РОЗДІЛ 1.
МАСИВИ ТА ВКАЗІВНИКИ

1.1. Масив як змінна

Поняття масиву

У багатьох задачах у пам'яті програми треба зберігати великі набори однотипних даних. Для цього використовують масиви.

> **Масив** – це змінна, утворена послідовністю змінних, які називаються **елементами**, є однотипними й ідентифікуються номерами (**індексами**). Елементи займають послідовні ділянки пам'яті й до них є **прямий доступ**: будь-який елемент масиву доступний за допомогою його індексу.

Означення масиву має вигляд *T a[n]*, де *T* – тип елементів, *a* – ім'я масиву, *n* – *цілий константний вираз*, що задає довжину масиву. Елементи можуть мати довільний скалярний або структурний тип (масив та інші, про які йдеться в наступних розділах). Індексами елементів є цілі числа від `0` до *n*–1.

- ✓ Тип і кількість елементів масиву (його **довжина**) фіксуються в означенні або під час створення й далі не змінюються.
- ✓ Константу, що визначає довжину масиву, рекомендується іменувати.

Елемент масиву ідентифікується іменем масиву й індексом. Наприклад, елемент масиву зі ста елементів, `int a[100]`, позначається виразом вигляду `a[`*IE*`]`, де вираз *IE* повинен мати ціле значення від `0` до 99.

- ✓ Зазвичай програміст повинен сам стежити, щоб індекс елемента був у межах масиву. Вихід індексу за межі масиву може мати *непередбачувані наслідки*.

Приклади

1. Розглянемо інструкції, які дають ім'я `N` довжині 10, означають масив з `N` цілих елементів, присвоюють їм послідовні значення від 0 до 9 й виводять квадрати цих значень на екран.

```
const int N=10;
int a[N], i;
for (i=0; i<N; ++i)
  { a[i]=i; cout << a[i]*a[i] << ' '; }
```

2. Програма отримує мільйон псевдовипадкових цілих чисел від 0 до 99 й виводить кількість поява кожного з них. Числа утворюються як остачі від ділення на 100 значень, отриманих від бібліотечної функції `rand()`. Для роботи з нею потрібен файл `<cstdlib>`. "Зерно", яке визначає послідовність чисел, задається користувачем. Кожне число `rand()%100` використовується як індекс у масиві лічильників `cnt`, а кількість його появ стає значенням `cnt[rand()%100]`. Накопичені кількості виводяться.

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main() {
  const int MLN=1000000;
  const int N=100;
  int k, seed;
  cout << "Enter int seed>"; cin >> seed;
  srand(seed);
  int cnt[N]; for(k=0; k<N; ++k) cnt[k]=0;
  for(k=0; k<MLN; ++k)     //утворення та
    ++cnt[rand()%100];     //підрахунок чисел
  for(k=0; k<N; ++k)
    cout << k << ' ' << cnt[k] << endl;
  system("pause"); return 0;
}
```

◂

✓ Деякі, але не всі, компілятори дозволяють задати довжину масиву *ім'ям змінної* після того, як вона отримала значення, наприклад, так: `int n; cin>>n; int cnt[n];`. Проте покладатися на цю можливість *не будемо*.

Операція `sizeof`, застосована до імені масиву, дає *розмір масиву* в байтах. Наприклад, якщо є масив `int a[5]`, то `sizeof a` має значення 20 = 5×4, адже `sizeof int` дорівнює 4. Відповідно значенням виразу `sizeof(a)/sizeof(int)` є 5 – довжина масиву.

На практиці насправді може оброблятися лише частина елементів масиву, розташованих зазвичай на його початку. Звідси часто під довжиною масиву розуміють саме довжину послідовності значень на початку масиву, що утворюються та обробляються. Найчастіше цю реальну довжину зберігають в окремій змінній. Приклади наведено нижче.

Ініціалізація масиву

Значення, що ініціалізують послідовні елементи масиву, задаються списком констант у дужках {}, наприклад `int a[4]={2,10,3,15};`. Якщо констант менше, ніж елементів у масиві, то вони присвоюються першим елементам масиву, а решта елементів отримують нульові значення відповідного типу. Наприклад, означення `int cnt[4]={};` ініціалізує всі елементи значенням 0. Якщо констант у списку більше, ніж елементів масиву, то це є помилкою.

В ініціалізації можна не вказувати довжину масиву – вона стає рівною кількості констант. Наприклад, означення `int a[]={9,8,7};` задає масив із трьох елементів.

Якщо ініціалізацію не задано, то статичні змінні (зокрема елементи масивів, означених за межами функцій) як початкові значення отримують нулі відповідних типів. Значеннями автоматичних змінних, у тому числі й елементів масивів, є випадкове "сміття".

Приклад обробки масиву

Числа Фібоначчі було введено для підрахунку кількості пар кролів за такого припущення. Кожна пара кролів приносить щорічно приплід в одну пару (самку й самця), які, у свою чергу, починають давати приплід через два роки після народження. Смертністю кролів нехтують. Якщо спочатку є одна пара новонароджених кролів, то через n років буде F_{n+1} пар. Змінимо цю модель, вважаючи, що кожна тварина живе 4,5 роки, та обчислимо, скільки пар кролів і якого віку буде через n років, де n – задане натуральне число.

Запишемо в рядок початкові кількості кролів, вік яких 0 років (новонароджених), 1 рік, 2, 3 й 4 роки. У наступні рядки – кількості через рік, два роки тощо.

```
спочатку     : 1 0 0 0 0
через 1 рік  : 0 1 0 0 0
через 2 роки : 1 0 1 0 0
через 3 роки : 1 1 0 1 0
через 4 роки : 2 1 1 0 1
через 5 років: 2 2 1 1 0
```

Неважко зрозуміти, що кількість кролів віку i ($i > 0$) – це кількість кролів віку i–1 у попередній рік, а кількість новонароджених є сумою кількостей кроликів, вік яких становить 2, 3 й 4 роки.

Отже, промоделюємо життя кролів за допомогою масиву `int r[5]`. Індекс елемента зображує вік кролів, значення – кількість кролів цього віку. Ініціалізуємо масив значеннями {1, 0, 0, 0, 0}. Далі його обробка здійснюється в циклі по роках. На кожному кроці спочатку значення `r[3]` копіюється в `r[4]`, потім `r[2]` – в `r[3]`, `r[1]` – в `r[2]`, `r[0]` – в `r[1]` (саме в такому порядку!). Елемент `r[0]` отримує значення `r[2]+r[3]+r[4]`.

```
#include <iostream>
using namespace std;
int main()
{ int years;
  cout<<"Enter number of years (>0):\n";
  cin>>years; if(years<=0) return 1;
  const int N=5;     // довжина масиву
  int r[N] = {1};    // r[0]=1, решта нулі
  int y, i;          // рік і вік
  for(y=1; y<=years; ++y){
    for(i=4; i>0; --i)
      r[i]=r[i-1];
    r[0]=r[2]+r[3]+r[4];
  }
  for(i=0; i<N; ++i)
    cout << r[i] << " ";
  cout << endl;
  return 0;
}
```

Вправи

1.1. Модифікувати наведену вище програму про кролів за умови:
   а) кролі живуть не 4,5 роки, а 12,5 років;
   б) кількість кролів виводиться щорічно.

1.2. Риби народжуються навесні й живуть не довше 9,5 років. Навесні на кожну рибу припадає в середньому B новонароджених мальків. Кількість риб, незалежно від їх віку, за рік (від весни до весни) зменшується в D разів. Навесні першого року у водоймище випустили M новонароджених

мальків. Написати програму обчислення, скільки риби й якого віку буде у водоймищі через *Y* років.

1.3. За заданим роком і номером дня в році (від 1 до 365 або до 366, якщо рік високосний) обчислити дату (число, місяць). Наприклад, за 2012 61 і за 2013 60 обчислюється 1 3 – перше березня.

1.4. За заданою датою (число, місяць, рік, наприклад, 6, 5, 2012) обчислити номер дня в році (від 1 до 365 або 366, якщо рік високосний).

1.5. За допомогою генератора псевдовипадкових чисел утворити послідовність цілих чисел у діапазоні від 140 до 220 (см), що виражають зріст студентів. Вивести кількість студентів кожного можливого зросту.

1.2. Типізовані вказівники

У цьому підрозділі наведено мінімальні відомості про вказівники. Головним їх призначенням є робота з динамічними даними (див. розд. 4, 7).

Адреси та вказівники

Оперативну пам'ять, доступну програмі, можна розглядати як послідовність байтів; у кожного її байта є номер – **адреса**. Кожна змінна займає послідовні байти пам'яті, кількість яких визначається типом змінної. **Адреса змінної** – це адреса її першого байта.

Вказівник – це змінна, можливими значеннями якої є адреси.

Будемо розглядати лише **типізовані вказівники** – їхніми значеннями можуть бути адреси даних тільки певного типу. **Тип адрес** даних типу *T* позначається виразом *T**. При цьому тип *T* називається **базовим**, а вказівник типу *T** – **вказівником на дані типу** *T* (**вказівником типу** *T*).

✓ У 32-розрядних середовищах програмування вказівник *будь-якого типу займає чотири байти*, у 64-розрядних – вісім байтів.

Термін "вказівник" – це переклад англійського ***P**ointer*, тому варто дотримуватися неписаного правила: у кінці або на початку імен вказівників присутня літера p, а імена типів вказівників (про це йдеться в наступному підрозділі) починаються з P.

Символ * є окремою лексемою й в оголошенні вказівника стосується імені цього вказівника, а не типу.

Приклад. Інструкції оголошення

```
int * ip1, * ip2, i2; char *chp; char ** chpp;
```

визначають: змінні ip1, ip2 – це вказівники на цілі числа, i2 – ціла змінна, chp – вказівник на символи, chpp – вказівник на вказівники на символи.

Підкреслимо: наведені означення *не надають змінним значень*. Якщо ці змінні є автоматичними, то їхні значення – це випадкове "сміття". ◂

Розглянемо, як вказівнику присвоїти значення. Перша можливість – надати йому адресу деякої змінної. Адресу змінної позначає вираз із **операцією взяття адреси** &, наприклад, &x. Якщо адресу змінної x присвоєно вказівнику p виразом p=&x, то кажуть, що вказівник p *встановлено на змінну* x. Тоді значенням виразу p є адреса змінної. Присвоїти значення p іншому вказівнику означає встановити його на ту саму змінну.

Приклад. Інструкції

```
int x, *ip1 = &x, * ip2; ip2 = ip1;
char ch, *chp = &ch, ** chpp = &chp;
```

встановлюють вказівники ip1, ip2 на змінну x, chp – на ch, chpp – на chp (рис. 1.1). ◂

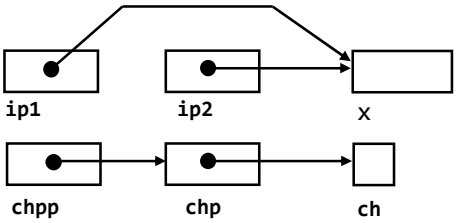

Рис. 1.1. Встановлення вказівників

Вираз *p з операцією **розіменування вказівника** * позначає дані, на які встановлено p, тобто *ідентифікує* їх.

Приклади

1. Після встановлень вказівників, як на рис. 1.1, вирази *ip1, *ip2 позначають змінну з іменем x, а вирази *chp, **chpp є си-

нонімами імені ch, тому кожен з виразів ch='a', *chp='a', **chpp='a' присвоює значення 'a' змінній ch.

2. У наведеній програмі функція ідентифікує дві цілі змінні своїми параметрами-вказівниками та обмінює місцями значення змінних. У функції вважається, що аргументи в її виклику дійсно задають адреси двох цілих змінних.

```
#include <iostream>
using namespace std;
void swapByPtrs(int * p1, int * p2){
  int t = *p1; *p1 = *p2; *p2 = t;
  return;
}
int main(){
  int a=1, b=2;
  swapByPtrs(&a, &b);
  cout << a << " " << b << endl; // вихід: 2 1
  system("pause");
  return 0;
}
```

Ім'я вказівника й вираз вигляду *&x*, де *x* – ім'я змінної, є найпростішими **адресними виразами** – виразами, значеннями яких є адреси. У мові C++ є також адресна константа NULL, яка позначає адресу 0. Ця адреса не може бути адресою жодних даних, тому найчастіше використовується як ознака того, що вказівник на жодні дані не встановлено. Складніші адресні вирази розглядаються нижче.

✓ Якщо *AE* – адресний вираз, то вираз вигляду **AE* позначає дані, адреса яких є значенням виразу *AE*.

Оголошення імені типу вказівників

Типу можна дати ім'я за допомогою **інструкції оголошення імені типу** typedef такого загального вигляду[1]:

```
typedef  вираз_що_задає_тип  ім'я;
```

Виразом, що задає тип, може бути ім'я стандартного або іншого типу, оголошеного програмістом, або складніший вираз, що описує

[1] Насправді вигляд може бути й іншим, але в даному випадку це не важливо.

тип. Також зазначимо, що в мові C++ таке оголошення задає не новий тип, а нове ім'я вже існуючого типу.

Приклади

1. Після інструкції

```
typedef int * PInt;
```

ім'я `PInt` позначає тип вказівників на цілі типу `int`. Це дозволяє оголошувати вказівники без знаків * перед іменами.

```
PInt pi1, pi2;
```

Оголошення імені типу вказівників гарантує від такої "дитячої" помилки, коли мали оголосити два вказівники, а оголосили вказівник `pp1` і цілу змінну `pp2`.

```
int * pp1, pp2;
```

2. Оголосимо імена типів вказівників на символи й вказівників на вказівники на символи.

```
typedef char * PChar;
typedef char ** PPChar;
```

Далі встановимо два вказівники й двічі виведемо символ `'a'`.

```
char ch = 'a';
PChar p1 = &ch;
PPChar p2 = &p1;
cout << *p1 << **p2;
```

◀

Імена типів вказівників замість виразів із * спрощують вигляд заголовків функцій, зокрема оголошення параметрів-посилань, що є вказівниками. Порівняйте такі прототипи.

```
int * f1(int * & p);
PInt f2(PInt& p);
```

Вони допустимі й еквівалентні, але другий виглядає простіше.

Арифметичні дії з адресами

Мова C++ дозволяє додати до адреси або відняти від неї ціле число. Можна також до цілого числа додати адресу. Отже, якщо *A* та *I* позначають, відповідно, адресний і цілий вирази, то *A+I*, *A–I*, *I+A* є адресними виразами.

Якщо доданок-адреса має тип `T*`, то ціле значення розглядається як *кількість одиниць даних типу* `T`. Наприклад, якщо `x` – змінна типу `int`, то адресні вирази `&x+1` і `1+&x` задають адресу цілої змінної, розташованої відразу після змінної `x`. Ця адреса

більше адреси змінної x на sizeof(int), тобто на 4. Віднімання цілого від адреси аналогічне. Так, значенням виразу &x-1 є адреса змінної, яка в пам'яті займає місце безпосередньо перед x.

Звідси, якщо p – вказівник типу T*, то вирази p+1, p+2, … задають адреси даних типу T, розташованих після *p "на відстані" 1, 2, … від *p. Так само вирази p-1, p-2, … позначають адреси даних типу T перед *p (рис. 1.2).

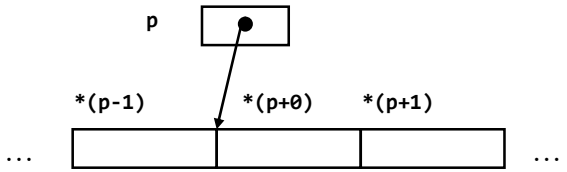

Рис. 1.2. Адресні вирази й дані

До вказівників як змінних застосовні складені присвоювання +=, -= з цілим виразом праворуч, а також ++ і --. Вони збільшують або зменшують значення вказівника на відповідну кількість sizeof(T), тобто переставляють його з кроком sizeof(T). Наприклад, якщо p має тип int*, то вираз p+=2 збільшує значення p (*адресу*) на 8, переставляючи його вперед на дві одиниці даних типу int.

До однотипних адрес застосовна операція віднімання. Якщо *AE1*, *AE2* – адресні вирази типу *T**, то значенням *арифметичного* виразу *AE2-AE1* є *ціла кількість одиниць даних типу T*, які "уміщаються між адресами" *AE1* і *AE2* (можливо, це число від'ємне). Зокрема, виразами *AE2* та *AE1* можуть бути імена вказівників.

Нарешті, адреси однотипних даних можна **порівнювати** за допомогою операторів ==, !=, <, <=, >, >=. Наприклад, якщо p – вказівник на деякий тип, то значенням виразів p<p+1, p!=p-1 є істина.

✓ Арифметичні операції з вказівниками вимагають підвищеної уваги. Помилки в цих операціях призводять зазвичай до спроб змінити зовсім не ті дані, які передбачає програміст, тому є *дуже небезпечними*.

Вправи

1.6. Припустимо, що p – вказівник, встановлений на цілу змінну зі значенням 3. Що можна сказати про значення виразів p, *p, &p, *&p й &*p?

1.7. Чому дорівнюють `sizeof(char)` і `sizeof(char*)`?

1.8. Припустимо, що `p1` і `p2` – вказівники типу `int*`. У чому різниця між присвоюваннями `p1=p2` та `*p1=*p2`? Чи є допустимими присвоювання `*p1=p2`, `p1=*p2`, `p1=&p2`?

1.9. Що буде виведено за програмою?

```
#include <iostream>
using namespace std;
void swapPtrs(int * & p1, int * & p2){
  int* t = p1; p1 = p2; p2 = t;
  return;
}
int main(){
  int a=1, b=2;
  int *pA=&a, *pB=&b;
  swapPtrs(pA, pB);
  cout << a << " " << b << endl;
  cout << *pA << " " << *pB << endl;
  system("pause");
  return 0;
}
```

1.3. Вказівники й масиви

Вказівники на елементи масиву

Якщо `p` – вказівник типу `T*`, то вираз вигляду `*(p+i)`, де `i` має ціле значення, ідентифікує дані типу `T` "на відстані `i`" від `*p`. Ці самі дані позначає й вираз `p[i]`, тобто вирази `*(p+i)` та `p[i]` *еквівалентні*; але вираз другого типу підтримується більшою кількістю мов програмування, ніж перший, і не використовує технічні особливості реалізації масивів у мові C++.

✓ У мові C++ ім'я масиву – це *адресний вираз, що позначає незмінюване значення*, яке є адресою першого елемента масиву. Ім'я масиву може бути операндом арифметичних операцій з адресами (звісно, окрім присвоювань).

Приклади

1. Нехай оголошено масив цілих `a` й вказівник на цілі `p`.

```
int a[10]; int * p;
```

За дії цих оголошень обидва вирази `p=&a[0]` і `p=a` встановлюють `p` на елемент `a[0]`. Після цього вирази `p[0]`, `p[1]`, …, `p[9]` ідентифі-

кують елементи масиву a так само, як і вирази a[0], a[1], …, a[9]. Можна встановити p на якийсь інший елемент масиву, присвоївши адресу елемента, наприклад, p=&a[5] або p=a+5.

2. Розглянемо таку програму:

```
#include <iostream>
using namespace std;
typedef int * PInt;
int main(){
  int a[] = {10, 11, 12, 13};
  int N = sizeof(a)/sizeof(int);
  PInt p = a, p2;
  for(int i=0; i<N; ++i) cout<<p[i]<<' '; /*1*/
  cout << endl;
  for(p2=a+N; p2>a; cout<<*(p2-=1)<<' '); /*2*/
  cout << endl;
  for(p2=a; p2<a+N; cout<<*(p2++)<<' ');  /*3*/
  cout << endl;
  cout << p2-a << ' ' << *a << endl;
  return 0;
}
```

Спочатку утворюється масив із чотирьох елементів. Його довжина обчислюється та зберігається в змінній N. Вказівник p встановлюється на його перший елемент.

У циклі в рядку /*1*/ вказівник p вказує на початок масиву, а вираз p[i] за значень i 0, 1, 2, 3 позначає послідовні елементи масиву. Виводяться їхні значення від 10 до 13.

У рядку /*2*/ вказівник p2 встановлюється на ділянку пам'яті *після останнього елемента* масиву. Далі в циклі адресне значення p2 спочатку зменшується на величину sizeof(int), тобто на 4, а потім виводиться значення елемента масиву, на який він вказує. Ці дії закінчуються, коли p2 досягає першого елемента масиву. Виводяться значення від 13 до 10.

У рядку /*3*/ вказівник p2 встановлюється на початок масиву. Далі дії аналогічні /*2*/, лише p2 *збільшується після виведення*, а все закінчується, коли значення p2 вийде за межі масиву. Виводяться значення від 10 до 13.

Наприкінці виводиться 4 – кількість змінних типу int, що вміщаються між вказівниками a й p2, тобто довжина масиву, і 10 – значення елемента з індексом 0.

Параметри функції, що зображують масив

Ім'я масиву є адресним виразом, незмінюваним значенням якого є адреса початку масиву. У виклику функції в мові C++ підстановка аргументу, який є масивом, на місце параметра – це *передавання адреси* початку масиву в пам'ять виклику функції. Звідси параметр функції, що зображує масив, *має бути вказівником і параметром-значенням*.

Оголошення параметра, що зображує масив у функції, зазвичай має вигляд `T ім'я[]` або `T *ім'я`, де `T` – вираз, що задає тип елементів масиву. Обидва вирази в заголовку функції задають одне й те саме – *вказівник на дані типу* `T`.

- ✓ Якщо параметр функції `a` оголошено у вигляді `T a[]` або `T *a`, то вираз `sizeof(a)` у тілі функції має значення 4 – кількість байт у вказівнику (у 64-розрядному середовищі буде не 4, а 8). У мові C++ елементи масиву, заданого аргументом у виклику функції, у пам'ять виклику *не копіюються*.

Проте адреса початку масиву ніяк не вказує на його довжину, необхідну для обробки масиву у функції. Отже, довжину масиву у функції зазвичай зображує окремий параметр.

Приклад. Розглянемо функцію, що виводить значення елементів масиву цілих. Параметрами є масив і його довжина.

```
void outAIntN(const int a[], int n)
{ for (int i=0; i<n; ++i)
    cout << a[i] << ' ';
  cout << endl;
  return;
}
```

Якщо у виклику цієї функції, наприклад `outAIntN(x, m);`, указано масив `x`, що має розмір `NMAX`, то значення `m` має бути не більше `NMAX`. Це повинен забезпечити програміст, що пише виклик, адже у функції справжня довжина масиву-аргументу невідома. Якщо значення `m` більше `NMAX`, то під час виконання виклику обробляються дані за межами масиву, а це може мати непередбачувані наслідки. Отже, функція працює з масивами будь-якої довжини, і це може бути *небезпечним*.

Специфікатори `const`, записані в заголовку наведеної функції, означають: *елементи й довжина масиву незмінювані* в тілі функції. Узагалі специфікатор `const` перед іменем типу позна-

чає незмінюваність даних цього типу. Отже, елементи масиву цілих у наведеній функції мають тип `const int`, тобто "незмінюване ціле". Водночас параметр `a` – це вказівник на дані цього типу, тобто змінна, й у функції за потреби його можна змінювати. ◂

Обробка початку масиву. У багатьох ситуаціях функція має обробляти не всі елементи масиву, а деяку їх частину, найчастіше – на початку масиву. Довжину цієї *робочої частини* масиву доцільно зобразити окремим параметром. Якщо ця довжина перед викликом функції невідома, то параметр має бути параметром-посиланням.

Приклад. Функція має заповнювати масив цілих значеннями, що їх задає користувач за допомогою клавіатури. Ознакою кінця введення елементів масиву є введення символів, що не зображують ціле значення. Перший параметр функції зображує масив, другий – його довжину. Значенням третього параметра (це параметр-посилання) під час виконання виклику стає кількість елементів, що отримують значення. Функція повертає булеве значення – ознаку того, що введено не більше значень ніж уміщує масив.

```
bool inAInt(int a[], int Nmax, int & n)
{ cout << "Enter int values (not more than " <<
          Nmax << ")\n";
  bool ok=true; n=0; int v;
  while(ok && cin >> v)
    if(n<Nmax)
      a[n++]=v;
    else ok=false;
  return ok;
}
```

Вираз `n<Nmax` у тілі циклу не дозволяє записати в масив більше ніж `Nmax` значень. Вираз уведення `cin >> v` записано в умові продовження. Він має ненульове значення, якщо введення відбулося, а інакше – значення 0. Уведення не відбувається, якщо символи, набрані користувачем, не зображують значення з типу `int`, зокрема, якщо користувач натиснув на клавіші `Ctrl-Z` і `Enter`. Це дозволяє користувачу заповнити не весь масив, а лише деякий його початок. ◂

Вправи

1.10. Написати функцію, яка за двома цілочисловими масивами однакового розміру визначає, чи мають їх відповідні елементи (з однаковими індексами) однакові значення. За її допомогою визначити, чи збігаються перша й друга половини масиву (за непарного розміру серединний елемент масиву не розглядається).

1.11. Написати функцію, яка виводить значення елементів свого параметра-масиву з парними індексами (0, 2, 4 тощо). Скористатися нею для виведення значень елементів масиву з парними індексами, а потім – елементів з непарними індексами.

1.12. Написати програму з трьома функціями: уведення значень елементів масиву цілих (можливо, значення отримують не всі елементи), виведення, порівняння двох масивів. У головній функції ввести два масиви довжиною не більше 10, вивести їх і результат їх порівняння (масиви рівні, мають різні довжини, відрізняються деякими елементами).

1.13. Прочитати натуральне число типу `int`, основу системи числення p, де $p < 37$, і вивести:

а) p-ковий запис числа;

б) значення p-кових цифр у вигляді багаточлена зі степенями числа p; піднесення до степеня позначити символом `^`, множення – символом `*`. За цифри 0 відповідний степінь числа p не виводиться, а за цифри 1 виводиться без неї, наприклад, за числа 1407 і основи 10 виводиться багаточлен `10^3+4*10^2+7*10^0`. *Вказівка*. Заповнення масиву, що зображує запис числа, і його виведення оформити у вигляді окремих функцій.

1.14. Написати функцію, що за масивом дійсних знаходить середнє арифметичне значень його елементів.

1.15. Написати функцію, яка за послідовністю дійсних чисел x_0, x_1, …, x_{n-1} у масиві обчислює таку величину:
$x_{n-1}(x_{n-1}+x_{n-2})(x_{n-1}+x_{n-2}+x_{n-3})\ldots(x_{n-1}+\ldots+x_0)$.

1.16. Написати функцію, що циклічно зсуває на одну позицію ліворуч значення елементів масиву цілих, не використовуючи додаткових масивів.

1.17. Написати функцію, що циклічно зсуває на одну позицію праворуч значення елементів масиву цілих, не використовуючи додаткових масивів.

1.18. Написати функцію, що без використання додаткових масивів обертає послідовність цілих у масиві. Наприклад, послідовність 0, 1, 2, 3, 4 має перетворитися на 4, 3, 2, 1, 0.

1.19. Написати функцію, яка без використання додаткових масивів перебудовує масив так, що спочатку підряд у тому ж самому порядку розташовані всі ненульові значення його елементів, а потім усі нульові.

1.20. Дано натуральне число n і дійсні числа $x_1, \ldots, x_{2n}$, зображені в масиві. Підрахувати суму тих чисел з $x_{n+1}, \ldots, x_{2n}$, які за величиною перевищують кожне з чисел $x_1, \ldots, x_n$.

1.21. Написати функцію, що без використання додаткових масивів перебудовує масив так, щоб спочатку були розташовані всі від'ємні елементи масиву, потім усі нульові, а потім усі додатні. Порядок елементів усередині групи можна змінювати.

1.4. Масиви, елементами яких є масиви

Поняття масиву масивів

Прямокутна таблиця з $m \times n$ однотипних елементів має *два виміри* – рядки й стовпчики. Її можна розглядати як послідовність m рядків, кожен з яких має n елементів, тобто як *масив, елементами якого є масиви*. Погляд на двовимірну таблицю як на масив масивів утілено в мові C++. Наприклад, таблицю цілих чисел 2×3 (2 рядки й 3 стовпчики) можна зобразити масивом `int a2[2][3]`. Цей масив має два елементи: кожен є масивом із трьох цілих змінних і зображує рядок таблиці (з індексом 0 або 1). Елементу таблиці відповідає ціла змінна, визначена двома індексами: перший указує рядок, другий – стовпчик. Послідовні цілі змінні в пам'яті зображують таблицю "рядками": `a2[0][0]`, `a2[0][1]`, `a2[0][2]`, `a2[1][0]`, `a2[1][1]`, `a2[1][2]`.

Послідовність двовимірних таблиць утворює тривимірну таблицю. Її можна зобразити масивом, елементи якого зображують двовимірні таблиці. Наприклад, масив `int a3[5][2][3]` має п'ять елементів, кожен як масив `a2`.

Масиви, елементами яких є масиви, для зручності будемо називати **багатовимірними**.[2] Наприклад, масив `a2`, наведений вище, назвемо двовимірним, масив `a3` – тривимірним. Перший вимір масиву називається **зовнішнім**, останній – **внутрішнім**. Наприклад, у масиві `int a3[5][2][3]` зовнішній вимір містить п'ять елементів (це двовимірні масиви), внутрішній – три (цілі змінні). Елементи внутрішнього виміру (як цілі змінні в масиві `int a3[5][2][3]`) будемо також називати елементами багатовимірного масиву.

Елементи тривимірного масиву `int a3[5][2][3]`, тобто цілі змінні, розташовуються в пам'яті аналогічно змінним масиву `int a2[2][3]`:

```
a3[0][0][0], a3[0][0][1], ..., a3[0][1][2],
...
a3[5][0][0], a3[5][0][1], ..., a3[5][1][2].
```

Отже, у послідовних елементів багатовимірного масиву найшвидше змінюється внутрішній індекс, найповільніше – зовнішній.

Ім'я багатовимірного масиву є незмінюваним адресним виразом, значення якого – адреса першого елемента масиву (що сам є масивом).

Приклади

1. Нехай означено масив `int a2[2][3]`. Значення виразів `a2` й `a2+1` – це адреси початків першого й другого масивів (рядків таблиці), кожен з яких складається із трьох цілих змінних. Вирази `a2[i]` та `*(a2+i)`, де `i` = 0 або 1, позначають `i`-й масив так само, як ім'я `a2` – увесь двовимірний масив. Аналогічно значеннями цих виразів є *адреси перших елементів масивів*. Звідси вирази `*(a2+i)+j` та `a2[i]+j`, де `j` = 0, 1, 2, є адресними й значенням кожного з них є адреса цілої змінної, що розташована в `i`-му "рядку" на `j`-му місці, тобто адреса цілого елемента `a2[i][j]`. Цей елемент позначають також вирази `*(*(a2+i)+j)`, `*(a2[i]+j)` та `(*(a2+i))[j]`.

2. Нехай означено масив `int a3[5][2][3]`. Значенням виразу вигляду `a3+i` (значення `i` від 0 до 4) є адреса початку `i`-го масиву з 2×3 цілих змінних. Вирази `*(a3+i)` та `a3[i]` позначають `i`-й масив 2×3; значенням цих виразів є його адреса. Аналогічно

[2] У мові C++ цього терміна немає, на відміну від деяких інших мов програмування.

значенням виразу `*(a3+i)+j`, де `j`=0 або 1, є адреса масиву з трьох цілих змінних, а виразу `*(*(a3+i)+j)+k` – адреса `k`-го елемента цього лінійного масиву, тобто адреса цілої змінної `a3[i][j][k]`. Її позначають також вирази, наприклад `*(a3[i][j]+k)` або `(*(a3+i)+j)[k]`. ◂

Багатовимірний масив ініціалізується аналогічно одновимірному, лише ініціалізуючим виразом може бути як послідовність значень, так і послідовність послідовностей. Зокрема, для двовимірного масиву може бути один або два рівні дужок `{}`, для тривимірного – від одного до трьох рівнів тощо. Наприклад, такі дві ініціалізації створюють масиви з однаковими значеннями (11, 12, 0 у першому рядку, 21, 0, 0 – у другому).

```
int a2[2][3] = {11,12,0,21};
int b2[2][3] = {{11,12},{21}};
```

Параметри для багатовимірних масивів

Параметри, що у функціях зображують багатовимірні масиви, розглянемо на прикладах із двовимірними масивами.

Невизначений зовнішній розмір. Параметр функції, що зображує багатовимірний масив, є вказівником на елементи масиву, які самі є масивами.

Елементи двовимірного масиву позначаються виразами вигляду `a[i][j]`, `(*(a+i))[j]` або `*(*(a+i)+j)`. Щоб компілятор правильно обробив ці вирази, йому потрібен розмір рядків масиву (розмір внутрішнього виміру). Розмір є добутком довжини рядка-масиву й розміру елементів масиву.

✓ Оголошуючи параметр, що зображує багатовимірний масив, *необхідно вказати довжину по всіх вимірах*, окрім зовнішнього.

Параметр, що зображує матрицю цілих, можна оголосити, наприклад, як `int a[][10]` або `int (*a)[10]`. За обома оголошеннями `a` є вказівником на масиви з 10 цілих, тому вираз вигляду `a[i][j]` або `(*(a+i))[j]` ідентифікує елемент у *i*-му рядку та *j*-му стовпчику. Аналогічно параметр, що зображує тривимірний масив цілих, можна оголосити як, наприклад, `int a[][20][30]` або `int (*a)[20][30]`.

В оголошенні параметра, наприклад `int(*a)[5]`, *дужки обов'язкові*, оскільки пріоритет постфіксної операції `[]` вище, ніж префіксної `*`. Вираз без дужок `int*a[5]` і еквівалентний йому `int*(a[5])` оголошують масив із 5 вказівників на цілі.

Для роботи з параметром функції, що зображує багатовимірний масив, може знадобитися параметр для довжини за зовнішнім виміром. Проте частіше в багатовимірному масиві обробляється робоча частина – перші рядки й перші стовпчики. Тоді у функції потрібні параметри, що зображують робочі довжини по всіх вимірах масиву.

Приклади.

1. Розглянемо програму з двовимірним масивом цілих `m`, що зображує матрицю розміром у межах 20×30. Довжини вимірів масиву (кількість рядків і стовпчиків) задано глобальними константами `RMAX` і `CMAX`. Матриця може мати й менше ніж 20 рядків або 30 стовпчиків, тому справжні кількості рядків і стовпчиків матриці зберігаються в окремих змінних `r` і `c`.

Функція `fillRands` отримує від користувача кількість рядків і стовпчиків матриці. Ці величини визначаються, коли виконується виклик функції, тому їх зображено параметрами-посиланнями. Якщо вони більше ніж, відповідно, `RMAX` і `CMAX`, то функція змінює їх; далі вона заповнює матрицю псевдовипадковими цілими числами.

Функція `outMInt` виводить значення елементів матриці. Вона отримує вже відомі кількості рядків і стовпчиків, тому має параметри-значення.

Головна функція оголошує змінні й викликає функції заповнення матриці та її виведення.

```
#include <iostream>
using namespace std;
const unsigned RMAX = 20, CMAX = 30;
void fillRands(int m[][CMAX],
       unsigned & r, unsigned & c)
{ cout << "Enter numbers of rows and columns "
       << "(positive less or equal "
       << RMAX << " and " << CMAX << "): " ;
  cin >> r >> c;

 if(r>RMAX) {
    r=RMAX;
    cout << "Too many rows. Set to " << RMAX;
  }
```

```
  if(c>CMAX) {
    c=CMAX;
    cout << "Too many columns. Set to " << CMAX;
  }
  int i, j;
  for(i=0; i<r; ++i)
    for(j=0; j<c; ++j)
       m[i][j]=rand();
}
void outMInt(int m[][CMAX],
       unsigned r, unsigned c)
{ for(int i=0; i<r; ++i)
    { for(int j=0; j<c; ++j)
        cout << m[i][j] << ' ';
      cout << endl;
    }
}
int main()
{ int m[RMAX][CMAX];
  unsigned r, c;
  fillRands(m, r, c);
  outMInt(m, r, c);
  return 0;
}
```

2. Функція `transp` транспонує квадратну матрицю, тобто відображає її відносно головної діагоналі. Кількість рядків і стовпчиків у квадратній матриці однакова й залишається без змін, тому її зображує параметр-значення `r`. Матриця займає частину масиву з довжиною рядків `NMAX`, тобто вважається, що значення `r` не більше `NMAX`.

```
void transp(int m[][NMAX], int r)
{ int i, j; int t;
  for(i=0; i<r; i++)
    for(j=i+1; j<r; j++)
      { t=m[i][j]; m[i][j]=m[j][i]; m[j][i]=t; }
}
```

Матриця з фіксованими розмірами. У багатьох задачах фігурують матриці, обидва розміри яких лежать у певних межах.

Можна оголосити ім'я типу двовимірних масивів для зображення матриць і користуватися ним у заголовках функцій.

Нехай матриці цілих чисел мають розміри в межах 20×30. Для їх зображення оголосимо ім'я типу масивів. В інструкції `typedef` ім'я типу масивів записується перед описом вимірів.

```
const unsigned RMAX = 20, CMAX = 30;
typedef int MInt[RMAX][CMAX];
```

Після цих оголошень можна означати та обробляти змінні типу `MInt` – двовимірні масиви. У зоні дії цих оголошень запишемо функцію виведення значень елементів матриці.

```
void outMInt(const MInt m, unsigned r, unsigned c)
{ // вважаємо, що r<RMAX, c<CMAX
  int i, k;
  for (i=0; i<r; ++i)
    { for(k=0; k<c; ++k)
        cout << m[i][k] << ' ';
      cout << endl;
    }
}
```

Вправи

1.22. Написати функцію побудови за дійсними числами a_0, a_1, …, a_{n-1} ($n \le 50$) такої квадратної матриці:

$$\begin{pmatrix} a_0 & a_1 & \dots & a_{n-2} & a_{n-1} \\ a_1 & a_2 & \dots & a_{n-1} & a_0 \\ \dots & & \dots & & \dots \\ a_{n-2} & a_{n-1} & \dots & a_{n-4} & a_{n-3} \\ a_{n-1} & a_0 & \dots & a_{n-3} & a_{n-2} \end{pmatrix}$$

1.23. За лінійним масивом a_0, …, a_{n-1} побудувати таку матрицю Вандермонда:

$$\begin{pmatrix} 1 & 1 & \dots & 1 \\ a_0 & a_1 & \dots & a_{n-1} \\ \dots & \dots & \dots & \dots \\ a_0^{n-1} & a_1^{n-2} & \dots & a_{n-1}^{n-1} \end{pmatrix}.$$

1.24. Написати функцію побудови матриці розмірністю $n \times n$ з елементами $a_{i,j} = \begin{cases} \sum_{k=i}^{j} \frac{x^k}{k!}, i \le j, \\ a_{j,i}, i > j. \end{cases}$

1.25. У матриці поміняти місцями: а) два рядки; б) два стовпчики; в) одночасно два рядки й два стовпчики, задані номерами.

1.26. Написати функцію множення двох матриць.

1.27. Не використовуючи додаткових масивів, повернути квадратну матрицю за годинниковою стрілкою на: а) 180°; б) 90°.

1.28. Не використовуючи додаткових масивів, транспонувати матрицю.

1.29. За квадратною числовою матрицею одержати послідовність чисел обходом матриці: а) "змійкою", починаючи по горизонталі з лівого верхнього кута; б) "спіраллю" за годинниковою стрілкою з лівого верхнього кута.

1.30. За квадратною числовою матрицею одержати суму чисел, записаних: а) у центральному ромбі; б) у правому чвертьтрикутнику.

1.5. Вказівники: додаткові можливості

Перетворення типів адрес

✓ Присвоювати адреси даних одного типу вказівникам на дані іншого типу *заборонено*. Проте адресу даних одного типу можна *перетворити* до типу адрес іншого типу.

Приклади

1. Якщо `a` – ім'я масиву цілих чисел, то на його перший байт можна встановити вказівник на символи:

```
char*p = (char*)&a[0]; або
char*p = (char*)a;.
```

2. Розглянемо функцію, в якій масив `ch` із чотирьох символів (по одному байту) обробляється як одне ціле число типу `int` за допомогою вказівника `p` на цей тип. Спочатку значення елементів масиву виводяться. Потім вказівник `p` встановлюється на елемент `ch[0]`, адресу якого перетворено до типу "адреса даних типу

int". Далі обробляється значення змінної *p – виводяться значення шістнадцяткових цифр числа, а цифри "викреслюються".

```
int main(){
  const int N=4;
  char ch[N] = {'0', 'A', '1', 'a'};
  for (int i=0; i<N; ++i)
    cout << '<' << ch[i] << "> ";
  cout << endl;
  int*p = (int*)ch;
  while (*p)
  { cout << (*p)%16 << ' ';
    *p/=16;
  }
  return 0;
}
```

Виходом функції будуть два таких рядки:

```
<0> <A> <1> <a>
0 3 1 4 1 3 1 6
```

Перший містить символи в масиві, другий – шістнадцяткові цифри цілого числа в тому самому масиві, від молодшої до старшої. Вони свідчать: символ '0' має шістнадцятковий код 30, 'A' – 41, '1' – 31, 'a' – 61. Звідси також ясно, що біти цілого значення насправді розташовані в пам'яті від молодшого до старшого, а не так, як ми їх зображуємо (старший – ліворуч, молодший – праворуч).

- ✓ *Обробка даних одного типу як даних іншого типу* (зокрема, за допомогою вказівників) вимагає підвищеної уважності й знання подробиць і особливостей зображення даних.

Вказівники на функції

Функція під час виконання програми займає певну ділянку пам'яті, тобто має адресу.

- ✓ Ім'я функції – це *адресний вираз*, значенням якого є адреса функції.

На функцію можна встановити вказівник, тип якого відповідає прототипу функції. Наприклад, якщо функція f має прототип int f(int), то вказівник pf на функції з цим прототипом оголошується так:

```
int (*pf)(int);
```

Дужки навколо *pf обов'язкові. Пріоритет префіксного оператора * нижче, ніж пріоритет дужок (), тому оголошення без дужок

`int*pf(int);` є прототипом функції, яка має ім'я `pf` і повертає вказівник на `int`.

Установити вказівник на функцію можна за допомогою її імені, наприклад `pf=f`. Після цього ім'я `pf` позначає функцію так само, як і `f`, тому, наприклад, вираз `pf(10)` є викликом функції `f` з аргументом `10`.

✓ Параметр функції може зображувати функції. Для цього параметр оголошують як *вказівник на функції* з відповідним заголовком.

Приклад. Табуляція функції полягає в тім, що виводяться її значення в певних точках. Припустимо, що точки – це цілі числа з деякого діапазону, а функція в цих точках має цілі значення. Потрібно пройти цілі точки діапазону й вивести значення функції в них.

Опишемо табуляцію у функції `fTab`. Дії з табуляції не залежать від того, яка саме функція табулюється, тому цю функцію зображує параметр функції `fTab`. За умовою, табульовані функції мають заголовок вигляду `int f(int)`, інші два параметри визначають діапазон точок. Отже, функція `fTab` може мати такий прототип:

```
void fTab(int (*f)(int), int, int);
```

У тілі функції `fTab` табульована функція викликається за допомогою параметра `f`.

Розглянемо програму з двома викликами функції `fTab`, у яких аргументи вказують на функції $f_1(x)=x^2+1$ і $f_2(x)=2x$. Функції табулюються в цілих точках від 0 до 8. Аргументом, що у виклику відповідає параметру-вказівнику на функції, може бути як ім'я функції, так і ім'я вказівника, установленого на неї.

```
void fTab(int (*f)(int), int low, int up);
int f1(int), f2(int);
int main()
{ int (* pf)(int) = f1;  // pf - синонім f1
  fTab(pf,0,8);          // табуляція f1
  fTab(f2,0,8);          // табуляція f2
  return 0;
}
// функція табуляції
void fTab(int (*f)(int), int low, int up)
{ for(int i=low; i<=up; i++)
    cout << i << ':' << f(i) << "  ";
  cout << endl;
}
```

```
// табульовані функції
int f1(int x) { return x*x+1; }
int f2(int x) { return 2*x; }
```

У наведеному прикладі можна оголосити *ім'я типу вказівників на функції*. Оголошення імені `PFII` типу вказівників на функції з прототипом вигляду `int f(int)` виглядало б так:

```
typedef int (*PFII)(int);
```

Відповідно прототип функції табулювання був би таким:

```
void fTab(PFII f, int low, int up);
```

Решта програми не змінилася б.

Вправи

1.31. Що виводиться за програмою?

```
#include <iostream>
using namespace std;
int main(){
  int i, a = 66*256*256*256+50*256*256+65*256+49;
  char * p = (char *)&a;
  for (i=0; i<4; cout << p[i++]);
  cout << endl;
  return 0;
}
```

1.32. Якщо функція $f(x)$ визначена на відрізку $[a; b]$, неперервна на проміжку $(a; b)$ і набуває на кінцях відрізка значення різних знаків, то рівняння вигляду $f(x) = 0$ має хоча б один корінь на $[a; b]$. Метод половинного поділу дозволяє наближено обчислити значення x, для якого $|f(x)| < \varepsilon$ (у деяких випадках воно буде наближенням кореня рівняння, але не в усіх). Суть методу полягає в тім, що обчислюється середина m відрізка $[a; b]$ і за умови $f(m) \neq 0$ пошук продовжується у такий самий спосіб на тому з відрізків $[a; m]$ і $[m; b]$, на кінцях якого функція набуває значення різних знаків. Пошук припиняється, якщо на якомусь кроці $f(m) = 0$ або довжина відрізка стає меншою за задану межу. Написати функцію, яка має параметр, що зображує функцію дійсного аргументу з дійсними значеннями, і реалізує метод половинного поділу. Написати програму, що розв'язує задачу для $\sin x - c = 0$, де $-1 < c < 1$, на відрізку $[-\pi/2; \pi/2]$.

1.33. Для того, щоб наближено обчислити інтеграл функції, визначеної на відрізку $[a; b]$, можна застосувати метод

трапецій. Відрізок $[a; b]$ розбивають на відрізки вигляду $[a+i\times h; a+(i+1)\times h]$ і обчислюють суму площ трапецій, утворених точками $a+i\times h$, $f(a+i\times h)$, $f(a+(i+1)\times h)$, $a+(i+1)\times h$. При цьому крок h вибирають так, що $a+m\times h = b$ за певного $m > 0$. Інтеграл можна обчислити шляхом подвоєння точності. Спочатку вибирають $m = 1$, тобто $h = b-a$, і обчислюють інтеграл. Далі на кожному наступному етапі крок h зменшують удвічі й обчислюють інтеграл. Етапи повторюють, поки значення інтеграла, одержані на двох суміжних етапах, відрізняються більше ніж на деяке задане ε. Написати функцію, яка має параметр, що зображує функцію дійсного аргументу з дійсними значеннями, і обчислює інтеграл методом трапецій з подвоєнням точності. Написати програму наближеного обчислення інтеграла функції $\sin x$ на відрізку $[0; \pi]$ (його точним значенням є 2).

Контрольні запитання

1.1. Що таке масив? Опишіть вигляд означення масиву.
1.2. Які вирази забезпечують доступ до окремого елемента масиву?
1.3. У чому полягає прямий доступ до елементів?
1.4. Чим з погляду математики є значення масиву?
1.5. Що таке адреса змінної?
1.6. За допомогою якої операції можна задати адресу змінної?
1.7. Що таке типізований вказівник?
1.8. Який вираз позначає тип адрес даних певного типу?
1.9. Чи залежать розміри типізованих вказівників від їх типів?
1.10. Що таке встановлення вказівника на змінну?
1.11. Що таке розіменування вказівника? Як воно позначається?
1.12. Як виглядає оголошення імені типу типізованих вказівників?
1.13. Які арифметичні дії можливі з адресами?
1.14. Яке значення має вираз, що є ім'ям масиву?
1.15. Як елементи масиву масивів зберігаються в пам'яті?
1.16. Чи допустима в мові C++ інструкція присвоювання, яка ліворуч містить лише ім'я масиву? Чому?
1.17. Чи завжди можна промоделювати масивом послідовність, довжина якої заздалегідь невідома?