Structures

Victor Eijkhout, Harika Gurram,
Je'aime Powell, Charley Dey

Fall 2018

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.)

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

How to use structures

1. Declare what is in your structure;
2. Make some structures;
3. Use them.

```
// definition of the struct
struct AStructName { int num; double val; }
int main() {
  // declaration of struct variables
  AStructName mystruct1,mystruct2;
  .... code that uses your structures ....
}
```

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 from running point in code directory struct:

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

Period syntax: 'apostrophe-s'.

Struct initialization

You assign a whole struct, or set defaults in the definition.

```
struct vector_a { double x; double y; } ;
// needs compiler option: -std=c++11
struct vector_b { double x=0; double y=0; } ;

int main() {

  // initialization when you create the variable:
  struct vector_a x_a = {1.5,2.6};
  // initialization done in the structure definition:
  struct vector_b x_b;
  // ILLEGAL:
  // x_b = {3.7, 4.8};
  x_b.x = 3.7; x_b.y = 4.8;
  //end snippet

  return 0;
}
```

Functions on structures

You can pass a structure to a function:

Code:

```
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 );
}
  /* ... */
  struct vector p1 = { 1.,1. };
  cout << "Displacement x,y?";
  double dx,dy; cin >> dx >> dy; cout << endl;
  cout << "dx=" << dx << ", dy=" << dy << endl;
  struct vector p2 = { p1.x+dx,p1.y+dy };
  cout << "Distance: " << distance(p1,p2) << endl;
```

Output from running pointfun in code directory struct:

```
Displacement x,y?
dx=5, dy=12
Distance: 13
```

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 from running pointadd in code directory struct:

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

(In case you're wondering about scopes and lifetimes here: 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 vectors to store the matrix rows, and then using the inner product method to multiply matrices?

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;
}
```