

Procedures

- **Have seen regular functions**
 - Accept arguments (maybe none)
 - Return a value
- **Also have another kind**
 - Accept arguments (maybe none)
 - Do not return a value

Procedure Example

```

void PrintNum (int number)
{
  cout << "The number is ";
  cout << number << endl;
}

```

void means no returned value

Does not return a value (no return statement)

Could have a return statement with no return value (?)

Procedure Call Example

```

void PrintNum (int number);
void main (int number)
{
  int x = -35;
  PrintNum (45);
  PrintNum (x);
}

```

Prototype

No return value for use in expression

Procedure called for side effect, not for return value.

Pass-by-Value Arguments

- **“Normal” arguments** (all so far)
- **Formal argument**
 - “Local” data object inside function
 - Receives copy of actual argument
 - Expression, data object, etc.
 - Changes do not affect actual argument

Pass-by-Value Example

```

void main ()
{
  int num = 2;
  PrintNum (num);
  cout << num;
}
void PrintNum (int number)
{
  number = 3;
  cout << number << endl;
}

```

Actual argument (points to `num`)

Formal argument (points to `number`)

What gets printed?

Pass-by-Reference Arguments

- **Formal argument “refers”**
 - To actual argument
- **Acts as an alias**
 - Changes to formal argument affect actual argument
 - Immediately

Pass-by-Reference Example

```

void main ()
{
  int num = 2;
  PrintNum (num);
  cout << num;
}
void PrintNum ((int&)number)
{
  number = 3;
  cout << number << endl;
}

```

Actual argument

Formal argument

What gets printed this time?

Another Value Return Method

```

void main ()
{
  int num = 0;
  GetNum (num);
  cout << num;
}
void GetNum (int& number)
{
  cout << "Enter number: ";
  cin >> number;
}

```

Most useful when there is more than one value to return to caller.

Is This Legal?

```

void main ()
{
  GetNum (3);
}
void SetNum (int& number)
{
  number = 5;
}

```

number refers to the literal "3"

The literal cannot be an lvalue, so this is illegal.

⋮

What About This?

```
void main ()  
{  
  PrintNum (3);  
}  
void PrintNum (const int& number)  
{  
  cout << number;  
}
```

Again, number refers
to the literal "3"

Better than
pass-by-value?

We promise not to use the
reference to modify, so OK.

⋮
