**B.I.Березін,С.Б.Березін(С.83) МАСИВИ І ПОКАЖЧИКИ**

Раніше ми ввели типи даних в мові С, які називаються іноді ба­зовими або вбудованими. На основі цих типів даних мова С дозволяє будувати інші типи даних і структури даних. Масив - один з най­більш простих і відомих структур даних. **Під масивом** в мові С **розу­міють набір даних одного і того ж типу, зібраних під одним ім'ям. Кожний елемент масиву визначається ім'ям масиву і порядковим номе­ром** елемента, який називається **індексом. Індекс** в мові С **завжди ціле число.**

**ОГОЛОШЕННЯ МАСИВУ В ПРОГРАМІ**

Основна форма оголошення масиву розмірності N така:

**тип <ім'я масиву>[размер1][размер2]...[размерН]**

Частіше за все використовуються **одновимірні масиви:**

**тип <ім'я масиву> [розмір] ;**

**тип** - базовий **тип елементів** масиву, **розмір - кількість елементів** одновимірного масиву.

При описі **двовимірного масиву** оголошення має наступний вигляд:

**тип <ім'я масиву> [размері][размер2];**

У цьому описі можна трактувати оголошення двовимірного масиву як оголошення масиву масивів, т. е. масив розміру [размер2], еле­ментами якого є одновимірні масиви <ім'я масиву>[размер1].

**Розмір масиву** в мові С може **задаватися константою** або **констан­тним виразом. Не можна задати масив змінного розміру.** Для цього існує окремий механізм, званий динамічним виділенням пам'яті.

**ОДНОВИМІРНІ МАСИВИ**

У мові С **індекс** завжди **починається з нуля.** Коли ми говоримо про перший елемент масиву, то маємо на увазі елемент з індексом 0. Еслі ми оголосили масив

int a[100] ;

це означає, що масив містить 100 елементів від а[0] до а[99]. Для одновимірного масиву легко підрахувати, скільки байт в пам'яті бу­де займати цей масив:

кільк.байтів=<розмір базового типу>\*<кільк.елементів>.

У мові С під масив завжди виділяється безперервне місце в опе­ративній пам'яті.

У мові **С не перевіряється вихід індексу за межі масиву.** Якщо **масив а[100]** описаний як цілочисельний масив, що має 100 елемен­тів, а ви в програмі **вкажете а[200], то повідомлення про помилку не буде видане,** а як значення елемента а[200] **буде видано деяке число,** що займає відповідні 2 байти. Можна визначити масив будь-якого визначеного раніше типу, наприклад

unsigned arr[40], long double al[1000], char ch[80].

|  |  |
| --- | --- |
| /\*поміняти місцями max з min\*/  #include <iostream.h> main()  { int i,j,a[10], max. nmax, min, nmin, temp; clrscr(); for (i=0; i<10; i++) сіп » a[i]; max=min=a[0]; nmax=nmin=0; for (i=0; i<10; i++) if(a[i]>max) { max=a[i]; nmax=i;} else if(a[i]<min) {min=a[i], nmin=i;} tern p= a[n m ax]; a[n max]=a[nm і n]; a[nm і n]=temp; for (i=0; i<10; i++) cout « a[i] «" "; } | // Сортування і програвання масиву  #include<iostream.h> #include<dos. h> *#'\* nclude<conio. h> void main() { int temp, і, j, a[ 1 0]; clrscr(); for (i=0;i<10;i++) сіп » a [ і ]; for (i=0;i<9;i++) for (j=i+1 ;j<10;j++) if (a[i]>a[j]) { temp=a[i]; a[i]=a[j]; a[j]=temp; } for (i=0;i<1 0;i++) { cout « a[i]«" "; sound(a[i]\*80); delay(500); nosou nd(); } getch(); } |

**МАСИВИ СИМВОЛІВ. РЯДКИ**

Однак масиви типу char - символьні масиви - займають в мові осо­бливе місце. У багатьох мовах е спеціальний тип даних - рядок сим­волів (string). У мові С окремого типу рядка символів немає, а ре­алізована робота з рядками шляхом використання одновимірних маси­вів типу char. У мові С символьний рядок - це одновимірний масив типу char, що закінчується нульовим байтом. Нульовий байт - це байт, кожний біт якого рівний нулю. Для нульового байта визначена спеціальна символьна константа ' \0 ' . Це потрібно враховувати при описі відповідного масиву символів. Так, якщо **рядок повинен** місти­ти N символів, то в описі **масиву** потрібно указати **N+1 елемент.**

Наприклад, опис

char str[11] ;

передбачає, що рядок містить 10 символів, а останній байт зарезер­вований під нульовий байт. Звичайно, ми задали звичайний одновимі­рний масив, але якщо ми хочемо трактувати його як рядок символів, то це буде рядок максимум з 10 елементів.

Хоча в мові С немає спеціального типу рядка, мова допускає ряд­кові константи. Рядкова константа - це список літер, взятих в по­двійні лапки. Наприклад,

"Borland C++ ", "Це рядкова константа".

У кінець рядкової константи не треба ставити символ '\0'. Це зробить компілятор, і рядок "Borland C++" в пам'яті буде вигляда-

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| В | о | г | 1 | а n | d |  | С | + | + | \0 |

**Є два** простих **способи ввести рядок** з клавіатури. Перший спосіб -скористатися функцією scanf() зі специфікатором введення %s. Треба пам'ятати, що функція **scanf() вводить** символи **до першого пропуско-вого символа. Другий** спосіб - скористатися спеціальною бібліотеч­ною функцією gets()**, оголошеною** в файлі stdio.h. **Функція** gets() дозволяє **вводити рядки, що містять пропуски.** Введення закінчується натисненням клавіші Enter. Обидві функції автоматично ставлять в кінець рядка нульовий байт. Не забудьте зарезервувати для нього місце. Як параметр в цих функціях використовується просто ім'я ма­сиву.

|  |  |
| --- | --- |
| #i ncl ude <stdio. h> void main () { char s1[80], s2[80]; scanf( %s, "s1); *І"* можна об'єднати 2 scanf в один s c a n f ( % s % s , " s 1 , s 2); \* / scanf("%S", s2); printf("%s\n", s1); printf("%s", s2); } ввели: Hello! Good I uck! Резул ьтат: Hello! Good | #i nclude <std io. h> void main () { char s1[80], s2[80]; gets(s1); gets(s2) puts(s1); puts(s2);  } ввели: Hello! Good luck! Результат: Hello! Good luck! |

Виведення виробляється функціями printf() або puts(). Обидві фу­нкції виводять вміст масиву до першого нульового байта. Функція **puts() додає в** кінці рядка, що виводиться символ **нового рядка.** У функції printf() **перехід** на новий рядок треба **передбачати в** рядку формату **самим.**

**ФУНКЦІЇ ДЛЯ РОБОТИ З РЯДКАМИ**

Для роботи з рядками існує спеціальна бібліотека, опис якої знаходиться в файлі **string.h.** Найчастіше використовуються функції

**strcpyO, strcat(), strlenQ, strcmpO.**

Виклик функції strcpy() має вигляд

**strcpy(si, s2) ;**

Функція strcpy() використовується **для копіювання вмісту рядка s2 в**

**рядок s1.** Масив s1 повинен бути досить великим, щоб в нього вміс­тився рядок s2. Якщо місця мало, компілятор не видає вказівки на помилку або попередження; це не перерве виконання програми, але може привести до псування інших даних або самої програми і непра­вильній роботі програми надалі. Виклик функції strcat() має вигляд

**strcat(sl, s2) ;**

Функція strcat() **приєднує рядок s2 до рядка s1 і вміщує його в** ма­сив, де знаходився рядок **s1,** при цьому рядок s2 не змінюється. Ну­льовий байт, який завершував рядок s1, буде замінений першим сим­волом рядка s2. їв функції strcpyO, і в функції strcat() рядок, що виходить, автоматично завершується нульовим байтом.

Розглянемо простий приклад використання цих функцій.

**Резул ьтат:**

**Hello, World!**

**Hello, World! World!**

**#include <stdio.h>**

**#і ncl ude <string . h>**

**main () {**

**char s1[20], s2[20];**

**strcpy(s1 , "Hello, ");**

**strcpy(s2, "World!");**

**puts(s1);**

**puts(s2);**

**strcat(s1, s2);**

**puts(s1);**

**puts(s2);**

**}**

Виклик функції strcmpO має вигляд

**strcmp(sl, s2);**

Функція strcmpO порівнює рядки si і s2 і повертає значення О, якщо рядки однакові, тобто містять одне і те ж число однакових си­мволів. Під порівнянням рядків ми розуміємо порівняння в лексико­графічному значенні, так як це відбувається, наприклад, в словни­ку. Звичайно, в функції відбувається посимвольне порівняння кодів символів. Код першого символа одного рядка порівнюється з кодом символа другого рядка. Якщо вони однакові, розглядаються другі си­мволи тощо. Якщо зі лексикографічно (в значенні словника) більше s2, то функція strcmpO повертає додатне значення, якщо менше -від'**ємне** значення.

Виклик функції strlen() має вигляд

**strlen(s) ;**

Функція strlen() повертає **довжину рядка з,** при цьому завершаль­ний **нульовий байт не враховується.** Виклик length("Hello") поверне

значення 5.

Розглянемо застосування цієї функції для обчислення довжини ря­дка, що вводиться з клавіатури.

#include <stdio.h>

#incl ude <string . h > m а і n () { char s(80], printf( "Введіть рядок:");

gets(s);

printf( "Рядок\п%з\п має довжину %d символів \n", s, strlen(s)); }

**ДВОВИМІРНІ МАСИВИ**

Як ми вже зазначали, мова С допускає **багатовимірні масиви,** най­простішою формою яких е двовимірний **масив (two-dimentional** array). Можна сказати, що двовимірний масив - це масив одновимірних маси­вів .

Двовимірний масив int a[3][4] можна подати у вигляді таблички:

Другий індекс

Перший індекс

|  |  |  |  |
| --- | --- | --- | --- |
| а[0] [0] | а[0][1] | а[0][2] | а[0] [3] |
| а[1] [0] | а[1][1] | а[1][2] | а[1][3] |
| а[2][0] | а[2] [1] | а[2][2] | а[2] [3] |

**Перший індекс - номер рядка, другий індекс - номер стовпця.** Кіль­кість байт пам'яті, яке необхідне для зберігання масиву, обчислю­ється по формулі

Кільк.байтів = <розмір типу даних>\*<кільк.рядків>\*<кільк.ствпців>.

У пам'яті комп'ютера масив розташовується безперервно по ряд­ках, тобто а[0][0], а[0][1], а[0][2], а[0][3], а[1][0], а[1][1], а[1] [2], а[2] [1],. ... а[2] [3] .

Потрібно пам'ятати, що пам'ять для всіх масивів, які визначені як глобальні, відводиться в процесі компіляції і зберігається весь час, поки працює програма.

Часто двовимірні масиви використовуються для роботи з таблицями, що містять текстову інформацію. Також дуже часто використовуються масиви рядків.

**ІНІЦІАЛІЗАЦІЯ МАСИВІВ**

Дуже важливо уміти ініціалізувати масиви, тобто привласнювати елементам масиву деякі початкові значення. У мові С для цього є спеціальні можливості. **Самий простий спосіб ініціалізації** наступ­ний: **в процесі оголошення масиву** можна указати в **фігурних дужках список ініціалізаторів:**

float а[6]={1.1, 2.2, 3.3, 4.0, 5, 6};

В іншому випадку така форма запису еквівалентна набору операторів:

а[0]=1.1; а[1]=2.2; ... а [5] =6.

**Багатовимірні масиви,** в тому числі і двовимірні масиви, можна ініціалізувати, розглядаючи іх як масив масивів.

**Ініціалізації** int а[3][5]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

і int а[3][5]={{1,2,3,4,5}, {6,7,8,9,10}, {11,12,13,14,15}};

**еквівалентні.**

Кількість ініціалізаторів не зобов'язана співпадати з кількістю

елементів масиву. **Якщо** ініціалізаторів **менше, то значення решти**

елементів масиву **не визначені.**

У той же час **ініціалізації**

int а[3][5]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11);

і

int а[3][5]={{1, 2, 3}, {4, 5, 6, 7, 8}, {9, 10, 11}};

**різні.**

**//change strings: 1-6, 2-5, 3-4**

#i nclude<std io. h >

void mai n()

{ int temp, i, j, a[6][4]={1,2,3,4,

5,6,7,8,

9,10,11,12,

1 3,14,1 5,16,

17,18,19,20,

21 ,22,23,24};

for (i=0;i<3;i++) for (j=0;j<4;j++)

{ temp=a[i][j]; a[i][j]=a[5-i][j]; a[5-i][j]=temp; } for (i=0; i<6; i++)

{

for (j=0;j<4;j++)

printf ("%4d", a[i][j]);

printf("\n");

}}

Символьні масиви можуть ініціалізувати як звичайний масив:

char str[15]={'В', ' о ' , ' г ' , ' 1 ' , ' а ' , ' n ' , ' d' **, ' ','** С',^',^'};

а можуть - як рядок символів:

"char str[15]= Borland C++";

Відмінність цих двох способів полягає в тому, що **у другому випа­дку буде доданий ще і нульовий байт.** До того ж другий спосіб коро­тше. Допускається також оголошення і ініціалізація масиву без яв­ної вказівки розміру масиву. Наприклад, для виділення місця під символьний масив звичайним способом

char str[80]= "Це оголошення і ініціалізація масиву символів";

ми повинні вважати кількість символів в рядку або указати явно більший розмір масиву.

При ініціалізації масиву без вказівки його розміру

char str[ ]= "Це оголошення і ініціалізація масиву символів";

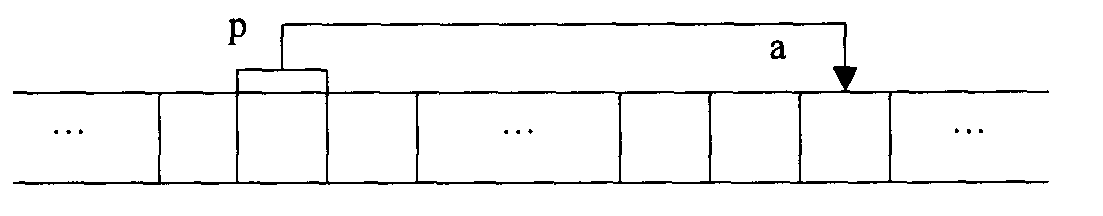
компілятор сам визначить необхідну кількість елементів масиву, включаючи нульовий байт. Можна оголошувати таким же способом маси­ви будь-якого типу:

int mass []={!, 2, 3, 1, 2, 3, 4};

**Від LG:** При ініціалізації можна не вказувати розмірність масиву, вона обчислюється авто­матично (проте для двовимірних масивів кількість стовпців треба указати), а при оголошенні - обов'язково. При оголошенні масивів з невідомою кількістю елемен­тів можна не вказувати розмір тільки в самих лівих квадратних дужках.

**ПОКАЖЧИКИ І АДРЕСИ (Керніган, Рітчі і Б.І.Березін,С.)(Б.Березін)**

Пам'ять машини являє собою масив послідовно розташованих і пронумерованих комірок, з якими можна працювати окремо і зв'язани­ми ділянками. Покажчик - це група комірок в пам'яті комп'ютера, в **яких** може зберігатися адреса.



Унарний оператор & видає адресу об'єкта, так що ін­струкція

р=&а;

привласнює адресу комірки а змінній р (тепер р вказує на а або по­силається) .

Оператор & застосовується тільки до об'єктів, розташованих в пам'яті: до **змінних** і елементам масивів. Його операндом **не** може бути ні вираз, ні константа, ні регістрова змінна.

Унарний оператор \* є оператор розкриття посилання. Застосований до покажчика, він видає об'єкт, на який даний покаж­чик посилається.

**ОГОЛОШЕННЯ ПОКАЖЧИКІВ**

Якщо змінна буде покажчиком, то вона повинна бути оголошена таким чином:

**тип \*<ім'я змінної>;**

У цьому оголошенні тип - деякий тип мови С, визначальний тип об'єкта, на який вказує покажчик (адреса якого містить); \* - озна­чає, що наступна за нею змінна є покажчиком.

**ОПЕРАЦІЇ НАД ПОКАЖЧИКАМИ**

З покажчиками пов'язані **дві** спеціальні **операції.: & і \*.** Обидві ці операції є **унарними, т. е. мають один операнд,** перед якими вони ставляться. Операція & відповідає операції "взяти адресу". Опера­ція \* відповідає словам "значення, розташоване за вказаною адре­сою" .

Особливість мови С полягає в тому, що знак \* відповідає двом операціям, що не мають один до одного ніякого відношення: арифме­тичній операції множення і операції взяти значення. У той же час сплутати їх в контексті програми не можливо, оскільки одна з опе­рацій унарна (містить один операнд), інша - множення - бінарна (містить два операнди). Унарні операції & і \* **мають найвищий пріо­ритет** нарівні з унарним мінусом.

В оголошенні змінної, що є покажчиком, дуже важливий базовий тип. Якщо покажчик має базовий тип int, то змінна займає 2 байти, char - 1 байт тощо. Приклад.

int а=3, Ь=5;

int \*р;

р = &а; /\* тепер р вказує на а\*/ Ь = \*р; /\* b тепер дорівнює З\*/ \*р= 0; /\*а тепер дорівнює О\*/

&\*а => а - розадресація.

Унарні оператори \* і & мають більш високий пріоритет, ніж арифметичні оператори:

b = \*р + 1 (взяти те, на що вказує р, додати до нього 1, а результат привласнити змінній b.

До покажчиків можна застосувати операцію привласнення. Покаж­чики одного і того ж типу можуть використовуватися в операції при­власнення, як і будь-які інші змінні. Розглянемо приклад. #include <stdio. h> void mai n() { int x= 1 0;

int \*p, \*g;

p=&x;

g=p;

printf("%p", р); /\* друк вмісту р \*/

printf("%p",g); /\* друк вмісту g \*/

р г і n t f (" % d % d ", x, \* g); / \* друк величини хі величини за адресою g\*/

**} Результат:** FFF4 FFF4 10 10

У цьому прикладі приведена ще одна специфікація формату функ­ції printf() - %р. Цей формат використовується для друку адреси пам'яті в шістнадцятковій формі.

Не можна створити змінну типу void, але можна створити покаж­чик на тип void. **Покажчику на void можна привласнити покажчик будь-якого іншого типу.** Однак при **зворотному привласненні необхід­но** використати явне **перетворення покажчика на void/­void \*pv;**

float f, \*pf;

pf=&f;

pv=pf;

pp=(fioat\*) pv;

У мові С допустимо привласнити покажчику будь-яку адресу па­м'яті. Однак, якщо оголошений покажчик на ціле

int \*р;

а за адресою, яка привласнена даному покажчику, знаходиться змінна х типу float, то при компіляції програми буде видане повідомлення про помилку в рядку

р=&х;

Цю помилку можна виправити, перетворювавши покажчик на int до типу покажчика на float явним перетворенням типу:

p=(int\*)&x;

Але при цьому втрачається інформація про те, на який тип вка­зував початковий покажчик.

Як і над іншими типами змінних, **над покажчиками можна виробля­ти арифметичні операції:** складання і віднімання (операції ++ і є окремими випадками операцій складання і віднімання). Арифметичні

дії над покажчиками мають свої особливості. Виконаємо найпростішу програму

#include <stdio. h> void main() { і n t x= 1 0;

int \*p, \*g;

p=&x;

g=p;

printf("%p", p); /\* друк вмісту p \*/ printf("%p", p++); /\* друк вмісту g \*/ } **Результат:** FFF4 FFF6

Після виконання цієї програми ми побачимо, що при операції ++1 значення покажчика р збільшилося не на 1, а на 2. І це правильне, оскільки нове значення покажчика повинно вказувати не на наступну адресу пам'яті, а на адресу наступного цілого. А ціле, як ми па­м'ятаємо, займає 2 байти. Якби базовий тип покажчика був не int, a double, то були б надруковані адреси, відмінні на 8 (Результат:

FFEE FFF6), саме стільки байт пам'яті займає змінна типу double, тобто при **кожній операції ++р значення покажчика буде** збільшувати­ся **на кількість байт, що займаються змінної базового** типу покажчи­ка .

Операції над покажчиками не обмежуються тільки операціями ++ і

--. До покажчиків можна додавати деяке ціле або відняти ціле. int \*p=2000; float \*p=2000;

Р=Р+3; р=р+10;

**Результат:** р=2006 **Результат:** р=2040

Загальна формула для обчислення значення покажчика після вико­нання операції р=р+п; буде мати вигляд

**<р>=<р>+п\*<кільк.байтів пам'яті базового типу покажчика>**

Можна також відняти один покажчик з іншого. Так, якщо р і pi -покажчики на елементи одного і того ж масиву, то операція р-рі дає такий же результат, як і віднімання індексів відповідних елементів масиву.

**Інші арифметичні операції** над покажчиками **заборонені,** напри­клад не можна **скласти два покажчики, помножити покажчик на число і т.д.**

|  |  |
| --- | --- |
| #include <std io. h > void rnai n() { int \*p, \*g, x; p=&x;  g=p;  printf("\n\n\np=%p", p); P= P + 8; printf(" p+5=%p", p); printf(" g=%p", g); printf(" p-g=%p", p-g);  } Результат: p=07DO p+5=07EO g=07DO p-g=0008 | #incl ude <std io. h> void main() { int \*p, \*g, x; p=&x;  g=p;  p r і n t f (" \ n \ n \ n p = % p ", p); P= P + 8; printf(" p+5=%p", p); printf(" g=%p", g); printf(" p+g=%p", p+g); } Результат: Error UKAZAT2.CPP 14: Invalid pointer addition |

Покажчики **можна порівнювати.** Застосовні всі 6 операцій:

**<, >, <=, >=, =, == і !=.**

Порівняння р < g означає, що адреса, що знаходиться в р, менше адреси, що знаходиться в g.

Якщо рід вказують на елементи одного масиву, то індекс еле­мента, на який вказує р, менше індексу масиву, на який вказує g.

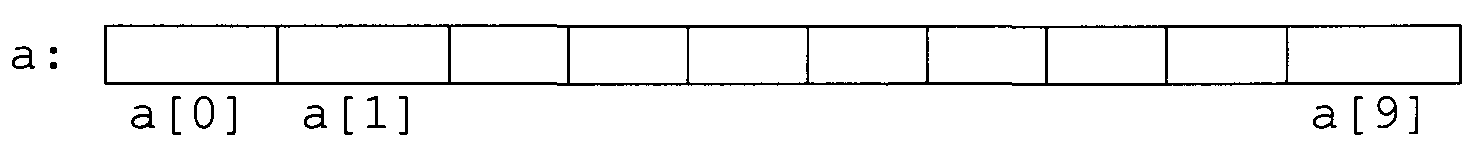
**ЗВ'ЯЗОК ПОКАЖЧИКІВ І МАСИВІВ**

Будь-який доступ до елемента масиву за допомогою операції ін­дексування може бути виконаний за допомогою покажчика (що в зага­льному випадку працює швидше).

Декларація

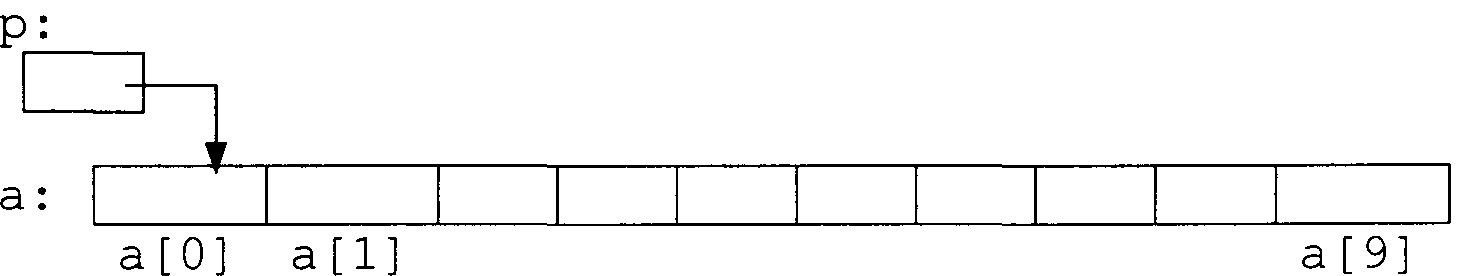
int a[10]

визначає масив а розміру 10:



Запис а[і] посилає нас до і-му елемента масиву. int \*р;

р=&а[0]; /\* р вказує на нульовий елемент а або містить адресу елемента а[0] \*/



х = \*р; => х = а[0], У= \*(Р+1); => У = а[1];

Значення змінної типу масив (ім'я **масиву)** є адреса **нульового елемента масиву.**

р = &а[0]; => р = а;

\*(а+і) ^ а[і] &а[і] => а+і

Результат буде один і той же. Перевага використання другого варіанту полягає в тому, що арифметичні операції над покажчиками виконуються швидше, якщо ми працюємо з підряд йдучими елементами масиву. Якщо ж вибір елементів масиву випадковий, то швидше і більш наочна робота з індексами.

**Між ім'ям масиву і покажчиком,**-виступаючим в ролі імені маси­ву, існує **одна відмінність.** Покажчик змінна, тому **можна написати р = а або р++.** Але ім'я масиву не є змінною, і записи типу а = р або **а++ не допускаються.**

Дуже часто доводиться працювати над обробкою текстів, т. е. з масивами рядків. Як ми пам'ятаємо, в мові С рядок - це масив сим­волів, що закінчується нульовим байтом. Розглянемо дві програми, що реалізовують практично, одні і ті ж дії.

#incl ude <std io. h>

#include <ctype.h>

void main()

{ char \*p, str[]="String From Letters in Different Registers";

/\* Рядок, що Складається з Букв в Різних Регістрах; \*/ int і=0; printf( "Рядок Буде Надрукований Заголовними Буквами");

while (str[i]) printf("%c", toupper(str[i++]));

p=str; printf(" Рядок Буде Надрукований Малими Буквами");

while (\*p) printf("%c", tolower(\*p++)); }

Якщо в цих прикладах замінити рядок на англійській мові на ря­док, набраний російськими буквами, то ніякого перетворення букв в рядкові або, навпаки, в прописні не станеться. Це пов'язано з тим, що стандартні функції toupper() і tolower () аналізують значення

10 аргументу і повертають те ж саме значення, якщо він не є відповід­но малою або великою буквою латинського алфавіту. Якщо ж аргумент є малою буквою латинського алфавіту, то значенням функції toupper() буде відповідна велика буква (точніше, код цієї букви). Функція tolower () змінює код лише великих букв латинського алфаві­ту. Прототипи цих функцій знаходяться в заголовному файлі ctype.h.

**МАСИВИ ПОКАЖЧИКІВ**

Покажчики, як і змінні будь-якого іншого типу, можуть об'єдну­ватися в масиви. Оголошення масиву покажчиків на 10 цілих чисел має вигляд int \*x[10] ;

Кожному з елементів масиву можна привласнити адресу; наприклад, третьому елементу привласнимо адресу цілої змінної у:

х[2]=&у;

щоб знайти значення змінною у, можна написати \*х(2].

Наведемо приклад використання масиву покажчиків. Частіше за все це буває зручно при обробці масиву рядків.

/\* you must run. exe-file to watch the rezult of this program. Перегляд файлів в поточному каталозі з одним з шести розширень \*/

#include <std io. h >

#include <string.h> ^include <stdlib. h>

#include <conio. h>

main()

{char ch, s[80], \*ext[]={"exe", "corn", "cpp", "c", "pas", "\*"};

clrscr();

for(;;) {do { printf( "Файли з розширенням:^");

printf("1. exe\n"); "printf( 2. com\n"); "printf( 3. cpp\n"); "pnntf( 4. з \ n ");

printf("5. pas\n"); printf("6. \*\n"); //any extension printf("7. quit\n");

printf("BauJ вибір(1-7):)( \n");

ch=getche();

printf("\n");

} while (ch<'1' ;! ch>'7');

if (ch=='7') break;

strcpy(s, "dir \*."); strcat(s, ext[ch-'0'-1 ]); strcat(s, "/p"); system(s);} }

Тут функція system() - бібліотечна функція, яка примушує опе­раційну систему DOS виконати команду, що є аргументом цієї функ­ції.

Взагалі рядкова константа в мові С асоціюється з адресою початку рядка в пам'яті, тип рядка виходить char\* (покажчик на тип char). Тому можливо і активно використовується наступне привласнення:

char *\*pc;*

"рс = Hello, World!";

У мові С можлива також ситуація, коли покажчик вказує на пока­жчик. У цьому випадку опис буде мати наступний вигляд:

int -\*'\*point;



point має тип покажчик на покажчик на int. Відповідно, щоб набу­ти цілочисельного значення змінною, на яку указьіваеі point, треба у вираженні використати \*\*point.;

Приклад використання:

11

^include <stdio. h>

void m а і n()

{ int i, pi, ppi;

і =7; pi=&i;

p p i = & p i;

printf( "i = %d pi = %p ppi = %p \n", i, pi, ppi);

\*pi++;

printf( "i = %d pi = %p ppi = %p \n", i, pi, ppi);

\*\*ppi = 12;

printf( "i = %d pi = %p ppi = %p \n", i, pi, ppi);

}

**ІНІЦІАЛІЗАЦІЯ ПОКАЖЧИКІВ**

Після того як покажчик був оголошений, але до того, як йому було привласнене якесь значення, покажчик містить невідоме значен­ня. **Спроба використати покажчик до привласнення йому якогось зна­чення є** неприємною **помилкою,** оскільки вона **може порушити роботу не**

тільки вашої **програми,** але **і операційної системи.** Навіть якщо цьо­го не сталося, результат роботи програми буде неправильним і знай­ти цю помилку буде досить складно.

Вважають, що покажчик, який вказує в "нікуди", повинен мати значення null, однак і це не робить його "безпечним". Після того, як він попаде в праву або ліву частину оператора привласнення, він знову може стати "небезпечним".

З іншого боку нульовий покажчик можна використати, наприклад, для позначення кінця масиву покажчиків.

Якщо була спроба привласнити яке-небудь значення тому, на що вказує покажчик з нульовим значенням, система видає попередження, що з'являється під час роботи програми (або після закінчення робо­ти програми) "Null pointer assignment". Поява цього повідомлення є мотивом для пошуку використання неініціалізувати покажчика в програмі.