More on Functions:
Reference Parameters

#include <iostream>
using namespace std;

double f (double x, int y);

int main (){
  cout << f(10,4) << "\n";
  return 0;
}

double f (double x, int y) {
  if (y==0)
     return 1;
  else
     return (x * f (x, y-1));
}

1E3
Topic 10

■ ADVANCED EXERCISE
■ What does this program print?
■ What does the function f compute?
Objectives

- This topic should allow students to
  - Write functions that do not return a value (aka void functions, or procedures)
  - Recognise the need for and write functions that take by-reference parameters.
  - Choose between functions and procedures.
- Complete Chapter 6 of the textbook.
  - Skip Overloading; skip Default Parameters

void functions

- Our functions till now returned a single value.
- Some sub-tasks don’t return anything
  - E.g. the task of printing output
- These are implemented as void functions in C++
  - void print_results (int StID, double grade);
Print a word in a box

- A function that takes a word and prints it out in a box.

```cpp
void printwordinbox (string word);
```

<table>
<thead>
<tr>
<th>function</th>
</tr>
</thead>
</table>

```cpp
int main () {
    string s;
    cout << "Enter a word:";
    cin >> s;
    printwordinbox (s);
    return 0;
}
```

A void function – it returns nothing.

Just call it - no value returned so nothing to store or print.

Print payroll details

- Our payroll program could be simplified by using a function to print the pay slip.

```cpp
void print_details (string name, double net, double tax, double prsi, double dues);
```

```cpp
int main () {
    //read data, compute pay and tax ....
    print_details (name, net_pay, tax, prsi, union_dues);
    return 0;
}
```

A void function – it returns nothing.

Just call it - no value returned so nothing to store or print.
Void Function Definition

- The function needs no return statement.
- See payroll4fns.cpp for the payroll one.

```cpp
void printwordinbox (string word) {
    int n = word.length();
    for (int i = 1; i <= n+4; i++) {
        cout << '-';
    }
    cout << endl;
    cout << "| " << word << " |" << endl;
    for (int i = 1; i <= n+4; i++) {
        cout << '-';
    }
    cout << endl;
    // no return statement needed
}
```

Need for call-by-reference parameters

- Our functions can only return a single value.
- Sometimes we want to delegate to a function to read in a bunch of data for us.
  - E.g. read payroll details
- Sometimes we want a function to return multiple values
  - E.g. daynumber -> day, month, year
- Sometimes we want a function to change two values in tandem
  - E.g. simplify fraction 7 / 84 -> 1 / 12
Call-by-reference parameters

- For a function to provide us with multiple values, we need to give it a way to update storage places.
- Instead of passing a value into a function, we pass a reference to a place where a value is stored.

Consider a reading function

- Consider the read_record() function which we need for payrollFNS.cpp
  - read_record(numhours, hourlyrate, credit)
- It returns nothing. It prints nothing. What's it supposed to do?
  - To provide values for the variables numhours, hourlyrate, credit in the main program.
WRONG definition of read_details ()

- void read_record (int hours, double rate, double credit) {
  ... 
  cin >> hours; 
  ... 
}
- This won’t change the value of the main program’s variable numhours.

Call-by-value parameters

- Up to now, our parameters are call-by-value
  - The value of the argument is passed in.
  - read_record (numhours, ...) will pass in 0 or whatever numhours happens to be at that time.
  - The parameter variable hours inside read_record function is initialised to 0, and then cin gives it another value.
  - But the main numhours variable is not affected.
- For those who’ve been at the lectures, numhours is at one side of the stage, hours at the other. Only values cross the stage.
Call-by-reference parameters

- We need a way to pass the address of a variable to the read_record() function.
- We want the function to be able to access and update the place where numhours is stored by the main program.
- So we need to pass in a reference to the variable numhours.
- This is call-by-reference.

```cpp
main()

void read_record (int& hours, ..)

hours

read_record (numhours, ..);

One storage location accessed by main as numhours and by the function as hours
```
Call-by-reference syntax

- The function declaration should be
  ```
  void read_record (int& hours, ...){
  ```
- The `&` says “address of” or “reference to”
- `int& hours` can be read as “hours is a reference to an integer variable”.
- The call `read_record (numhours, ...);`
  passes a reference to the variable in to the function,
- Inside the function, this parameter is called `hours`.
- `cin >> hours;` will update the place referenced by `hours`, which in this case is `numhours`.

Which to use

- If you want a function to change the value of a variable it must be a call-by-reference parameter i.e. use `&`
- If the function only needs the value of the parameter don’t use `&`
  - it’s confusing - the reader will expect the function to change the value
  - and dangerous - the function may inadvertently change the value of a main program variable.
void f (int x, int y)

Which is the most likely role of f?

A. Return $x + y$
B. Print $x + y$
C. Swap $x$ and $y$
D. Read in $x$ and $y$

void f (int& x, int& y)

Which is the most likely role of f?

A. Return $x + y$
B. Print $x + y$
C. Swap $x$ and $y$
D. Read in $x$ and $y$
Which declaration?

g returns the max of x and y

A. `void g (int& x, int& y)`
B. `int g (int& x, int& y)`
C. `int g (int x, int y)`
D. `void g (int x, int y)`

Which declaration?

int q = h (5, 75);

A. `void h (int& x, int& y)`
B. `int h (int& x, int& y)`
C. `int h (int x, int y)`
D. `void h (int x, int y)`
Which declaration?

x = 5; y = 10;
p(x, y);
cout << x << y;

A. void p (int& x, int& y)
B. int p (int& x, int& y)
C. int p (int x, int y)
D. void p (int x, int y)

Side-effects

- Value returning functions (i.e. non-void ones) should not have any side-effects.
- Functions should be called for the value they return only.
- If a (value returning) function has side effects
  - E.g. printing, reading, updating a value
- then it won’t be as broadly useful and could be confusing
Clean living!!

- Functions that return a value should have no other effect
  - Including they should not update reference parameters.
- If your function needs to return multiple values use reference parameters for all of them and make it a void function.
- This is recommended practice, not absolute.

```c
void f (int x);
int main () {
  int y = 10;
  f (y);
  cout << "y is " << y << " x is ";
  return 0;
}

void f (int x) {
  x = x + 50;
  cout << "x is " << x;
}
```

No &s.
This prints:

A. y is 10 x is 60
B. x is 60 y is 10
C. y is 60 x is 60
D. x is 60 y is 60
void f (int& x);

int main () {
    int y = 10;
    f (y);
    cout << "y is " << y << " \n";
    return 0;
}

void f (int& x) {
    x = x + 50;
    cout << "x is " << x << " \n";
}

A. y is 10 x is 60
B. x is 60 y is 10
C. y is 60 x is 60
D. x is 60 y is 60

With &s.
This prints:

Exercises

- We can now complete the “functions” version of the payroll program.
- Write a function to swap the values of two integer variables.
- Write a function to order the values of two integer variables, so that \( x \leq y \).
- Convert daynumber to day, month, year.