Увод в програмирането

Типове указател и псевдоним

Тип указател

- Променлива, това е място за съхранение на данни, което може да съдържа различни стойности. Идентифицира се с дадено от потребителя име (идентификатор). Има си и тип. Дефинира се като се указват задължително типът и името ѝ. Типът определя броя на байтовете, в които ще се съхранява променливата, а също и множеството от операциите, които могат да се изпълняват над нея.
- Освен това, с променливата е свързана и стойност – неопределена или константа от типа, от който е тя. Нарича се още **rvalue**.
- Мястото в паметта, в което е записана rvalue, се идентифицира с адрес, който се нарича адрес на променливата или **lvalue**. По-точно адресът е адреса на първия байт от множеството байтове, отделени за променливата.

Тип указател …

- *Пример:*

  ```
  int i = 1024;
  ```

  дефинира променлива с име i и тип int. Стойността ѝ (rvalue) е 1024. i именува място от паметта (lvalue) с размери 4 байта, като lvalue е адреса на първия байт на това място.
- Намирането на адреса на дефинирана променлива става чрез унарния префиксен дясно-асоциативен оператор **&** (амперсанд). Приоритетът му е същия като на унарните оператори +, -, !, ++, -- и др.

Тип указател …

- **Оператор &**
- *Синтаксис*

  &<променлива>

  където <променлива> е вече дефинирана променлива.
- *Семантика*

  Намира адреса на <променлива>.
- *Пример:*

  &i е адреса на променливата i и може да се изведе чрез оператора cout << &i;

Тип указател …

- Операторът & не може да се прилага върху константи и изрази, т.е. &100 и &(i+5) са недопустими обръщения. Не е възможно също прилагането му и върху променливи от тип масив, тъй като те имат и смисъла на *константни* указатели.
- Адресите могат да се присвояват на специален тип променливи, наречени променливи от тип указател или само указатели.

Тип указател …

- **Дефиниране**

  Нека T е име или дефиниция на тип. За типа T, T* е тип, наречен указател към T. T се нарича **указван тип** или **тип на указателя**.

- *Примери:*

  int* е тип указател към тип int;

  enum {a, b, c}* е тип указател към тип enum {a, b, c}.

Тип указател …

- **Множество от стойности**

  Състои се от адресите на данните от тип T, дефинирани в програмата, преди използването на T*. Те са константите на типа T*. Освен тях съществува специална константа с име NULL, наречена **нулев указател**. Тя може да бъде свързвана с всеки указател независимо от неговия тип. Тази константа се интерпретира като “сочи към никъде”, а в позиция на предикат е false.

Тип указател …

- Променлива величина, множеството от допустимите стойности, на която съвпада с множеството от стойности на типа T*, се нарича **променлива от тип T***, **променлива от тип указател към тип T** или само **указател към тип T**. Дефинира се по общоприетия начин.

Тип указател …

- **Дефиниция на променлива от тип указател**

T *<променлива> [= <стойност>]
{,*<променлива> [= <стойност>]};

където

- T е име или дефиниция на тип;
- <променлива> е идентификатор;
- <стойност> е (шестнадесетично) число, представляващо адрес на данна от тип T или NULL.

Тип указател …

- T определя типа на данните, които указателят адресира, а също и начина на интерпретацията им.
- Дефиницията

  ```
  T* a, b;
  ```

  е еквивалентна на

  ```
  T* a;
  T b;
  ```

  т.е. само променливата a е от тип указател към тип T.

Тип указател …

- *Примери:*

  ```
  int *pint1, *pint2;
  ```

  задава два указателя към тип int

  ```
  int *pint1, pint2;
  ```

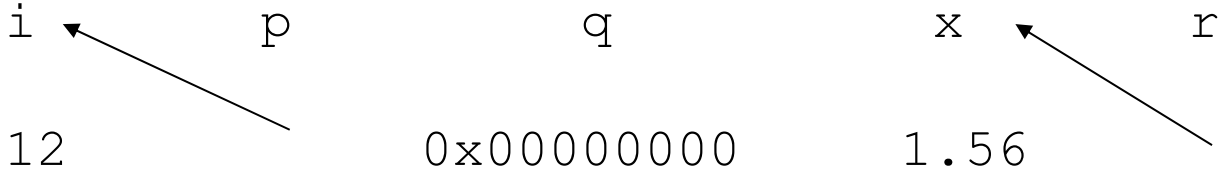
  указател pint1 към int и променлива pint2 от тип int.
- Дефиницията на променлива от тип указател предизвиква в ОП да се отделят 4B, в които се записва някакъв адрес от множеството от стойности на съответния тип, ако дефиницията е с инициализация и неопределено или NULL, ако дефиницията не е с инициализация. Този адрес е **стойността** на променливата от тип указател, а записаното на този адрес е **съдържанието** ѝ.

Тип указател …

- *Пример:* В резултат от изпълнението на дефинициите

```
int i = 12;
int* p = &i;       // p е инициал. с адреса на i
double *q = NULL; // q е инициал. с нулевия указател
double x = 1.56;
double *r = &x;    // r е инициал. с адреса на x
```

ОП



Тип указател …
Операции и вградени функции

- **Извличане на съдържанието на указател**

  Осъществява се чрез префиксния, дясноасоциативен унарен оператор *. * има приоритет на унарен оператор.
- **Оператор ***
- *Синтаксис*

  **<променлива_от_тип_указател>*
- *Семантика*

  Извлича стойността на адреса, записан в <променлива_от_тип_ указател>, т.е. съдържанието на <променлива_от_тип_ указател>.

Тип указател …
Операции и вградени функции

- Като използваме дефинициите от примера по-горе, имаме:

*p е 12 // 12 е съдържанието на p

*r е 1.56 // 1.56 е съдържанието на r

- Освен, че намира съдържанието на променлива от тип указател, обръщението

  *<променлива_от_тип_указател>

  е променлива от тип T. (Всички операции, допустими за типа T, са допустими и за нея.
- Като използваме дефинициите от примера по-горе, *p и *r са цяла и реална променливи, съответно. След изпълнение на операторите за присвояване

```
*p = 20;
*r = 2.18;
```

стойността на i се променя на 20, а тази на r – на 2.18.

Тип указател …
Операции и вградени функции

- **Аритметични и логически операции**
- Променливите от тип указател могат да участват като операнди в следните аритметични и логически операции +, -, ++, --, ==, !=, >, >=, < и <=. Изпълнението на аритметични операции върху указатели е свързано с някои особености, заради което аритметиката с указатели се нарича още адресна аритметика. Особеността се изразява в т. нар. мащабиране. Ще го изясним чрез пример.

```
int *p;
double *q;
...
p = p + 1;
q = q + 1;
```

Тип указател …
Операции и вградени функции

- Операторът p = p + 1; свързва p не със стойността на p, увеличена с 1, а с p + 1*4, където 4 е броя на байтовете, необходими за записване на данна от тип int (p е указател към int). Аналогично, q = q + 1; увеличава стойността на q не с 1, а с 8, тъй като q е указател към double (8 байта са необходими за записване на данна от този тип).
- Общото правило е следното: Ако p е указател от тип T*, p+i е съкратен запис на p + i*sizeof(T), където sizeof(T) е функция, която намира броя на байтовете, необходими за записване на данна от тип T.

Тип указател …
Операции и вградени функции

- **Въвеждане**

  Не е възможно въвеждане на данни от тип указател чрез оператора cin. Свързването на указател със стойност става чрез инициализация или оператора за присвояване.
- **Извеждане**

  Осъществява се по стандартния начин - чрез оператора <<.

Тип указател …
Операции и вградени функции

```
include <iostream.h>
int main()
{  int n = 10;   // дефинира и инициализира цяла променлива
   int* pn = &n; // дефинира и инициализира указател pn към n
   // показва, че указателят сочи към n
   cout << "n=  " << n << "  *pn= " << *pn << '\n';
   // показва, че адресът на n е равен на стойността на pn
   cout << "&n= "<< &n << "  pn= " << pn << '\n';
   // намиране на стойността на n чрез pn
   int m = *pn;        //  m == 10
   // промяна на стойността на n чрез pn
   *pn = 20;
   // извеждане на стойността на n
   cout << "n= " << n << '\n'; // n == 20
   return 0;
}
```

Тип указател … Допълнение

- В някои случаи е важна стойността на променливата от тип указател (адресът), а не нейното съдържание. Тогава тя се дефинира като указател към *тип void*. Този тип указатели са предвидени с цел една и съща променлива - указател да може в различни моменти да сочи към данни от различен тип. В този случай, при опит да се използва съдържанието на променливата от тип указател, ще се предизвика грешка.
- Съдържанието на променлива - указател към тип void може да се извлече само след привеждане на типа на указателя (void*) до типа на съдържанието. Това може да се осъществи чрез операторите за преобразуване на типове.

Тип указател … Допълнение

- *Пример:*

```
int a = 100;
void* p;    // дефинира указател към void
p = &a;     // инициализира p
cout << *p; // грешка
cout << *((int*) p);
// преобразува p в указател към int
// и тогава извлича съдържанието му.
```

Тип указател … Допълнение

- В дефиницията:

  T* const <идентификатор>;

  <идентификатор> е константен указател към тип T и не може да бъде променяна стойността му.
- В дефиницията:

  const T* <идентификатор>;

  <идентификатор> е указател към константа от тип T и не може да бъде променяно съдържанието му.

Тип указател … Допълнение

- *Пример:*

```
int i, j = 5;
int *pi;       // pi е указател към int
int * const b = &i;
     // b е константен указател към int
const int *c = &j;
     // c е указател към цяла константа.
b = &j;
     // грешка, b е константен указател
*c = 15;
     // грешка, *c е константа
```

Указатели и масиви

- В C++ има интересна и полезна връзка между указателите и масивите. Изразява се в това, че имената на масивите са указатели към техните “първи” елементи. Последното позволява указателите да се разглеждат като алтернативен начин за обхождане на елементите на даден масив.
- *Указатели и едномерни масиви*

  Нека a е масив, дефиниран по следния начин:

```
int a[100];
```

Указатели и масиви …

- Тъй като a е указател към a[0], *a е стойността на a[0],

  т.е. *a и a[0] са два различни записа на стойността на първия елемент на масива.
- Тъй като елементите на масива са разположени последователно в паметта,
  a + 1 е адреса на a[1],
  a + 2 е адреса на a[2] и т.н.
  a + n-1 е адреса на a[n-1].

  Тогава *(a+i) е друг запис на a[i] (i = 0, 1, ..., n-1).

Указатели и масиви …

- Има обаче една особеност. Имената на масивите са константни указатели. Заради това, някои от аритметичните оператори, приложими над указатели, не могат да се приложат над масиви. Такива са ++, -- и присвояването на стойност.

```
int a[] = {1, 2, 3, 4, 5, 6};
int i;
for (i = 0; i <=  5; i++)
   cout << a[i] << '\n';
for (i = 0; i <=  5; i++)
   cout << *(a+i) << '\n';
```

Указатели и масиви …

- Операторът

```
for (i = 0; i <= 5; i++)
{ cout << *a << '\n';
  a++;
}
```

съобщава за грешка заради a++ (a е константен указател и не може да бъде променян

```
int* p = a;
for (i = 0; i <= 5; i++)
{ cout << *p << '\n';
  p++;
}
```

Указатели и масиви …

- **Указатели и двумерни масиви**
- Името на двумерен масив е константен указател към първия елемент на едномерен масив от константни указатели. Ще изясним с пример казаното.
- Нека a е двумерен масив, дефиниран по следния начин:

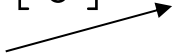
  ```
  int a[10][20];
  ```

  Променливата a е константен указател към първия елемент на едномерния масив a[0], a[1], ..., a[9], като всяко a[i] е константен указател към a[i][0] (i = 0,1, ..., 9)

Указатели и масиви …

```
int a[10][20];
a
```

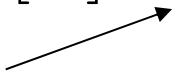
```
a[0]  a[0][0]  a[0][1]  …  a[0][19]
```



```
a[1]  a[1][0]  a[1][1]  …  a[1][19]
```



```
…
a[9]  a[9][0]  a[9][1]  … a[9][19]
```



Указатели и масиви …

- Тогава

**a == a[0][0]

a[0] == *a a[1] == *(a + 1) ... a[9] == *(a + 9),

т.е.

a[i] == *(a + i)

- Като използваме, че операторът за индексиране е лявоасоциативен и с по-висок приоритет от оператора *, получаваме:

  a[i][j] == (*(a+i))[j] == *(*(a+i)+j).

Указатели и низове

- Низовете са масиви от символи. Името на променлива от тип низ е константен указател, както и името на всеки друг масив. Така всичко, което казахме за връзката между масив и указател е в сила и за низ – указател.

```
...
char str[] = "C++Language";
   // str е константен указател
char* pstr = str; // pstr е указател към низа str
while (*pstr)
{   cout << *pstr << '\n';
    pstr++;
}
// pstr вече не е свързан с низа “C++Language”.
```

Указатели и низове …

```
#include <iostream.h>
int main()
{  char* str = "C++Language";
        // str е променлива
   while (*str)
   {   cout << *str << '\n';
       str++;
   }
   return 0;
}
```

Указатели и низове …

- Примерите показват, че задаването на низ като указател към char има предимство пред задаването като масив от символи. Ще отбележим обаче, че дефиницията

```
char* str = "C++Language";
```

не може да бъде заменена с

```
char* str;
cin >> str;
```

докато

```
char str[20];
cin >> str;
```

е допустимо.

Тип псевдоним

- Чрез псевдонимите се задават алтернативни имена на обекти в общия смисъл на думата (променливи, константи и др.).
- **Дефиниране**

  Нека T е име на тип. Типът T& е тип псевдоним на T. T се нарича **базов тип** на типа псевдоним.
- **Множество от стойности**

  Състои се от всички имена на дефинирани вече променливи от тип T.
- *Пример:*

```
int a, b = 5;
int x, y = 9, z = 8;
...
```

  Множеството от стойности на типа int& съдържа имената a, b, x, y, z.

Тип псевдоним …

- Променлива величина, множеството от допустимите стойности на която съвпада с множеството от стойности на даден тип псевдоним, се нарича променлива от този тип псевдоним.
- **Дефиниране на променлива от тип псевдоним**

<дефиниране_на_променлива_от_тип_псевдоним> ::=
T &<променлива> = <вече_дефинирана_променлива_от_тип_T>
{, &<променлива> = <вече_дефинирана_променлива_от_тип_T>};

където T е име тип. Променливата след знак = се нарича инициализатор.

Тип псевдоним …

- *Забележка:*Дефиницията

  ```
  T& a = b, c = d;
  ```

  е еквивалентна на

  ```
  T& a = b;
  T c = d;
  ```

  т.е. операторът & след типа T се отнася само за първата променлива след него.
- *Пример:*

  ```
  int a = 5;
  int& syna = a;
  double r = 1.85;
  double &syn1 = r, &syn2 = r;
  int& syn3 = a, syn4 = a;
  ```

Тип псевдоним …

- **Дефиницията на променлива от тип псевдоним задължително е с инициализация – дефинирана променлива от същия тип като на базовия тип на типа псевдоним.** След това не е възможно променливата-псевдоним да стане псевдоним на нова променлива. Затова тя е “най-константната” променлива, която може да съществува.
- *Пример:*

```
int a = 5;
int &syn = a; // syn е псевдоним на a
int b = 10;
int& syn = b; // error, повторна дефиниция
```

Тип псевдоним …

- **Операции и вградени функции**

  Дефиницията на променлива от тип псевдоним свързва променливата-псевдоним с инициализатора и всички операции и вградени функции, които могат да се прилагат над инициализатора, могат да се прилагат и над псевдонима му и обратно.
- *Примери:*

```
int ii = 0;
int& rr = ii;
rr++;
int* pp = &rr;
```

Тип псевдоним …

```
int a = 5;
int &syn = a;
cout << syn << "  " << a << '\n';
int b = 10;
syn = b;
cout << b << "  " << a << "  " << syn << '\n';
```

извежда
5 5
10 10 10
Операторът syn = b; е еквивалентен на a = b;.

Тип псевдоним …

```
int i = 1;
int& r = i;     // r и i са свързани с 1
cout << r;      // извежда 1
int x = r;      // x има стойност 1
r = 2;     // еквивалентно е на i = 2;
```

- *Забележка:* В някои случаи компилаторът може да оптимизира кода, като отстрани псевдонима. Тогава по време на изпълнение на програмата не съществува обект, представляващ псевдонима.

Тип псевдоним …

- **Константни псевдоними**

  В C++ е възможно да се дефинират псевдоними, които са константи. За целта се използва запазената дума const, която се поставя пред дефиницията на променливата от тип псевдоним. По такъв начин псевдонимът не може да променя стойността си, но ако е псевдоним на променлива, промяната на стойността му може да стане чрез промяна на променливата.
- *Пример:*

```
int i = 125;
const int& syni = i;
cout << i << " " << syni << '\n';
                // i и syni имат стойност 125
syni = 25;      // грешка
cout << i << " " << syni << '\n';
```

Тип псевдоним …

```
int i = 125;
const int& syni = i;
cout << i << " " << syni << '\n';
i = i + 25;
cout << i << " " << syni << '\n';
```

ще изведе

125 125

150 150

Последното показва, че константен псевдоним на променлива защитава промяната на стойността на променливата чрез псевдонима.