Structures

Victor Eijkhout and Carrie Arnold and Charlie Dey

Fall 2017

Structures

Bundling information

Sometimes a number of variables belong logically together. For instance two doubles can be the x, y components of a vector.

This can be captured in the `struct` construct.

```
struct vector { double x; double y; } ;
```

(This can go in the main program or before it.)

Initialize:

```
struct vector { double x=0.; double y=0.; } ;
```

```
# this syntax needs:
#   icpc -std=c++11 -o .....
```

The elements of a structure are usually called *members*.

Using structures

Once you have defined a structure, you can make variables of that type. Setting and initializing them takes a new syntax:

Code:

```
int main() {

  struct vector p1,p2;

  p1.x = 1.; p1.y = 2.;
  p2 = {3.,4.};

  p2 = p1;
  cout << "p2: " << p2.x << "," << p2.y << endl;
```

Output:

```
./point
p2: 1,2
```

Period syntax: 'apostrophe-s'.

Functions on structures

You can pass a structure to a function:

```
double distance( struct vector p1,struct vector p2 ) {
  double d1 = p1.x-p2.x, d2 = p1.y-p2.y;
  return sqrt( d1*d1 + d2*d2 );
}
  /* ... */
  cout << "Distance: " << distance(p1,p2) << endl;
```

Returning structures

You can return a structure from a function:

Code:

```
struct vector vector_add
      ( struct vector p1,
struct vector p2 ) {
   struct vector p_add =
     {p1.x+p2.x,p1.y+p2.y};
   return p_add;
};
  /* ... */
  p3 = vector_add(p1,p2);
  cout << "Added: " <<
    p3.x << "," << p3.y << endl;
```

Output:

```
./pointadd
Added: 5,6
```

(Something weird here with scopes: the explanation is that the returned value is copied.)

Exercise 1

Write a function `inner_product` that takes two `vector` structures and computes the inner product.

Exercise 2

Write a 2×2 matrix class (that is, a structure storing 4 real numbers), and write a function `multiply` that multiplies a matrix times a vector.

Can you make a matrix structure that is based on the vector structure, for instance using it to store the matrix columns?

Project Exercise 3

Rewrite the exercise that found a predetermined number of primes, putting the `number_of_primes_found` and `last_number_tested` variables in a structure. Your main program should now look like:

```
cin >> nprimes;
struct primesequence sequence;
while (sequence.number_of_primes_found<nprimes) {
  int number = nextprime(sequence);
  cout << "Number " << number << " is prime" << endl;
}
```