

Pointers



- **Another kind of data type**
- **Similar to a reference**
 - Object that is an “alias” for another
- **Unlike reference**
 - Aliasing must be explicit
 - Can change the “referent”
 - What the “reference” refers to

Reference Review

- **Definition uses “&” suffix**
 - In function argument, function return value, ordinary data object
- **Automatic aliasing**
 - Action on reference becomes action on referent (“referred to” object)

Reference Argument Example

```

int a = 3;
func (a);
cout << a; // 6
...
void func (int& ref_a)
{
  cout << ref_a; // 3
  ref_a = 6;
}

```

Initialize reference argument (points to (a) in func call)

Access through reference (points to ref_a in cout statement)

Modify through reference (points to ref_a in assignment statement)

Reference Object Example

```

int a = 3;
int& ref_a = a;
cout << ref_a; // 3
ref_a = 6;
cout << a; // 6

```

Initialize reference object

Access through reference

Modify through reference

Pointer Operation Syntax

- **Pointer declaration**
 - Suffix "*" after type
- **Pointer dereference**
 - Prefix "*" operator
- **Address-of operator**
 - Prefix "&" applied to object
 - Returns a pointer pointing to object

Pointer Object Example

```

int a = 3;
int* ptr_a = &a;
cout << *ptr_a; // 3
*ptr_a = 6;
cout << a; // 6

```

Initialize pointer object with address

Access through pointer

Modify through pointer

Pointer Argument Example

```

int a = 3;
func (&a);
cout << a; // 6
...
void func (int* ptr_a)
{
  cout << *ptr_a; // 3
  *ptr_a = 6;
}

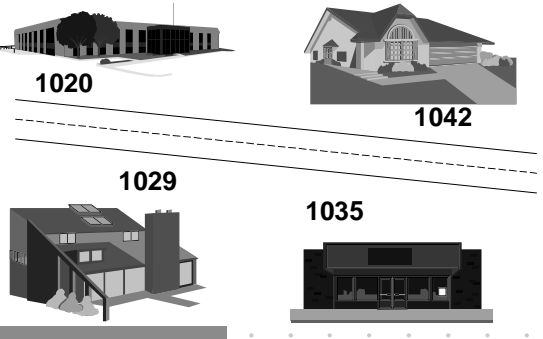
```

Initialize pointer argument

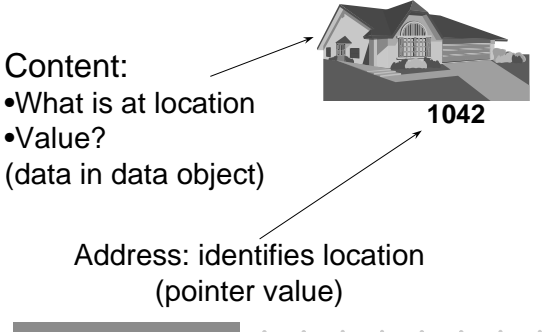
Access through pointer

Modify through pointer

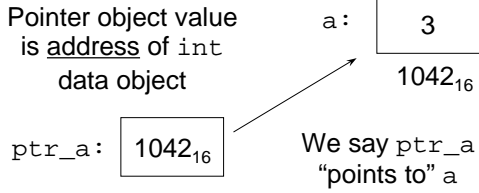
Neighborhood Organization



Neighborhood: Addressing

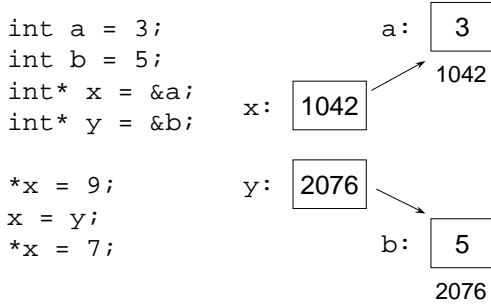


Pointers & Addresses



We "dereference" (with "*" operator) a pointer value to access the data object

Pointer Assignment



Pointer Declaration Caution

- `int a;` ← Define an int object
- `int* w;`
`int *x;` ← Define pointers to int objects
- `int *y, z;` ← y is pointer to int, but z is int!
- `int *y, *z;` ← y and z are both pointers to int

Pointer Declaration Syntax

• Unfortunately

- “*” binds to object, not type
- “int* x,y;” is “int *x; int y;”
 - See text, page 611

• Solutions

- One object per declaration (preferred!)
- Typedef: “typedef int* intptr;”

Pointers to Class Types

• Similar to built-in types

- Use operators “&” and “*”

• Accessing members

- Date today (1,1,2001);
- Date* datep = &today;
- int dd = (*datep).GetDay();

Member Access via Pointers

```

int dy;
Date d (1,1,2000);
Date* dptr = &d;
dy = (*dptr).GetDay();
dy = dptr->GetDay();

```

Basic member access
(works, but messy)

Alternate notation
(equivalent)

Rvalues and Lvalues Text, page 608

• **Lvalue**

- Can be evaluated and modified
- Legal to appear on left side of “=”
 - Also can appear on right side

• **Rvalue**

- Can be evaluated, not modified
- Legal to appear on right side of “=”

• **What are some examples?**

Other Pointer Details Text, page 609

• **Pointer type compatibility**

- Pointer type depends on referent type
- Can't assign “char*” to “double*”

• **Null pointer**

- Pointer may point to “nothing”
 - Unlike reference!
- Get null pointer by assigning “zero”
 - Sometimes use “NULL” (macro)

Constant Pointers

