Object Oriented Programming

Shaobai Kan

Pointers

Outline

Outline

- Pointer

Outline

- Pointer
- Declare and initialize a pointer variable

Outline

- Pointer
- Declare and initialize a pointer variable
- Pointer operators

Pointer

Pointer variables

Definition. A pointer is a variable that holds a memory address.

Pointer variables

Definition. A pointer is a variable that holds a memory address.

Recall. A variable is a location in your computer's memory in which you can store a value and from which you can retrieve that value.

Pointer variables

Definition. A pointer is a variable that holds a memory address.

Recall. A variable is a location in your computer's memory in which you can store a value and from which you can retrieve that value.

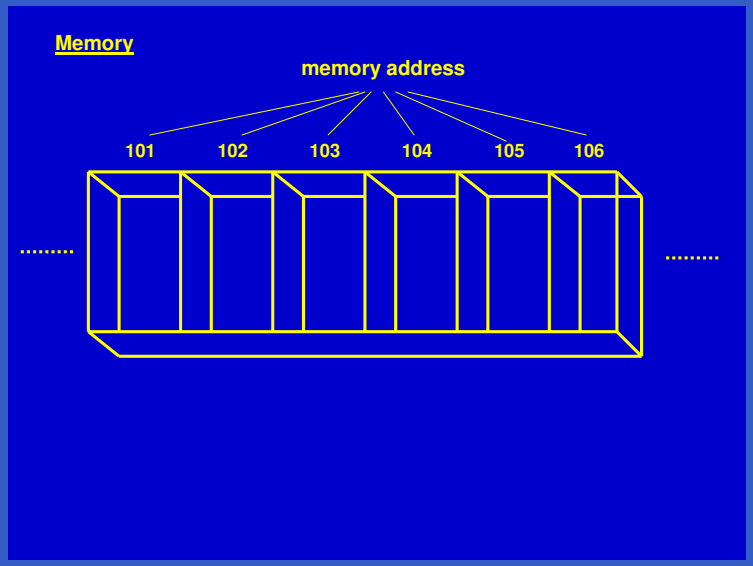
- An integer variable holds an integer number.
- A character variable holds a letter
- A pointer is a variable that holds a memory address.

Recall: memory

Recall. Computer memory is where variable values are stored. By convention, computer memory is divided into sequentially numbered memory locations. Each of these locations is a memory address.

Recall: memory

Recall. Computer memory is where variable values are stored. By convention, computer memory is divided into sequentially numbered memory locations. Each of these locations is a memory address.



Q: Why do we need pointers?

Fact. Pointer is one of the most powerful tools available to a C++ programmer.

Q: Why do we need pointers?

Fact. Pointer is one of the most powerful tools available to a C++ programmer.

- Pointers enable pass-by-reference

Q: Why do we need pointers?

Fact. Pointer is one of the most powerful tools available to a C++ programmer.

- Pointers enable pass-by-reference
- Pointers can be used to create and manipulate dynamic data structures (i.e. data structures that can grow and shrink).
  e.g. linked lists, queues, stacks, trees.

Q: Why do we need pointers?

Fact. Pointer is one of the most powerful tools available to a C++ programmer.

- Pointers enable pass-by-reference
- Pointers can be used to create and manipulate dynamic data structures (i.e. data structures that can grow and shrink).
  e.g. linked lists, queues, stacks, trees.
- C-style, pointer-based strings are widely used in C/C++ systems.

Memory address of a variable

Addresses of variables

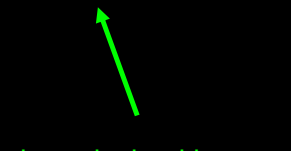
```
# include <iostream>
using namespace std;


int main( )
{
        unsigned short shortVar = 5 ;
        unsigned long longVar = 65535 ;

        cout << "shortVar:\t" << shortVar;
        cout << "\tAddress of shortVar:\t" << &shortVar << endl;

        cout << "longVar:\t" << longVar;
        cout << "\tAddress of longVar:\t" << &longVar << endl;

        return 0 ;
 }
```

address operator

address operator

Output

```
shortVar:       5       Address of shortVar:   0013FF60

longVar:        65535   Address of longVar:    0013FF54
```



Declare and initialize a pointer variable

Declare variables *v.s.* declare pointers

Recall. Declare regular built-in type variables

e.g.

```
int a;
double b;
char c;
```

Declare variables *v.s.* declare pointers

Recall. Declare regular built-in type variables

e.g.

```
int a;
double b;
char c;
```

Question. How to decare a pointer?

Declare variables *v.s.* declare pointers

Recall. Declare regular built-in type variables

e.g.

```
int a;
double b;
char c;
```

Question. How to decare a pointer?

e.g.

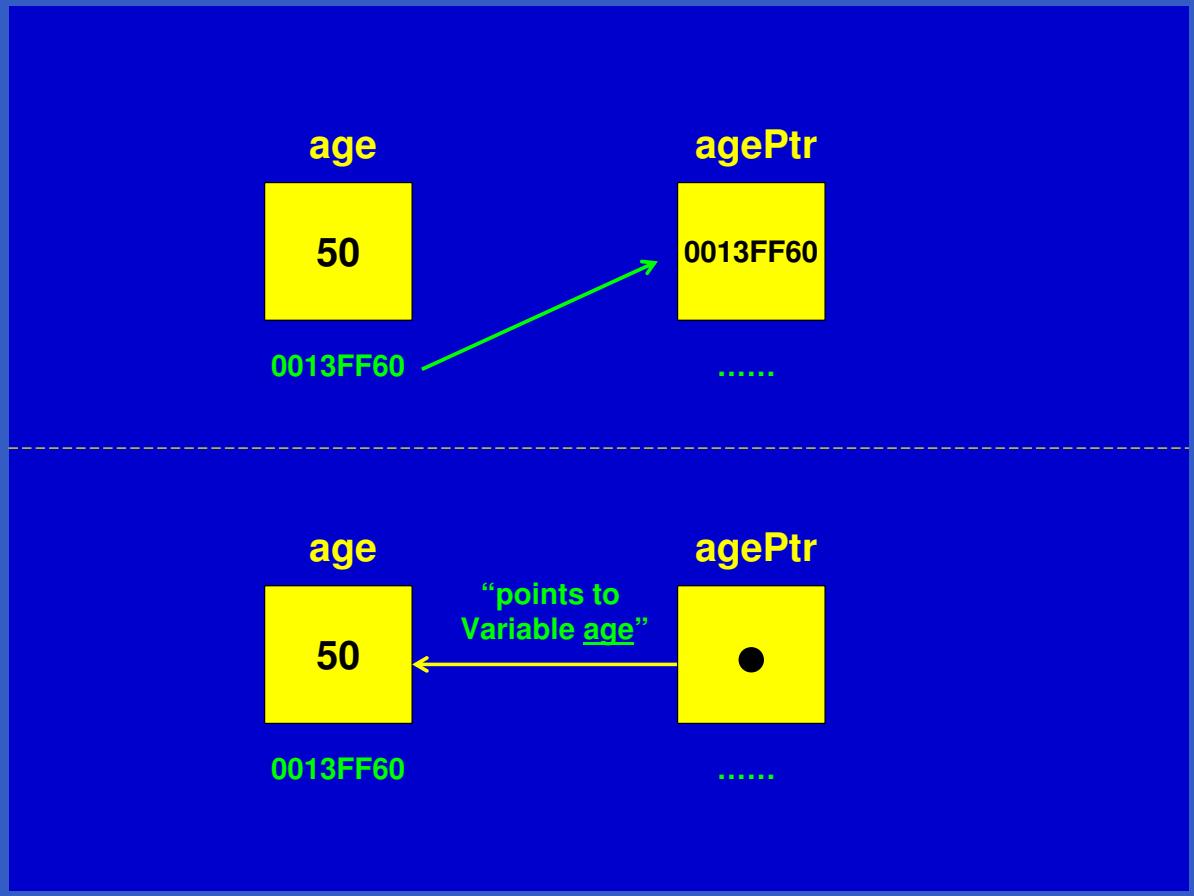
```
int *aPtr;
double *bPtr;
```

Declare and initialize a pointer I

Declaration & initialization of pointers

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

int main( )
{
        int age = 50 ;
        int * agePtr ;              // decalre a pointer"agePtr"

        agePtr = &age;          // put age's address in agePtr

        cout << "age:\t" << age;      // directly references a value
        cout << endl;
        cout << "agePtr:\t" << agePtr << endl;

        return 0 ;
 }
```

Declare and initialize a pointer I



Output: Declare and initialize a pointer I

```
age:     50

agePtr:     0013FF60
```

Declare and initialize a pointer II

Declaration & initialization of pointers II

```
# include <iostream>

using namespace std;
int main( )
{
        double y = 5.0 ;
        double * yPtr = &y ;


         cout << "y:\t" << y << endl;
         cout << "yPtr:\t" << yPtr << endl;


        return 0 ;
 }
```

Output: Declare and initialize a pointer II

```
y:       5
yPtr:    0013FF5C
```

Initialize a pointer

Fact. Pointers should be initialized either when they are declared or in an assignment.

- int *countPtr = &a;
  —— points to another variable a
- int *countPtr = 0;
  —— points to nothing
- int *countPtr = NULL;
  —— points to nothing

Pointer operators

Address operator *v.s.* dereferencing operator

address operator (&). is a unary operator that obtains the memory address of its operand.

e.g.

```
int *pX = & x;
```

Address operator *v.s.* dereferencing operator

address operator (&). is a unary operator that obtains the memory address of its operand.

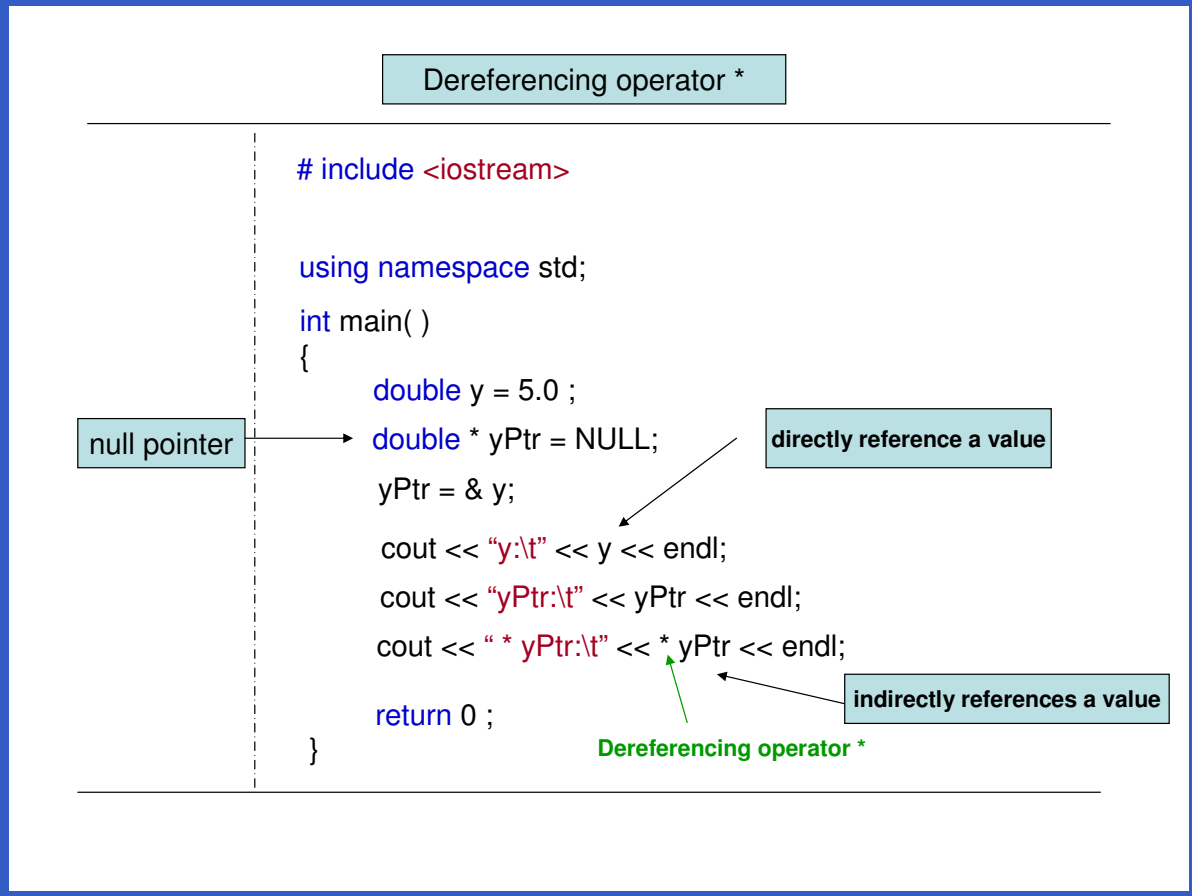
e.g.

```
int *pX = & x;
```

dereferencing operator (*). returns a synonym (i.e. an alias or a nickname) for the object to which its pointer operand points.

equivalence: *pX $\Leftrightarrow$ x

Dereferencing operator *



Output: dereferencing operator *

```
y:        5
yPtr:     0013FF5C
*yPtr:    5
```

Exercise

Exercise: dereferencing operator

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

int main( )

{
        int myAge;
       int *myAgePtr = &myAge;

        myAge = 5;
        cout << “myAge:” << myAge << “\t”;
        cout << “*myAgePtr:” << *myAgePtr << “\n”;

       *myAgePtr = 7;
        cout << “myAge:” << myAge << “\t”;
        cout << “*myAgePtr:” << *myAgePtr << “\n”;
        return 0 ;
 }
//Note:     *myAgePtr  ⟺  myAge
```

Output: exercise

```
maAge:5         *pAge:5
myAge:7         *pAge:7
```

Precedence and associativity of operators

Order

Operators	Associativity	Type
( ) [ ]	left to right	highest
++ -- static_cast<>()	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	logic AND
||	left to right	logic OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment
,	left to right	comma

Example: address (&) *v.s.* dereferencing (*)

Address operator *v.s.* Dereferencing operator

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

int main( )
{
     int a ;
     int * aptr

     a = 7;
     aptr = & a;

     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 << endl;
     cout << "&* aPtr: " << &* aPtr << "\t*& aPtr" << *& aPtr << endl;
     return 0 ;
}
 // *   and   &  are   inverse of each other
```

Output: address (&) *v.s.* dereferencing (*)

```
The address of a is 0012F580
The value of aPtr is 0012F580

The value of a is 7
The value of *aPtr is 7
 &* aPtr = 0012F580              *& aPtr = 0012F580
```

Homework:

- Read Sec. 7.1-7.3