AAB University

Faculty of Computer Sciences

Programimi i Orientuar në Objekte

Java 9:

Ponterët

Doc. Dr. **Mentor Hamiti**

mentor.hamiti@universitetiaab.com

- Pointerët
- Operatorët
- Qasjet referente
- ***const*** në kuadër të pointerëve
- Operatori **sizeof**
- Shprehjet dhe aritmetika me pointerë
- Relacioni mes Pointerëve dhe fushave

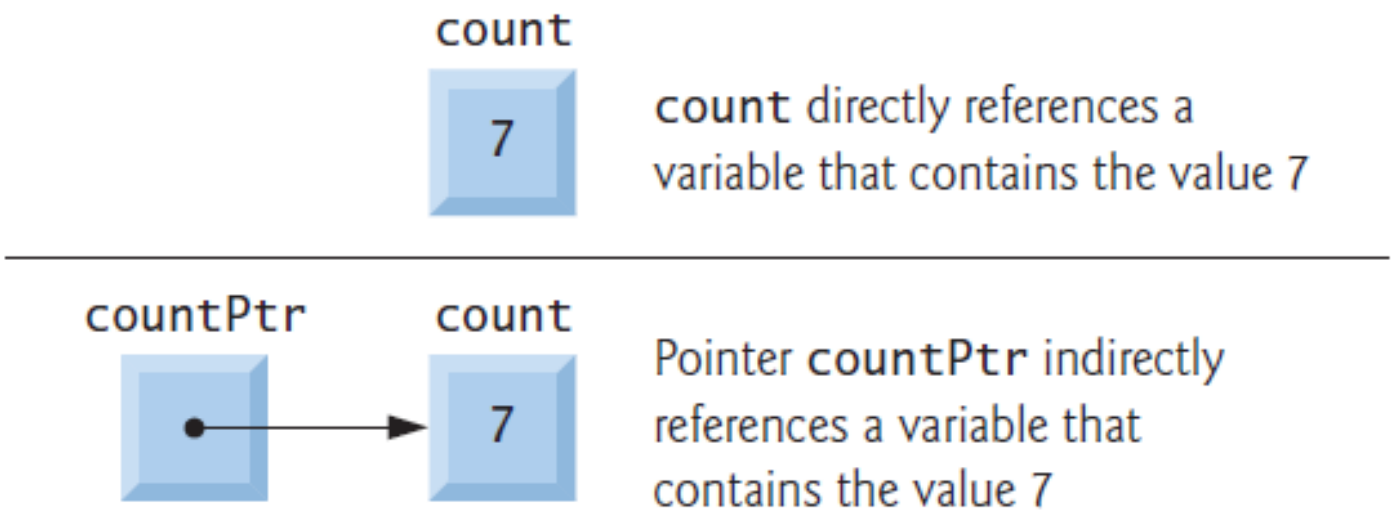
Pointerët

- Shfrytëzimi i pointerëve është sfidues në C++
- Duhet shfrytëzuar në mënyrë korekte dhe me përgjegjësi
- Ata mundësojnë shfrytëzimin dinamik të memorjes, siç ishte rasti i listave, stekut, rradhës dhe pemëve
- Gjithashtu shfrytëzimi i tyre mund të ndërlidhet edhe me fushat

Pointerët

- **Mënyra indirekte**
- Pointeri përmban adresën ku ndodhen variablat, në të cilat vendosen vlerat konkrete
- Në këtë kontekst, emri i variablës në mënyrë direkte i referohen vlerës, kurse pointeri në mpnyrë indirekte I referohet të njejtës vlerë
- Referimi i vlerës përmes pointerëve është e njohur si **mënyrë indirekte**
- *Pointerët mund të deklarohen që t'iu referohen objekteve ose të dhënave të ndryshme*

- Mënyra direkte dhe indirekte e referimit të vlerave



Pointerët

- Shprehja:

```
int  *countPtr,  count;
```

  - Deklaron variablën countPtr të tipit int * dhe lexohet "countPtr është pointer në int"

- Variabla count është deklaruar të jetë e tipit int, e jo pointer në int

- Simboli * vlenë vetëm për countPtr

  - *Pra * nuk është operator shumëzimi në këtë rast;*

Common Programming Error

Assuming that the * used to declare a pointer distributes to all names in a declaration's comma-separated list of variables can lead to errors. Each pointer must be declared with the * prefixed to the name (with or without spaces in between). Declaring only one variable per declaration helps avoid these types of errors and improves program readability.

Good Programming Practice

Although it's not a requirement, including the letters `Ptr` in a pointer variable name makes it clear that the variable is a pointer and that it must be handled accordingly.

Pointerët

- **Inicializimi i Pointerëve**
- Pointerët mund të inicializohen si **nullptr** (*C++11*)
- Pointeri me vlerë **nullptr** "*nuk tregon asgjë*" dhe është i njohur si **pointer zero**

Error-Prevention

Initialize all pointers to prevent pointing to unknown or uninitialized areas of memory.

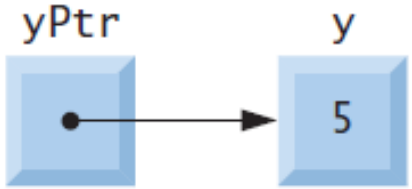
Pointerët

- Në versionet paraprake të C++, null pointeri është preferuar të deklarohet me **0** ose **NULL**
  - *Inicializimi i pointerit me NULL është ekuivalent me 0*

- **Operatori i adresës (&)**
- **Operatori** (**&**) është operator unarë që tregon adresën përkatëse

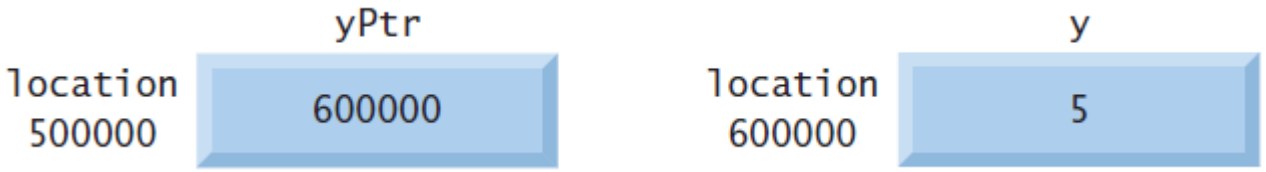
```
int   y = 5;                  // deklarimi i variablës y
int  *yPtr = nullptr;         // deklarimi  i pointerit yPtr
yPtr = &y;                    // adresa e y i ndahet yPtr
```

- Inerpretimi grafik i pointerit që tregon lokacionin memorues:



Operatorët

- Shembull:
  - Interpretimi i **pointerit** në variablën **y** i ruajtur në lokacionin memorues **600000** dhe i variablës **yPtr** të ruajtur në lokacionin **500000**
    - Prezentimi i **y** dhe **yPtr** në kuadër të memorjes:



Operatorët

- **Operatori indirekt** (*)
- Operatori unarë * - zakonisht nënkuptohet si operatorë indirekt

```
int   y = 5;
int  *yPtr = nullptr;
yPtr =  &y;
*yPtr =  9;

cout<<yPtr<<endl;
cout<<&y<<endl;
cout<<*yPtr<<endl;
cout<<y<<endl;
```

```
004EFB44
004EFB44
9
9
```

Operatorët

- Shembulli 1:

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

int main()
{
    int a; // a is an integer
    int *aPtr; // aPtr is an int * which is a pointer to an integer
    a = 7; // assigned 7 to a
    aPtr = &a; // assign the address of a to aPtr

    cout << "The address of a is " <<&a
         << "\nThe value of aPtr is " << aPtr;
    cout << "\n\nThe value of a is " << a
         << "\nThe value of *aPtr is " << *aPtr;
    cout << "\n\nShowing that * and & are inverses of "
         << "each other.\n&*aPtr = " << &*aPtr
         << "\n*&aPtr = " << *&aPtr << endl;

    cin.get(); return 0;
}
```

```
The address of a is 0025FAA0
The value of aPtr is 0025FAA0

The value of a is 7
The value of *aPtr is 7

Showing that * and & are inverses of each other.
&*aPtr = 0025FAA0
*&aPtr = 0025FAA0
```

Operatorët

- (&) (*) janë operatorë unarë të nivelit të katërt

Operators	Associativity	Type
:: ()	*[See caution in Fig. 2.10]*	highest
() []	left to right	function call/array access
++ -- static_cast<*type*>(*operand*)	left to right	unary (postfix)
++ -- + - ! & *	right to left	unary (prefix)
* / %	left to right	multiplicative
+ -	left to right	additive
<< >>	left to right	insertion/extraction
< <= > >=	left to right	relational
== !=	left to right	equality
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment
,	left to right	comma

Qasjet referente

- Egzistojnë tri mënyra në C++ për ndarje të argumenteve të funksioneve:
  - 1. Përmes vlerës,
  - 2. Përmes referencës
  - 3. Përmes pointerit

- Shembulli 2:

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

int cubeByValue( int ); // prototype

int main()
{
    int number = 5;

    cout << "The original value of number is " << number;
    number = cubeByValue( number ); // pass number by value to cubeByValue
    cout << "\nThe new value of number is " << number << endl;

    cin.get();  return 0;
}

// calculate and return cube of integer argument
int cubeByValue( int n )
{
return n * n * n;   // cube local variable n and return result
};
```

```
The original value of number is 5
The new value of number is 125
```

Qasjet referente

- Shembulli 3:

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

void cubeByReference( int * ); // prototype

int main()
{
    int number = 5;

    cout << "The original value of number is " << number;
    cubeByReference( &number ); // pass number address to cubeByReference
    cout << "\nThe new value of number is " << number << endl;

    cin.get();  return 0;
}

// calculate cube of *nPtr; modifies variable number in main
void cubeByReference( int *nPtr )
{
*nPtr = *nPtr * *nPtr * *nPtr; // cube *nPtr
} // end function cubeByReference
```

```
The original value of number is 5
The new value of number is 125
```

Analiza grafike e qasjes përmes vlerës (Sh. 2)

Step 1: Before `main` calls `cubeByValue`:

```
int main()
{
   int number = 5;

   number = cubeByValue( number );
}
```

number: 5

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

n: undefined

Step 2: After `cubeByValue` receives the call:

```
int main()
{
   int number = 5;

   number = cubeByValue( number );
}
```

number: 5

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

n: 5

Step 3: After `cubeByValue` cubes parameter `n` and before `cubeByValue` returns to `main`:

```
int main()
{
   int number = 5;

   number = cubeByValue( number );
}
```

number: 5

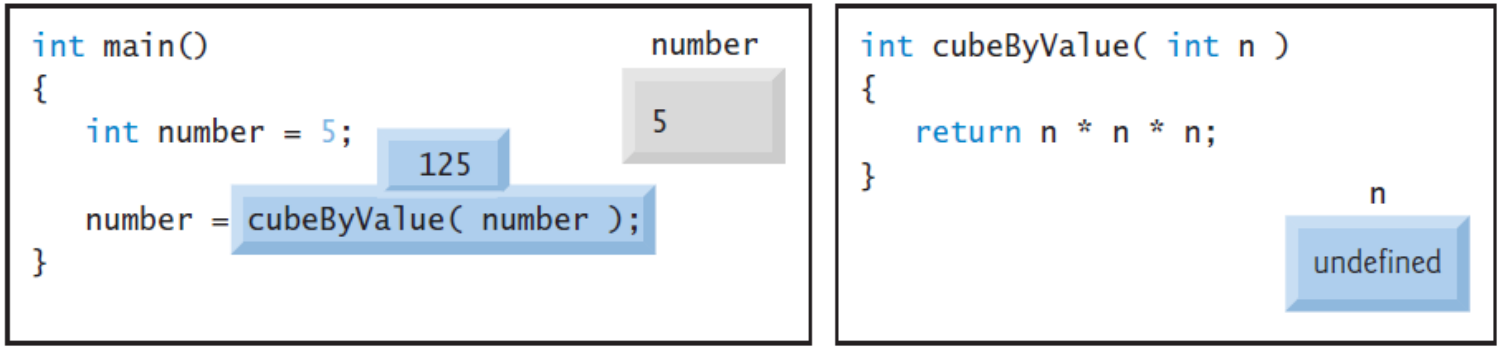
```
int cubeByValue( int n )
{
   return n * n * n;
}
```

125

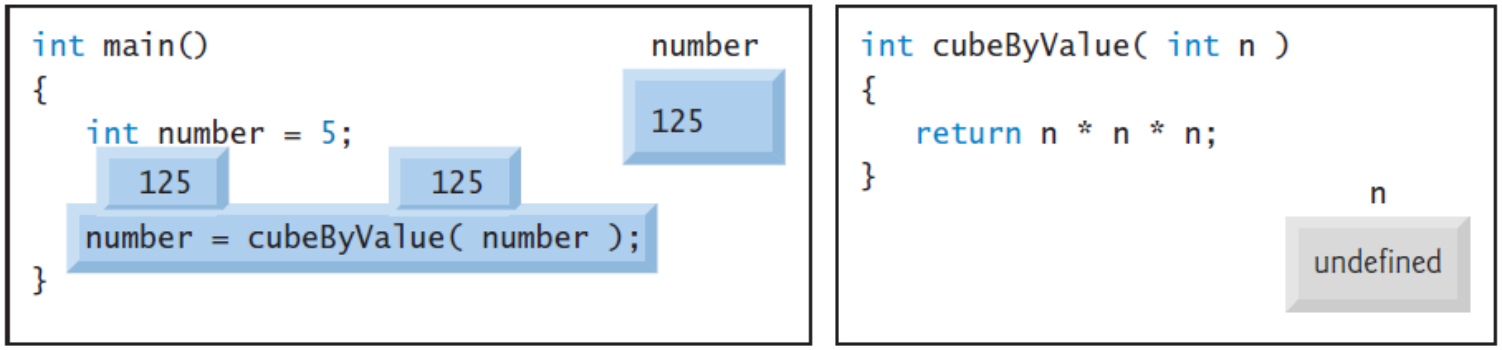
n: 5

Analiza grafike e qasjes përmes vlerës (Sh. 2)

Step 4: After `cubeByValue` returns to `main` and before assigning the result to `number`:

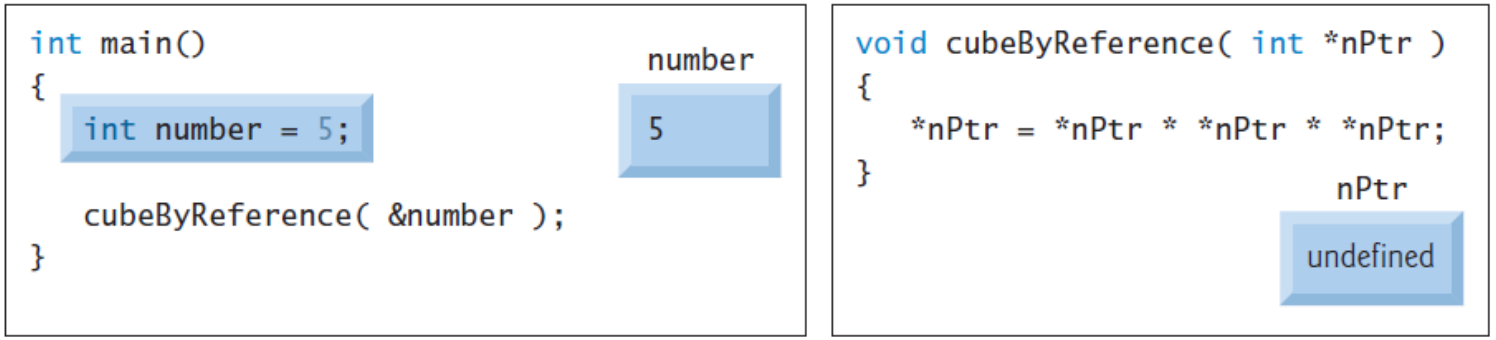


Step 5: After `main` completes the assignment to `number`:

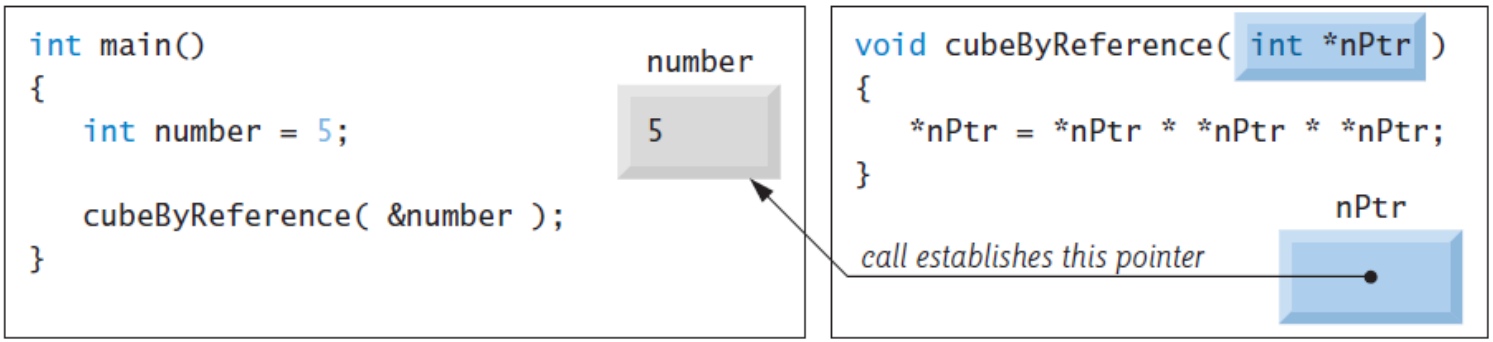


Analiza grafike e qasjes përmes referencës (Sh. 2)

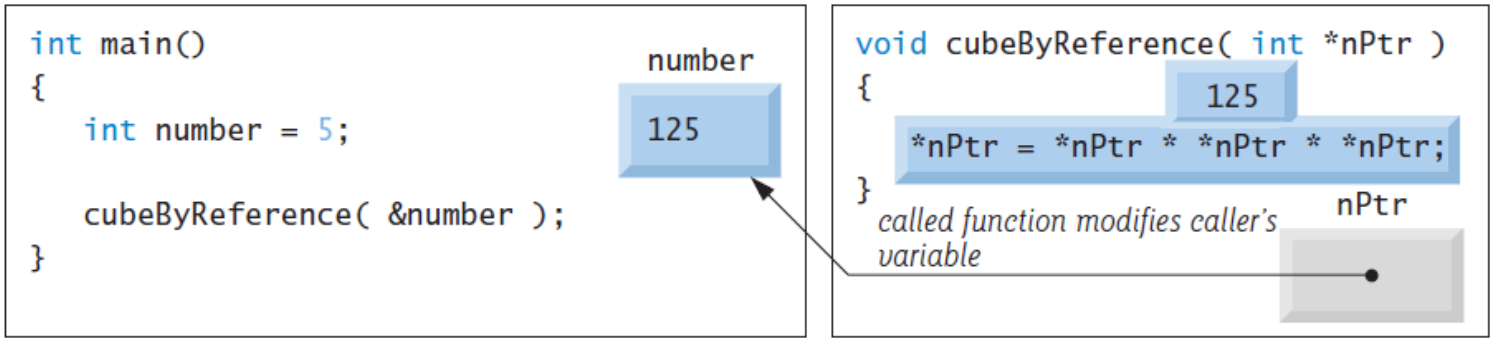
Step 1: Before `main` calls `cubeByReference`:



Step 2: After `cubeByReference` receives the call and before `*nPtr` is cubed:



Step 3: After `*nPtr` is cubed and before program control returns to `main`:



Fushat me ponterë

- Deklarimi i fushës

```
// c është fushë njëdimenzionale me 12 integer
int  c [12];
```

- Qasja ndaj elementeve të fushës
  - Përmes [ ]

- Inicializimi i elementeve të fushës

```
int  n[ 5 ] = { 50, 20, 30, 10, 40 };
```

Fushat me ponterë

- Declarimi i fushës
- Parametrat deklarohen në header:

```
int  sumElements (const int values[], const size_t
        numberOfElements )
```

- E njejtë është edhe nëse deklarohet si në vijim:

```
int sumElements( const int *values, const size_t
        numberOfElements )
```

Fushat me ponterë

- Fushat e këtilla të deklaruara nga shfrytëzuesi kanë disa kufizime:
  - Ata *nuk mund të krahasohen* përmes operatorëve të barazisë
  - Ata *nuk mund të marrin vlerat* nga një fushë tjetër
  - Atyre *nuk iu dihet madhësia*

- Nëse inkuadrohet **array** dhe **vector** atëherë mund të shfrytëzohen disa benificione plotësuese të fushave!

Fushat me ponterë

- Shfrytëzimi i fushave të këtilla nganjëherë gjen zbatim edhe në ca raste specifike
  - Programimi përmes linjës komanduese (**command-line arguments**)

    P.sh:

  - Në **Windows**
    - Komanda: **dir /p**
  - Në **Linux** ose **OS X**
    - Komanda: **ls -la**

const *në kuadër të pointerëve*

- Mundësitë e shfrytëzimit të opcionit **const**

- **Parimi i privilegjes më të ulët**:

  Gjithmonë funksioneve duhet dhënë qasje të mjaftueshme në të dhëna për egzekutime përkatëse, por jo më shumë se sa duhet!

Software Engineering Observation

If a value does not (or should not) change in the body of a function to which it's passed, the parameter should be declared `const`.

const *në kuadër të pointerëve*

- Egzistojnë katër mënyra për qasje të këtilla përmes pointerëve:

  1. Pointeri jokonstant në të dhëna jokonstante
  2. Pointeri jokonstant në të dhëna konstante
  3. Pointeri konstant në të dhëna jokonstante
  4. Pointeri konstant në të dhëna konstante

- Çdo kombinim ofron nivele të ndryshme të qasjes

1. Pointeri jokonstant në të dhëna jokonstante

- Qasja pa kufizime ofrohet në rastin: **Pointeri jokonstant në të dhëna jokonstante**
  - Të dhënat mund të modifikohen përmes pointerit, dhe gjithashtu edhe pointeri mund të modifikohet përr të treguar të dhëna të tjera

- Shprehja vijuese:

  int *countPtr

  Nuk shfrytëzon ocionin **const**

2. Pointeri jokonstant në të dhëna konstante

- Pointeri jokonstant në të dhëna konstante
  - Pointeri mund të modifikohet përr të treguar të dhëna të tjera, por e dhëna qe tregon ai pointer nuk mund të modifiohet përmes pointerit
- Çdo tentim për ndryshim të të dhënave të tilla rezulton me “compilation error”
- Shembull:

```
const int *countPtr;
```

2. Pointeri jokonstant në të dhëna konstante

- Shembull 4:

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

void f( const int * ); // prototype


int main()
{
    int y=0;
    f( &y );    // f attempts illegal modification
}


void f(const int *xPtr )
{
    *xPtr = 100;    // error: cannot modify a const object
};
```

```
error C3892: 'xPtr' : you cannot assign to a variable that is const
```

3. Pointeri konstant në të dhëna jokonstante

- Pointeri konstant në të dhëna jokonstante paraqet rastin kur pointeri gjithmonë I referohet lokacionit të njejtë, dhe të dhënat mund të ndryshohen përmes atij pointeri vetëm në atë lokacion
- Pointerët e deklaruar **const** duhet patjetër të deklarohen dhe inicializohen njëkohësisht

3. Pointeri konstant në të dhëna jokonstante

- Shembull 5:

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

int main()
{
    int x, y;

    // ptr is a constant pointer to an integer that can
    // be modified through ptr, but ptr always points to the
    // same memory location.
    int * const ptr = &x; // const pointer must be initialized

    *ptr = 7; // allowed: *ptr is not const

    ptr = &y; // error: ptr is const; cannot assign to it a new address

}
```

```
error C3892: 'ptr' : you cannot assign to a variable that is const
```

4. Pointeri konstant në të dhëna konstante

- Ky është rasti i privilegjeve minimale që sigurohet përmes **Pointerit konstant në të dhëna konstante**
  - Pointeri i tillë gjithmonë tegon lokacionin e njejtë memorues dhe të dhënat në atë lokacion nuk mund të ndryshohen

4. Pointeri konstant në të dhëna konstante

- Shembull 6:

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

int main()
{
    int x = 5, y;

    // ptr is a constant pointer to a constant integer.
    // ptr always points to the same location; the integer
    // at that location cannot be modified.

    const int *const ptr = &x;

    cout << *ptr << endl;

    *ptr = 7; // error: *ptr is const; cannot assign new value
    ptr = &y; // error: ptr is const; cannot assign new address

}
```

```
error C3892: 'ptr' : you cannot assign to a variable that is const
error C3892: 'ptr' : you cannot assign to a variable that is const
```

*Operatori **sizeof***

- Operatori **sizeof** përcakton numrin e bajtëve të rezervuar në kuadër të fushave ose tipeve tjera të të dhënave, variablave ose konstanteve gjatë kompajlimit të programeve

Operatori **sizeof**

- Shembull 7:

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

size_t getSize(double *); // prototype

int main()
{
    double array[ 20 ]; // 20 doubles; occupies 160 bytes on our system
    cout <<"The number of bytes in the array is " << sizeof( array );

    cout << "\nThe number of bytes returned by getSize is " <<getSize( array ) << endl;

    cin.get();  return 0;
}

size_t getSize( double *ptr )
{
return sizeof( ptr );
}
```

```
The number of bytes in the array is 160
The number of bytes returned by getSize is 4
```

Shprehjet dhe aritmetika me pointerë

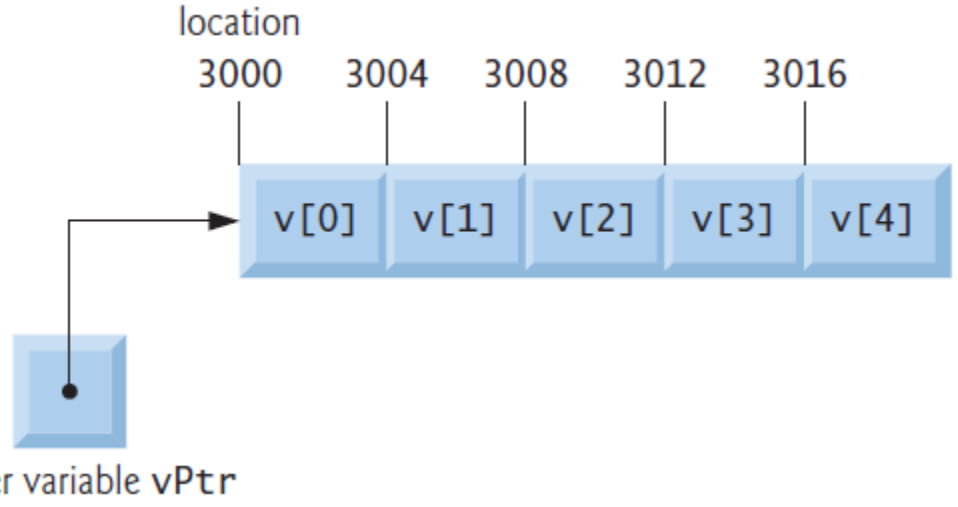
- Pointerët shfrytëzohen si operandë në shprehje aritmetikore, në shprehjet për ndarje të vlerave dhe në shprehjet e krahasimit

- C++ ofron **aritmetikën me pointerë**:
  - incrementi (++)
  - dekrementi (--)
  - një integer mund ti shtohet një pointeri (+ or +=)
  - një integer mund ti zbritet një pointeri (- or -=)
  - një pointer mund të zbritet nga një pointer tjetër i tipit të njejtë
    - *kjo vlenë në rastin kur pointerët u takojnë elementeve të të njejtës fushë*

Shprehjet dhe aritmetika me pointerë

- Assume that int **v[5]** has been declared and that its first element is at memory location **3000**
- Assume that pointer **vPtr** has been initialized to point to **v[0]** (i.e., the value of **vPtr** is **3000**)
- Variable **vPtr** can be initialized to point to **v** with either of the following statements (*for a machine with four-byte integers*):

```
int *vPtr = v;
int *vPtr = &v[ 0 ];
```



Shprehjet dhe aritmetika me pointerë

- Mbledhja dhe zbritja e intexherëve nga Pointerët
- Në aritmetikën konvencionale, 3000 + 2 rezulton në vlerën 3002
  - Rasti i këtillë kuptohet se nuk vlenë në aritmetikën me pointerë
  - Në rastin e pointerëve vlera do të rritet për 2 herë sa është madhësia e të dhënës që tregon pointeri

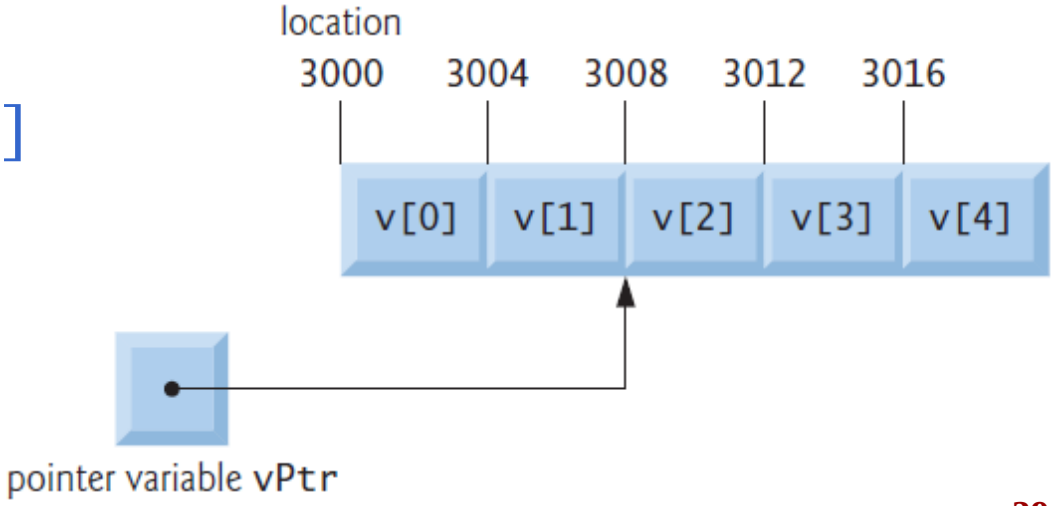
Shprehjet dhe aritmetika me pointerë

- Shembull:

  vPtr += 2;

  - Kjo shprehje do të rezultojë në **3008** (nga llogaritja 3000 + 2 * 4), nën supozimin se variabla **int** rruhet në 4 bajtë memorje
  - Nëse int ruhet në 8 bajtë memorje, atëherë llogaria do të ishte **3016** (3000 + 2 * 8)

- Në fushën **v**, **vPtr** do të tregon v[2]



Shprehjet dhe aritmetika me pointerë

- Zbritja e pointerëve
- Pointerët mund të zbriten mes veti, nëse iu referohen elementeve të të njejtës fushë
- Shembull:
  - Nëse **vPtr** përmban adresën **3000** dhe **v2Ptr** përmban adresën **3008**, shprehja

    x = v2Ptr - vPtr;

  - Do tia ndan **x** vlerën **vPtr - v2Ptr** - në këtë rast, **2**

Shprehjet dhe aritmetika me pointerë

- Ndarja e vlerave përmes pointerëve
- Një pointer mund ti ndahet një pointeri tjetër vetëm nëse janë të të njejtit tip

Shprehjet dhe aritmetika me pointerë

- Krahasimi i pointerëve

- Pointerët mund të krahasohen përmes operatorit të barazisë dhe operatorëve relacionalë
- Krahasimi përmes operatorëve relacional bëhet në rastet kur pointerët i takojnë të njejtës fushë
- Krahasimi i pointerëve krahason adresat e ruajtura në pointerët përkatës
- Rast më i shpeshtë është kur pointeri krahasohet a është **nullptr**, **0** ose **NULL**

Relacioni mes Pointerëve dhe fushave

- Pointerët mund të shfrytëzohen për disa operacione në kuadër të fushave të deklaruara nga shfrytëzuesi
- Shembull:

```
// create 5-element int array b; b is a const pointer
int b[ 5 ];

// create int pointer bPtr, which isn't a const pointer
int *bPtr;

// assign address of built-in array b to bPtr
bPtr = b;

// also assigns address of built-in array b to bPtr
bPtr = &b[ 0 ];
```

Relacioni mes Pointerëve dhe fushave

- Shembull:

```
// Built-in array element b[ 3 ] can alternatively be
// referenced with the pointer expression
 *( bPtr + 3 )

// Just as the built-in array element can be referenced
// with a pointer expression
&b[ 3 ]

// The address can be written with the pointer expression
bPtr + 3

// Next expression also refers to the element b[ 3 ]
*( b + 3 )
```

Relacioni mes Pointerëve dhe fushave

- Shembull 8:

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

int main()
{
    int b[] = { 10, 20, 30, 40 }; // create 4-element array b
    int *bPtr = b; // set bPtr to point to array b

    // output array b using array subscript notation
    cout << "Array b printed with:\n\nArray subscript notation\n";
    for ( int i = 0; i < 4; ++i )
        cout << "b[" << i << "] = " << b[ i ]<< '\n';

    // output array b using the array name and pointer/offset notation
    cout << "\nPointer/offset notation where "
        << "the pointer is the array name\n";
    for ( int offset1 = 0; offset1 < 4; ++offset1 )
        cout << "*(b + " << offset1 << ") = " << *( b + offset1 )<< '\n';

    // output array b using bPtr and array subscript notation
    cout << "\nPointer subscript notation\n";
    for ( int j = 0; j < 4; ++j )
        cout << "bPtr[" << j << "] = " <<bPtr[ j ] << '\n';
    cout << "\nPointer/offset notation\n";

    // output array b using bPtr and pointer/offset notation
    for ( int offset2 = 0; offset2 < 4; ++offset2 )
        cout << "*(bPtr + " << offset2 << ") = "
        <<*( bPtr + offset2 )<< '\n';

    cin.get();  return 0;
}
```

```
Array subscript notation
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40

Pointer/offset notation where
the pointer is the array name
*(b + 0) = 10
*(b + 1) = 20
*(b + 2) = 30
*(b + 3) = 40

Pointer subscript notation
bPtr[0] = 10
bPtr[1] = 20
bPtr[2] = 30
bPtr[3] = 40

Pointer/offset notation
*(bPtr + 0) = 10
*(bPtr + 1) = 20
*(bPtr + 2) = 30
*(bPtr + 3) = 40
```

- Pyetje eventuale?!

mentor.hamiti@universitetiaab.com