Pointer variables

Lecture 9

Variables

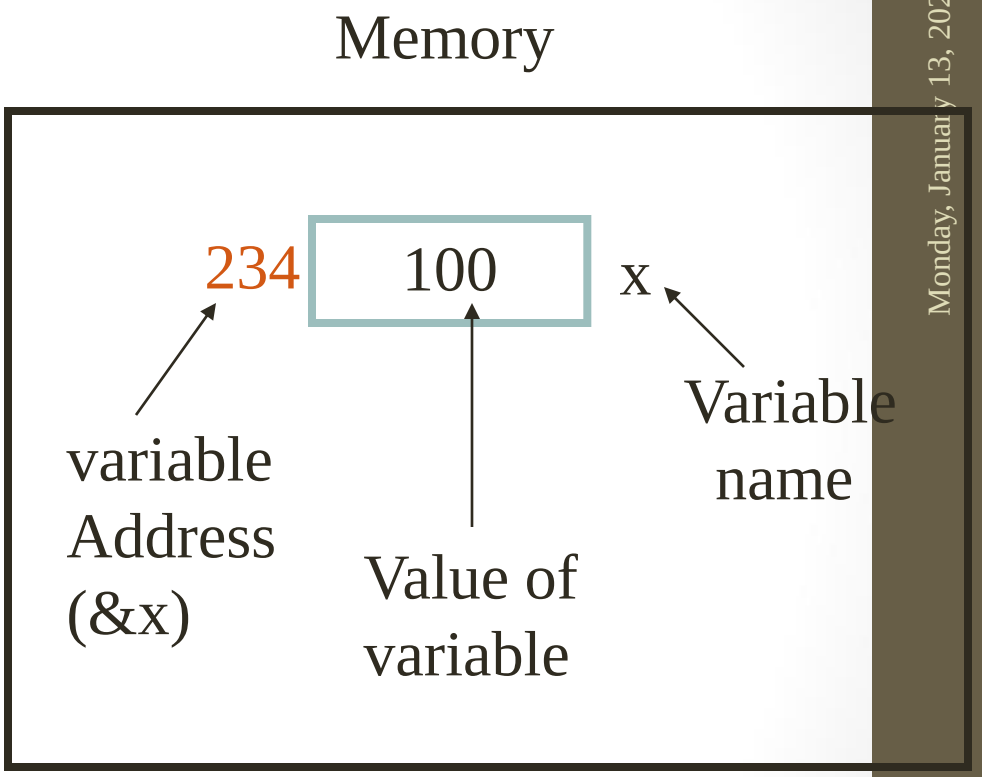
- variable in a program is something with a name, the value of which can vary

  int *var*;

- compiler and linker assigns a specific block of memory within the computer to hold the value of that variable

Variable

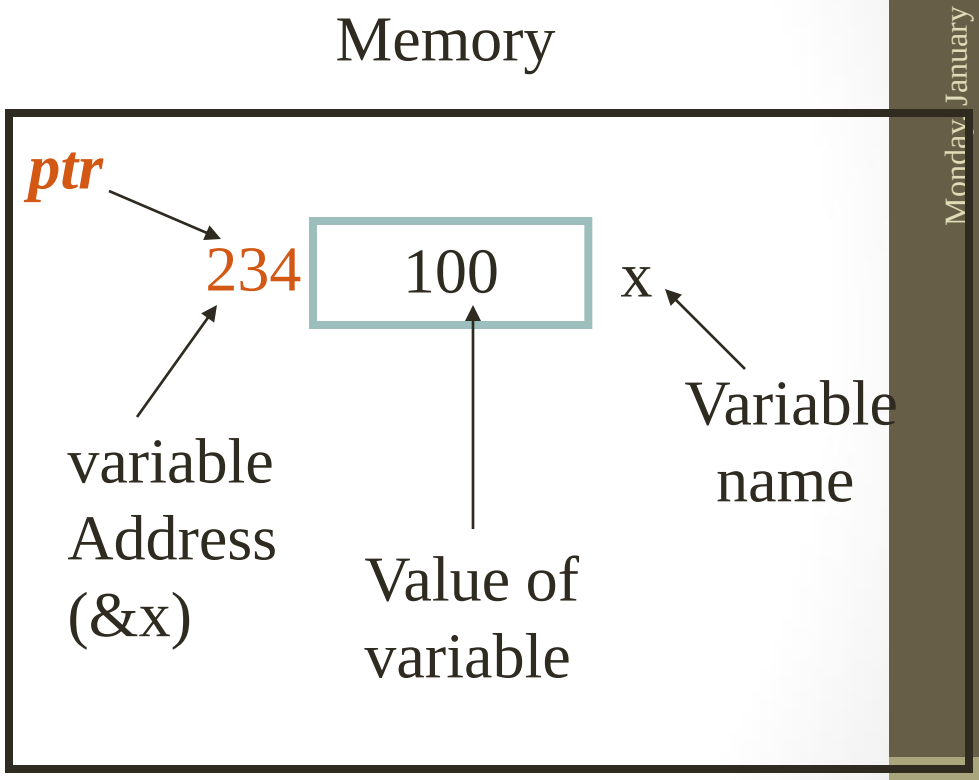
```
int x=100;
cout<<x;
cout<<(&x);
```



Pointers

- variable that represents the location (rather than the value) of a

```
int x;
int *ptr;
ptr=&x;
```



How to access the value of variable through pointer?

```
void main()
{  int x=10;
   int *ptr=&x;
   cout<<ptr;
   cout<<(*ptr);
}
```

Pointer to objects

```
class A
{  public:
     int x;
     int y;
   A() { x=10;
     y=20;}
   void display()
   { cout<<x<<y;}
};
```

```
void main()
{ A a1;
  A *ptr;
 ptr=&a1;
 cout<<ptr->x;
 cout<<ptr->y;
 ptr->display();
}
```

Dynamic memory management

- C++ enables programmers to control the allocation and deallocation of memory in a program for any built-in or user-defined type.
- *Dynamic allocation* is the creating of an object while the program is running, using the new operator.
- This is called dynamic memory management which is accomplished using ***new*** and ***delete*** operators.

new operator

- The new operator allows dynamic allocation of memory
- The object or variable is created in the free store (heap) – a region of memory assigned to each program for storing objects created at execution time

Example

```
class A
{  public:
     int x;
     int y;
   A() { x=10;
     y=20;}
   void display()
   { cout<<x<<y;}
};
```

```
void main()
{   A *ptr;
 ptr=new A;
 cout<<ptr->x;
 cout<<ptr->y;
 ptr->display();
}
```

Basic data types

- C++ allows us to provide an initializer for a newly created fundamental-type variable

```
int *p=new int(13);
double *pi=new double(3.14)
```

Delete operator

- To destroy a dynamically allocated object and free the space for the object, use delete operator

- The delete operator erases the object from the heap.

Example

```
class A
{  public:
     int x;
     int y;
   A() { x=10;
     y=20;}
   void display()
   { cout<<x<<y;}
};
```

```
void main()
{   A *ptr;
 ptr=new A;
 cout<<ptr->x;
 cout<<ptr->y;
 ptr->display();
 delete ptr;
}
```

Memory leaks

A *memory leak* is an error condition that is created when an object is left on the heap with no pointer variable containing its address. This might happen if the object's pointer goes out of scope:

```
void MySub()
{
  Student * pS = new Student;

  // use the Student for a while...

}  // pS goes out of scope
```

(the Student's still left on the heap)

Dangling pointers

A *dangling pointer* is created when you delete its storage and then try to use the pointer. It no longer points to valid storage and may corrupt the program's data.

```
double * pD = new double;
*pD = 3.523;
.
.
delete pD;    // pD is dangling...
.
.
*pD = 4.2;    // error!
```

Avoid dangling pointers

To avoid using a dangling pointer, assign NULL to a pointer immediately after it is deleted. check for NULL before using the pointer

```
delete pD;
pD = NULL;
.
.
if( pD != NULL )        // check it first...
  *pD = 4.2;
```

Assignment

- What do you mean by Dangling of Pointers.