Указатели

Базово програмиране
Лекция 9
Милена Карова
кат. КНТ ТУ-Варна

Дефиниция

- Указателите са адресите на променливата в паметта. Адресът, който се използва за да се именува променливата (за да се посочи променливата) е указател. Той посочва мястото, където се записва променливата в паметта.
- Указателите се използват като фактически параметри при извикване на функцията, когато параметрите се подават по адрес.
- Указателят се записва като променлива
- Тъй като това е адрес от паметта, следователно в променливата се записва число
- В този пример указателят е от тип double

```
double *p;
```

тъй като в клетката чийто адрес сочи p съдържа double число

Дефиниране

- Самата стойност на указателя може да бъде само int

```
int *p1, *p2, v1, v2;
```

- p1 и p2 съдържат адреси на клетки, в които има цели числа
- Не пропускайте * пред името на указателя, в противен случай ще се превърне в обикновена променлива
- Ако p1 съдържа адреса на променливата v1, то казваме че p1 сочи v1 или p1 е указател на v1

Указател и адрес на променлива

- Може да се ползва символа & за да се определи адреса на променливата и той да се присвои на указателя

```
p1 = &v1;
```

- P1 сочи адреса на променливата v1
- Указателят е един адрес, а адресът не е цяло число. Това е абстракция. C++ „настоява“ указателят да се използва като адрес, но не като цяло число. Указателят не е стойност от тип цял или който и да е тип. Нормално не можете да запишете указателят в обикновена променлива от тип int. Също не можете да извършвате нормални аритметични действия. (Възможна е указателна аритметика: събиране и изваждане)

Оператор *

- Ако е дефинирана променливата v1, можете да се обърнете: 1) това е променливата v1 или 2)променливата сочена от p1 (*p1)
- Тук * има по-друго значение (dereferencing operator). Променливата се нарича „сочена“
- ще се получи

```
v1 = 0;
p1 = &v1;
*p1 = 42;
cout << v1 << endl;
cout << *p1 << endl;
```

- 42
- 42
- *p1 – съдържание на клетката, чийто адрес сочи p1
- double *p, v; p=&v; *p=99.9;

Операции с указатели

- Ако указателят p1 сочи променливата v1, а p2 – v2, то следната операция е напълно възможна

```
p2 = p1;
```

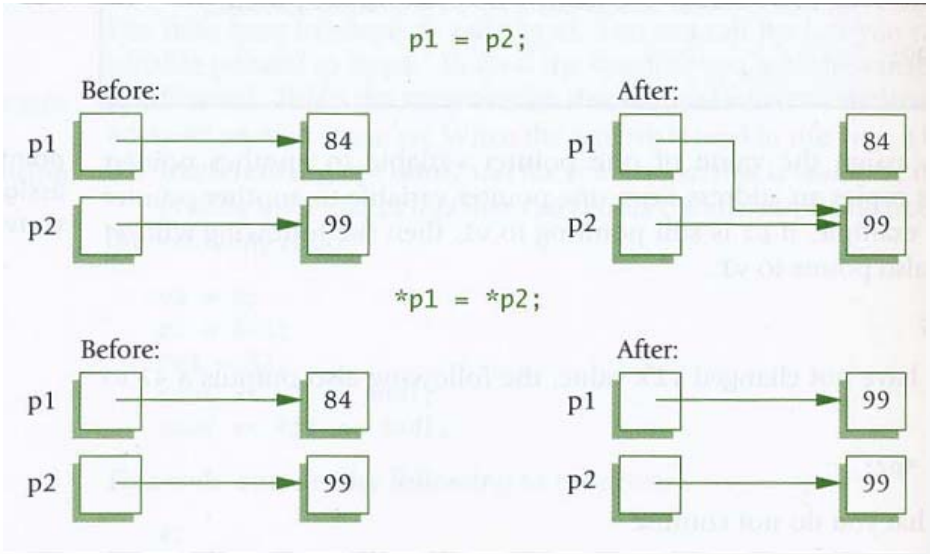
```
cout << *p2;
```

```
*p1 = *p2;
```

- В последния пример реално не се работи с указателите, а със стойностите на променливите,

които сочат

-



Динамични променливи

- Оператор new – създава нова променлива като резервира място в паметта, без да се знае идентификатора на променливата

```
p1 = new int;
```

- Създава се нова променлива от тип int и указателят p1 сочи нейния адрес
- С новата променлива

```
cin >> *p1;
*p1 = *p1 + 7;
cout << *p1;
```

се извършват действия без тя да има име

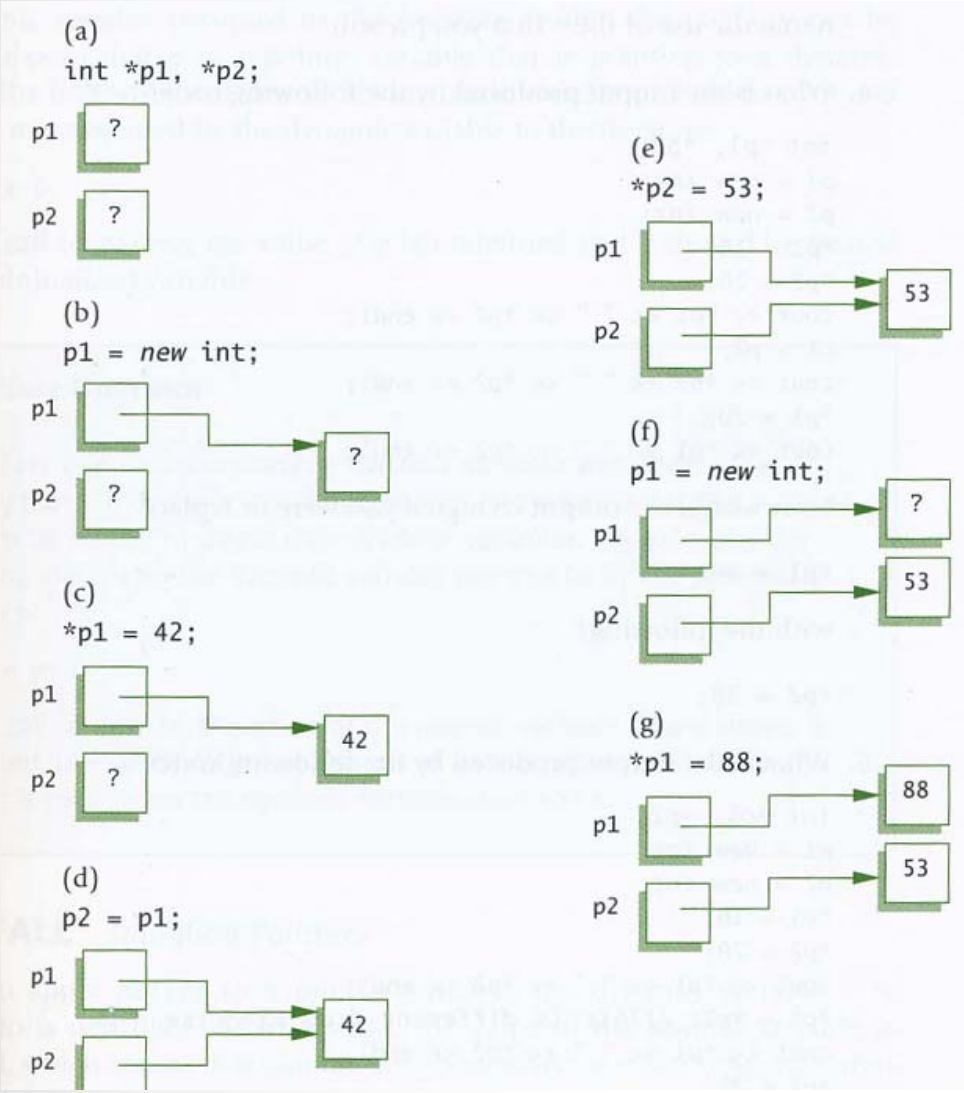
- Променливите, създадени посредством new наричат динамични, защото се създават и унищожават по време на изпълнение на програмата

Основни операции с указатели

```
#include <iostream>
using namespace std;

int main()
{
    int *p1, *p2;

    p1 = new int;
    *p1 = 42;
    p2 = p1;
    cout << "*p1 == " << *p1 << endl;
    cout << "*p2 == " << *p2 << endl;

    *p2 = 53;
    cout << "*p1 == " << *p1 << endl;
    cout << "*p2 == " << *p2 << endl;

    p1 = new int;
    *p1 = 88;
    cout << "*p1 == " << *p1 << endl;
    cout << "*p2 == " << *p2 << endl;
    cout << "Hope you got the point of this example!\n";
    return 0;
}
```

```
*p1 == 42
*p2 == 42
*p1 == 53
*p2 == 53
*p1 == 88
*p2 == 53
Hope you got the point of this example!
```

Графично представяне на операциите с указатели



- Мястото в паметта, отделено за динамични променливи се нарича freestore или heap.
- Операторът delete елиминира динамичните променливи и увеличава размера на freestore.
- Същата памет може да се използва за други динамични променливи
- Недефиниран указател при изтрита динамична променлива (dangling pointers)

Имена на масиви като указатели

- Можем да се обръщаме към масивите като към указатели и обратно (масиви от низове)
- Името на масив е указател
- Ако е даден масив int ara[5]={10,20,30,40,50};
- cout<<ara [0]; и cout<<*ara; са еквивалентни, получавате 10
- Обръщане към елемент от масива чрез индекс или чрез операция за определяне на съдържанието на указател
- cout<<*(ara+2); извеждане на третия елемент на масива ara[2] без използване на индекс
- Името на масива е просто указател към адреса на първия елемент от масива. Прилагате операция * съдържание на указател за да получите стойността на елемента

Вътрешно представяне на данните

- Ако напишете *(ara+3)C++ добавя 12 байта към адреса на ara за да се обърнете към елемента ara[3] и да получите третия елемент.
- Добавя се 3*sizeof(int) байта към адреса на първия елемент на масива
- Аритметичните действия над указателите дават възможност за да се посочва всяка стойност от списък, чийто първи елемент се посочва от указател т.е. ara[0], *ara и *(ara+5) се отнасят за един и същи елемент.
- Използването на указатели за достъп до елементите на масивите е по-ефективно от индексите.

Предимства на указателите

- Адресите указани от масиви не могат да бъдат променяни за разлика от адресните стойности на указателите
- Масивите са по-лесни за деклариране, затова първо масивите се декларират, а след това използват указатели за обръщение към масивите
- Какъв е изхода от следната програма?

```
void main()
{ int ctr;
  int iara[5]={10,20,30,40,50};
  int *iptr;  iptr=iara; //iptr=&iara[0];
  for(ctr=0;ctr<5;ctr++)
cout<<endl<<iara[ctr]<<"\t"<<iptr[ctr];
//използване на указатели
for(ctr=0;ctr<5;ctr++)

cout<<endl<<*(iara+ctr)<<"\t"
<<*(iptr+ctr);
```

```
10 10
20 20
30 30
40 40
50 50
```

Адресна аритметика

- Указателите могат да бъдат инкрементирани и декрементирани.
- При инкрементиране на указател се увеличава адресът, съдържащ се в него. Указателят не винаги се увеличава точно с 1.
- Към адреса на указателя се добавят толкова байта, колкото заема променливата към чийто адрес сочи указателя.
- При декрементиране се изважда размера на данните от типа на указателя, а не 1.

Пример за адресна аритметика

```
void main()
{ int
iara[5]={10,20,30,40,50};
   int *ip=iara;
   cout<<endl<<*ip; ip++;
   cout<<endl<<*ip; ip++;
   cout<<endl<<*ip; ip++;
   cout<<endl<<*ip; ip++;
   cout<<endl<<*ip;
}
```

Резултати:
10
20
30
40
50