

# HIGH-LEVEL PROGRAMMING 2

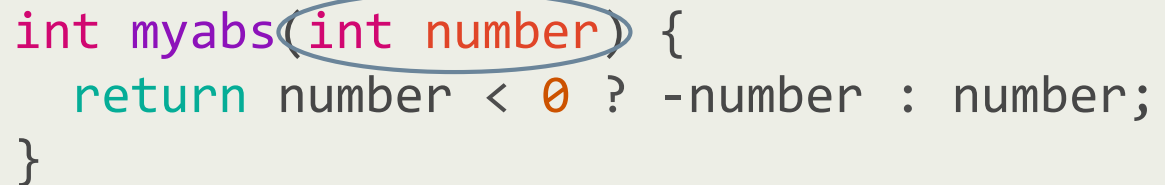
Default Function Parameters by Prasanna Ghali

# Functions: Parameters and Arguments

2

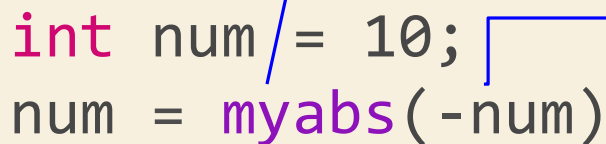
this variable is called *formal parameter* or just *parameter*

```
int myabs(int number) {  
    return number < 0 ? -number : number;  
}
```



client calls function *myabs* using function call operator *()*

```
int num = 10;  
num = myabs(-num)
```



this expression is called *function argument*

- 1) At runtime, expression (or argument) *-num* is evaluated
- 2) Result of evaluation is used to initialize parameter *number*
- 3) Changes made to parameter *number* are localized to function *myabs*
- 4) Function *myabs* terminates by returning value of type *int*
- 5) When function *myabs* terminates, variable *number* ceases to exist

# Idea: Default Parameters

3

- Some functions are called with one or more arguments having same values in most, but not all, calls
- In such cases, function can be declared with corresponding parameters having default values

# Idea: Default Parameters

4

- Consider definition of function `incr` in file *misc.cpp*

```
namespace misc_stuff {  
  
int incr(int value, int amount) {  
    return value+amount;  
}  
  
}
```

# Idea: Default Parameters

5

- Consider declaration of function `incr` in file *misc.hpp*

```
#ifndef MISC_HPP
#define MISC_HPP
namespace misc_stuff {

// function returns value+amount
int incr(int value, int amount);

}
#endif
```

# Idea: Default Parameters

6

- Consider client making calls to declaration of function `incr` in file `main.cpp`
- Since clients are often making calls to function `incr` with 2<sup>nd</sup> argument having value `1`, they'd prefer to simplify calls to `incr`

```
#include "misc.hpp"

int main() {
    int x{1}, y{2}, z{3};
    // some calls to function incr
    x = misc_stuff::incr(x, 1);
    y = misc_stuff::incr(y, 2);
    z = misc_stuff::incr(z, 1);

    // other stuff ...

    // more calls to incr ...
    x = misc_stuff::incr(x+y, 1);
    y = misc_stuff::incr(x*z, 1);
    // ...
}
```

# Idea: Default Parameters

7

- Designer of function `incr` would update function declaration in `misc.hpp` so that parameter `amount` will have default value `1`

```
#ifndef MISC_HPP
#define MISC_HPP
namespace misc_stuff {

    // function returns value+amount
    int incr(int value, int amount = 1);

}
#endif
```

# Idea: Default Parameters

8

- That's it ... no changes are required to definition of function `incr`

```
namespace misc_stuff {  
  
int incr(int value, int amount) {  
    return value+amount;  
}  
  
}
```



# Idea: Default Parameters

9

- Now clients can simplify their calls to function `incr` in file `main.cpp`
- When 2<sup>nd</sup> argument is missing in calls to `incr`, compiler will specify default value of `1` for that argument

```
#include "misc.hpp"

int main() {
    int x{1}, y{2}, z{3};
    // some calls to function incr
    x = misc_stuff::incr(x);
    y = misc_stuff::incr(y, 2);
    z = misc_stuff::incr(z);

    // other stuff ...

    // more calls to incr ...
    x = misc_stuff::incr(x+y);
    y = misc_stuff::incr(x*z);
    // ...
}
```

# Multiple Default Parameters

10

- ❑ You can have multiple default parameters; but they must *default right to left*
- ❑ Design your functions so that parameters liable to change the least are ordered from right to left

# Multiple Default Parameters

11

- You can have multiple default parameters; but they must *default right to left*

```
// in misc.hpp
namespace misc_stuff {
    void bar(int a, int b = 8, int c = 10);
}

// in misc.cpp
void bar(int a, int b, int c) {
    // some statements here ...
}

// in main.cpp
misc_stuff::bar(1);           // bar(1, 8, 10)
// only trailing args can be defaulted and left out in a call
misc_stuff::bar(1, , 9);     // error: missing argument
misc_stuff::bar(1, 2);       // bar(1, 2, 10)
misc_stuff::bar(1, 2, 9);     // bar(1, 2, 9)
```

# Multiple Default Parameters

12

- ❑ You can't break the *default right to left* rule in declarations too ...

```
// in misc.hpp
namespace misc_stuff {

// both function declarations are illegal!!!
void qux(int a, int b = 8, int c);
void quux(int a = 5, int b, int c = 10);

}
```

# Extending Function Parameter List

13

- Default parameters can be used to extend parameter list of existing functions without requiring any change in function calls already used in program
- See *circle.hpp*, *circle.cpp*, *circle-driver.cpp* for an implementation of this technique

# Handout

14

- See handout for more examples ...