Lecture 18

Devendra Ghate

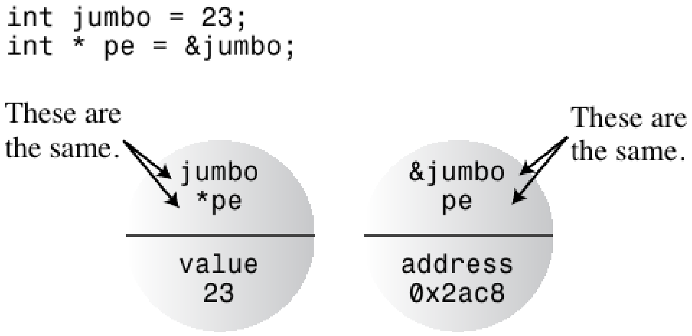
05/01/2021

"If you think it's simple, then you have misunderstood the pointers!" - Anon

Pointers

- `&` (referencing) and $*$ (dereferencing) operator (./35-pointers.cpp)
- Pointers are variables that store the addresses of various things (mostly other variables).
- What do we see when we print the pointers?
  - We see the address in the hexadecimal number system
- Is there a special datatype for pointers?
  - Yes and no. Datatype of pointer is defined by the data it is pointing to
- Why is the size of pointers 8 Bytes? What is the maximum RAM that it can handle?

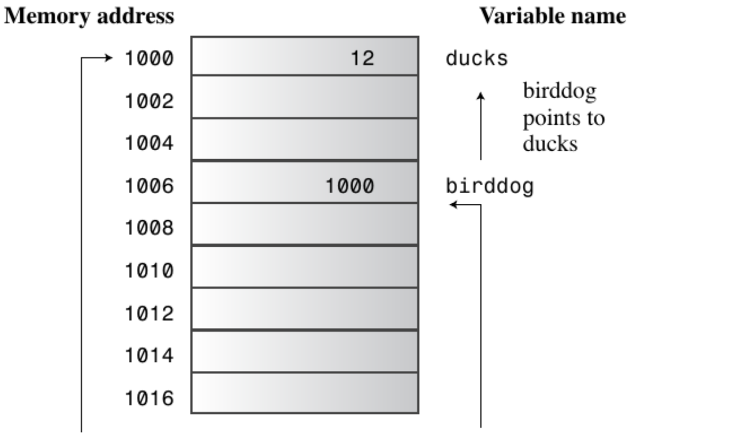
Pointers



```
int jumbo = 23;
int * pe = &jumbo;
```

Figure 4.8 Two sides of a coin.

Pointers



`int ducks = 12;`

creates ducks variable, stores the value 12 in the variable

`int *birddog = &ducks;`

creates `birddog` variable, stores the address of `ducks` in the variable

Pointers

(./37-pointerOperations.cpp)

```
int x = 10;
int *px;
px = &x;
x = x + 1;
*px = *px + 1;
```

Pointer to a pointer

(./38-pointerToPointer.cpp)

```
int x=10;
int *px;
int **ppx;
px = &x;
ppx = &px;
cout << x << endl;
cout << *px << endl;
cout << **ppx << endl;
```

Dynamic memory allocation

- Usually using `new`
  - `int *pa = new int;`
- Memory location to which `pa` points does not have a variable name associated with it
- It can only be accessed via `*pa`

Lets try this! (./39-new.cpp)

- `int *a; a = new int[10]`

Dynamic memory allocation

- Reverse process is `delete`

```
int * ps = new int;
delete ps;
delete ps; //Not allowed
int j = 5;
int * pj = &j;
delete pj; //Not allowed
```

- Its a good practice to free-up unneeded memory

Pointers to array

(./36-arrayAsPointer.cpp)

```
int a[] = {1, 2, 3};
int *pa;
pa = &a[0];
cout << *pa + 10 << endl
cout << a[0] << endl;
cout << *(pa + 1) << endl;
*(pa + 1) = 20;
cout << a[1] << endl;
```

- What about a 2D array?