Optional parameters, function overloading

- in this session we'll look at expanding the ways we can declare and call functions in two ways: optional parameters and function overloading
- it is possible to provided default values for a parameter, giving the caller the option of call the function with or without passing that specific parameter
- it is also possible to declare multiple different versions of a function with the same name, as long as the parameter lists are “different enough” for the compiler to always identify the correct one to call

Optional parameters

- in the declaration of a function we can assign a default value for one or more parameters, e.g.

```
void myfunction(int x, char y = 'a', float z = 1.3)
```

- this must be done in the prototype when following our course code standards
- the caller can omit one or more of the parameters, in which case the defaults are used, e.g.

```
myfunction(10); // uses 'a' for y, 1.3 for z
myfunction(10, 'q'); // uses 1.3 for z
```

Order of parameter assignment

- the parameters passed by the caller are assigned to the formal parameters in left-to-right order, then any remaining parameters use their default values

```
void myfunc(int x=1, int y=2, string z=”boo”);
```

- myfunc(3); // always gets assigned to x
- this means you cannot choose to use a default value for a “middle” parameter and pass a value for a later one, e.g. the following does not work:

```
myfunc(0, “hello); // tries to assign “hello” to y
```

Overloading functions

- we can declare multiple functions with the same name, as long as the number/types of parameters are completely unambigous, e.g. suppose we want functions to print a date, but want multiple ways to pass the date:

```
void printDate(string date);
void printDate(int yy, int mm, int dd);
void printDate(string weekday, string month, string day);
```

- when the compiler sees a function call to printDate, it looks at the number/types of parameters to see which version to call

Ambiguity in overloads

- when declaring the various versions of the function, it must not be possible to create an ambiguous call, e.g.

```
void f1(int x);
void f1(long y);
...
f1(3);  // could be a valid call to either version
```

- compiler decides which to use based on params only

```
void f1(int x);                    // later called with
int f1(long y);                    int i = f1(3);
                                   // gives error, f1 void
```

Optional params and ambiguity

- when functions include optional parameters then you must also account for any possible combination of passed parameters when considering ambiguity

```
void f1(int x);
int f1(int y, string z=”boo”);
f1(10); // could mean either one
```

Example

```
#include <iostream>
#include <string>
using namespace std;

void printDate(int mm, int dd=1);
void printDate(string wkday=”Monday”);

int main()
{
   printDate(10,31);
   printDate(); // uses “Monday”
   printDate(“Tuesday”);
   printDate(7); // uses 7,1
}
```

```
void printDate(int mm, int dd)
{
   cout << mm << “/” << dd;
}


void printDate(string wkday)
{
    cout << wkday;
}
```