тип* \**имя\_указателя*;

где *тип* может быть любым, кроме ссылки и битового поля (см. [2]). Бывают указатели-константы (значения которых нельзя изменять), бывают указатели на константу, бывают указатели-константы на константу — все их следует различать. При определении указателя желательно сразу же выполнить его инициализацию (присвоение начального значения) — по тем же правилам записи, что и для обычных величин (т. е. через знак равенства или в скобках). Вот примеры объявления и инициализации указателей:

int n=5; //целая переменная

const int cm=7; //целая константа

int \*pn=&n; //указатель на целую перем.

int \*rn=pn; //указатель на целую перем.,

//инициализированный

//значением другого указателя

const int \*pcm; //указатель на целую константу

//(без инициализации)

int \* const cpn=&n; //указатель-константа на

//целую переменную

const int \* const cpcm=&cm; //указ.-константа

//на целую конст.

Если указатель содержит адрес реального объекта (указывает на объект), то этот адрес отличен от нуля. Однако бывает так, что указатель не указывает ни на какой объект, т. е. является **пустым (нулевым) указателем**. Тогда его значение обозначается именем NULL, в современных компиляторах допускается также писать 0.

Над указателями допускается производить следующие **операции**:

**1. Разадресация (разыменование).** Эта операция предназначена для доступа к величине, адрес которой хранится в указателе. Пишется как унарная \*, слева от операнда. Операции \* и & взаимно обратны: если, к примеру, p есть адрес величины x (т. е. &x), то \*p — это есть сама величина x.

**2. Присваивание** — например, другому указателю или конкретному шестнадцатеричному адресу.

**3.** Некоторые **арифметические операции** — такие как сложение с целым числом, инкремент (++), декремент (--), нахождение разности указателей, сравнение указателей. Они требуют отдельного пояснения.

Память компьютера логически представляет собой длинную цепочку байтов, от первого до последнего. Каждый указатель связан с конкретным типом, а каждый тип имеет свой размер — количество байт, которое отводится в памяти для величин данного типа (и есть специальная функция sizeof с аргументом — наименованием типа, возвращающая размер этого типа). Если в каком-либо фрагменте памяти последовательно размещены величины одного типа, то выстраивается цепочка ячеек памяти, размера этого типа (напоминаем, что адрес ячейки — это адрес её начального байта). Тогда если имеется указатель (того же типа), указывающий на какую-либо из этих ячеек, то можно определить понятие «смещение указателя» — вперёд или назад (в сторону увеличения или уменьшения шестнадцатеричных адресов), на какое-то количество позиций (ячеек). Иными словами: если p — указатель, а n — целое число, то p+n — это указатель, смещённый на n позиций (n\*sizeof(*тип*) байт) вперёд. Соответственно, p-n — указатель, смещённый на n позиций (n\*sizeof(*тип*) байт) назад. Определив таким образом операцию сложения указателя с целым числом (положительным или отрицательным), несложно уже определить операции ++, -- (как сложение с единицей и вычитание единицы), разность указателей (обратная операция к сложению с целым числом, результат есть целое число — «расстояние» между двумя участками памяти, измеряемое в ячейках), сравнение указателей.

Указатели обычно используются при работе с **динамической памятью** — свободной памятью, в которой можно во время выполнения программы выделять место по мере необходимости. Доступ к выделенным участкам динамической памяти (называемым **динамическими переменными**) может производиться только через указатели. В языке C++ выделение участка динамической памяти производится с помощью операции new, освобождение этого участка — с помощью операции delete. Так, к примеру, операция

int \*d = new int;

выполняет выделение достаточного для размещения величины типа int участка динамической памяти и записывает адрес этого участка в переменную d. Доступ к самой динамической переменной осуществляется с помощью операции разадресации: \*d. Операция

delete d;

освобождает выделенный ранее участок динамической памяти, и доступ к динамической переменной прекращается.

В качестве динамических переменных наиболее часто выступают сложные объекты, состоящие из компонент, к изучению которых мы сейчас будем приступать.

В реальных задачах, из разных областей человеческой деятельности, часто возникают ситуации, когда имеется группа сходных по смыслу величин с одним общим наименованием и отличающихся друг от друга по некоторой дополнительной характеристике — индексу. Таким, к примеру, будет следующая группа из 10 величин с индексами:

*a*1, *a*2, *a*3, … , *a*10

В языках программирования для подобных групп имеется специальное понятие — массив.

Итак, **массивом** называется именованный комбинированный объект, состоящий из однотипных **компонент** или **элементов**. При этом каждая компонента обладает **индексом**. В языке C/C++ индексы допускаются только целочисленные и начинающиеся с нуля. Каждый массив обладает **размером** (количеством компонент), в памяти компьютера компоненты массива располагаются последовательно друг за другом, т. е. занимают связный участок.

**Описание массивов** выглядит следующим образом:

*тип\_компонент имя\_массива*[*размер\_массива*];

где *размер\_массива* — это целая константа или константное выражение.

В частности, вышеупомянутый массив 10 величин *ai* может быть описан так (при условии, что все *ai* имеют тип float):

float a[10];

При желании можно использовать **оператор определения типов** typedef:

typedef float mass[10];

mass a;

Здесь определён и использован новый тип mass, означающий массив из 10 вещественных элементов. Так удобно делать, если этот тип упоминается в программе неоднократно. Общий вид оператора typedef для массивов:

typedef *тип\_компонент имя\_нового\_типа*[*размер*];

Обращение к i‑му элементу массива (*ai*) делается путём указания его индекса в квадратных скобках после имени массива: a[i].

***Внимание!*** Поскольку индексация массива начинается с нуля, индекс последнего элемента массива на единицу меньше его размера. В частности, 10 элементов *ai* — это величины *a*[0], *a*[1], *a*[2], … , *a*[7], *a*[8], *a*[9].

Возможно описание массива с инициализацией. Начальные значения компонент при этом перечисляются в фигурных скобках через запятую, в порядке возрастания индексов:

int b[5]={9,5,1,6,3};

int t[6]={5,7,1,9};

В частности, *b*[0]=9, *b*[1]=5, *b*[3]=6. Для массива *t* размер объявлен 6, но перечислены только 4 компонента. Это означает, что оставшиеся 2 компонента (в конце массива) автоматически заполнятся нулями:

*t*[0]=5, *t*[1]=7, *t*[2]=1, *t*[3]=9, *t*[4]=0, *t*[5]=0.

Следующий пример показывает, как можно объявить массив, у которого все начальные значения нулевые:

int f[20]={};

Если при инициализации массива перечисляются все его элементы, то размер массива можно не указывать (оставить пустые скобки): он автоматически установится по количеству элементов. Так, следующие два описания будут эквивалентны:

int x[3]={5,1,9};

int x[]={5,1,9};

*Замечание 1.* Для размера массива предпочтительно использовать именованные константы, поскольку в этом случае для его изменения достаточно скорректировать значение только в одном месте программы.

*Замечание 2.* В современных версиях допускается объявление массива с не-константным размером. Однако в этом случае его нельзя инициализировать.

**Адрес массива** — это адрес начального байта выделенного для него непрерывного участка памяти. Совпадает с адресом его начального элемента (с индексом 0) и записывается как одиночное имя массива, без квадратных скобок (т. е. имя массива без квадратных скобок — это есть указатель на начало массива). А поскольку в памяти компьютера все элементы массива расположены друг за другом, адрес каждого из них может быть найден путём применения операции смещения указателя (сложения с целым числом). К примеру, если имеется описание

const int N=15;

int x[N];

то x — это то же самое, что и &x[0], x+1 — то же, что и &x[1], x+2 — то же, что и &x[2] и т. д. Это означает, что любой x[i] может быть записан через операцию разадресации, как \*(x+i).

Работа с массивами подразумевает применение циклов, с участием параметра, проходящего по индексам массива. В частности, ввод значений элементов вышеупомянутого массива x (в стиле C) может быть сделан одним из следующих способов:

1) с запросом на каждый элемент:

for(int i=0;i<N;i++) {

printf(”x[%d]=”,i);

scanf(”%d”,&x[i]);}

2) с запросом на весь массив сразу:

printf(”задайте %d чисел\n”,N);

for(int i=0;i<N;i++) scanf(”%d”,&x[i]);

Аналогично, вывод массива можно записать так:

1) в строку:

for(int i=0;i<N;i++) printf(”%d ”,x[i]);

2) в столбец:

for(int i=0;i<N;i++) printf(”%d\n”,x[i]);

***Внимание!*** При работе с массивами следует контролировать возможный выход (по индексам) за пределы массива. Автоматический контроль в компиляторах отсутствует.

**Пример 1.** В целочисленном массиве из 12 элементов все чётные элементы удвоить, нечётные — обнулить.

#include <stdio.h>

int main() {

const int N=12;

int a[N];

printf(”задайте %d чисел\n”,N);

for(int i=0;i<N;i++) scanf(”%d”,&a[i]);

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

if(a[i]%2==0) a[i]\*=2; else a[i]=0;

for(int i=0;i<N;i++) printf(”%d\n”,a[i]);

return 0;}

**Динамические массивы** создаются с помощью операции new, при этом указывается тип элементов и размер:

int n=100;

float \*p = new float[n];

Здесь в динамической памяти выделяется связная область, достаточная для размещения 100 элементов типа float, а адрес её начала записывается в указатель p. Динамические массивы нельзя инициализировать, и они не обнуляются. Доступ к элементам динамических массивов осуществляется точно так же, как и к статическим, через указание индекса: p[i] или \*(p+i). Память, зарезервированная под динамический массив, освобождается оператором delete с участием пустых квадратных скобок:

delete[] p;

Выделенную память под динамические объекты надо освобождать по мере того, как отпадает необходимость присутствия этих объектов. Если этого не делать, то может произойти «замусоривание» памяти «отработанными» объектами, к которым прекращён доступ.

Если размер массива есть переменная величина и возникает только в процессе работы программы, то этот массив предпочтительно описывать как динамический. Вот, в частности, вариант Примера 1 для массива, размер которого заранее не известен:

**Пример 1a.** Даны: целое *N* и целочисленный массив *a* из *N* элементов. Все чётные элементы в этом массиве удвоить, нечётные — обнулить.

#include <stdio.h>

int main() {

int N;

printf(”задайте N\n”);

scanf(”%d”,&N);

int \*a=new int[N];

printf(”задайте %d чисел\n”,N);

for(int i=0;i<N;i++) scanf(”%d”,&a[i]);

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

if(a[i]%2==0) a[i]\*=2; else a[i]=0;

for(int i=0;i<N;i++) printf(”%d\n”,a[i]);

delete[] a;

return 0;}

**Пример 2. Нахождение наибольшего элемента в массиве.** В массиве *x* из 15 элементов найти *max* — значение наибольшего элемента.

Поскольку *max* — это значение одного из элементов массива, ход действий будет таким: сначала временно положим max одному из элементов (например, *x*[0]), а затем будем уточнять его значение путём сравнения его с каждым из остальных элементов. Если очередной элемент *x*[*i*] окажется больше временного max, мы обновим значение *max* на значение этого *x*[*i*]. В конце прохода в *max* окажется записанным значение наибольшего элемента массива.

#include <iostream>

using namespace std;

int main() {

const int N=15;

float x[N];

cout<<”задайте ”<<N<<” чисел\n”;;

for(int i=0;i<N;i++) cin>>x[i];

float max=x[0];

for(int i=1;i<N;i++) if(x[i]>max) max=x[i];

cout<<”max=”<<max<<”\n”;

return 0;}

*Замечание.* Стартовый *max* можно определять не обязательно как *x*[0], а как любой из элементов, среди которых ищем наибольший. Только в этом случае цикл прохода по массиву с обновлением *max* надо проводить при *i* не от 1, а от 0.

***Упражнение.*** Как можно найти минимальный элемент массива *x*?

**Пример 2a.** Предположим, что максимальный элемент встречается в массиве только один раз и надо ещё определить его индекс *imax*. Дополняем программу следующим образом: определяем целую переменную *imax*, которая в каждый момент будет означать индекс элемента *max* в массиве *x*: как только присваиваем *max* значение какого-либо *x*[*i*], тут же *imax* присваиваем значение *i*.

#include <iostream>

using namespace std;

int main() {

const int N=15;

float x[N];

cout<<”задайте ”<<N<<” чисел\n”;;

for(int i=0;i<N;i++) cin>>x[i];

float max=x[0]; int imax=0;

for(int i=1;i<N;i++)

if(x[i]>max) {max=x[i]; imax=i;}

cout<<”max=”<<max<<”\n”;

cout<<”imax=”<<imax<<”\n”;

return 0;}

***Упражнение.*** Если максимальный элемент встречается в массиве несколько раз, индекс какого из них выйдет в качестве результата в этой программе?

**Пример 3.** Даны: *n*, массив *a* из *n* элементов. Переставить элементы массива таким образом, чтобы они шли в обратном порядке (другого массива не заводить).

Для этой задачи потребуется действие по замене значениями двух величин. Это действие производится с помощью вспомогательной буферной переменной, в которую будет записываться значение той величины, которая обновляется первой. Для абстрактных *x* и *y*, которые нужно обменять значениями, это записывается так:

r=x;

x=y;

y=r;

В нашем примере в качестве *x* будет присутствовать *a*[*i*], в качестве *y* — симметричный ему элемент относительно середины массива *a*[*n*–*i*]. Однако ошибочно было бы организовывать цикл обмена значениями по всем индексам, от 0 до *n*–1. Поскольку реальная замена закончится на середине прохода, потом начнётся обратная замена, и в результате массив останется как был. Следовательно, этот цикл должен быть завершён на уровне середины массива.

#include <iostream>

using namespace std;

int main() {

int n;

cout<<”задайте n\n”;

cin>>n;

int \*a=new float[n];

cout<<”задайте ”<<n<<” чисел\n”;;

for(int i=0;i<n;i++) cin>>a[i];

float r;

for(int i=0;i<n/2;i++)

{r=a[i]; a[i]=a[n-i]; a[n-i]=r;}

for(int i=0;i<n,i++) cout<<a[i]<<endl;

delete[] a;

return 0;}

**Задачи**

1. Дан массив из 12 вещественных элементов. Вычислить:

a) сумму всех элементов;

b) сумму всех положительных элементов;

c) количество положительных элементов;

d) сумму элементов с нечётными индексами.

1. Даны два 10‑элементных массива *a* и *b*. Вычислить



1. Даны натуральное число *n* и массив целых чисел *А* из *n* элементов. Определить, образуют ли элементы массива неубывающую последовательность.
2. Дан массив *A* размера *N*. Найти максимальный элемент из его элементов с нечетными номерами: *A*1, *A*3, *A*5, … .
3. В массиве из *n* элементов вычислить наименьший элемент от начала массива до первого нулевого элемента (если нулевых элементов нет — до конца массива).
4. Даны *n* точек плоскости (*хi*, *yi*). Найти радиус наименьшего круга с центром в начале координат, содержащего все эти точки.
5. В массиве из *n* чисел найти первый отрицательный элемент и его индекс в массиве. Если отрицательных элементов нет, вывести об этом сообщение.
6. В массиве из *n* элементов вычислить сумму элементов от первого положительного и до конца массива (если положительных элементов нет, то вывести 0).
7. Дан массив ненулевых целых чисел размера *N*. Проверить, чередуются ли в нем положительные и отрицательные числа. Если чередуются, то вывести 0, если нет, то вывести порядковый номер первого элемента, нарушающего закономерность.
8. Дан массив *x* из 55 элементов. Вычислить

