Structures

Ajit Rajwade

Structures

- A structure in C++ is a collection of variables.
- The variables in the collection are said to be **members** of the structure.
- For example, consider the following structure definition:

```
struct student {
    char name[100];
    char gender;
    float CPI;
    int roll_number;
};
```

Structures

- A general syntax for structure definition is as follows:

```
struct structure_name {
    member_type1 member_name1;
    member_type2 member_name2;
    .
    .
    member_typeN member_nameN;
};
```

Structures

- A second example:

```
struct book{
    char booktitle[500];
    char author_name[500];
    int year_publication;
    int num_pages;
    float price;
};
```

Structures

- The structure definition by itself does not create variables or reserve any memory.
- For that, we need to declare new variable of type (say) book or student in the following manner:
    - `book b1, b2;`
    - `student s1, s2, s3;`
- A structure by itself thus defines a new “compound” datatype. The different members of a structure contain information about different attributes of the structure.
- To access a particular member, we use syntax of the form: `b1.price = 3000;` or `b1.num_pages = 500;`
- The members of a structure can be used as variables in any expression of your choice.
- The advantage of a structure is that it creates an **object** with many attributes under one umbrella.

Structures

- Structures can be initialized in a manner similar to arrays, for example as follows:

```
book b1 = {“An Introduction to Programming Through C++”,
“Abhiram Ranade”,
2014, 470,400
 };
```

Array of Structures and Pointer to Structure

- We can make arrays of any data-type, including a data-type declared as a structure.
- For example: `book b[50]; student S[800];`
- To access a member of the `i`-th element of the array, we use syntax of the form `b[i].booktitle`, for example.
- Just as you have pointers to `int`, `char`, `double`, etc., you can have pointers to a structure data-type as well.
- Such a pointer contains the base address of the structure variable:

```
book b1, *b2; b2 = &b1;
```

- You can access member variables of the structure pointed to by `b2` in the following way: `(*b2).price` or using `b2->price`.
- The `->` operator works the same as the `(*).` operator.

Structures and Functions

- You can pass structure variables, arrays of structures, or pointers to structures as variables to a function.
- You can also return a structure variable from a function.
- For example, consider:

```
struct point {

double x,y;

};

point midpoint (point a, point b){

     point c;

     c.x = (a.x+b.x)/2; c.y = (a.y+b.y)/2;

     return c;

}

int main(){

point a,b,c; a.x = a.y = 10; b.x = b.y = 40;

c = midpoint(a,b);

}
```

Nesting structures

- A structure can contain any datatype, including another type of structure as its member.
- An example is here below: the structure triangle contains an array of 3 point structures. The structure circle contains a scalar radius value and another structure for the center-point.

```
struct triangle{
    point p[3];
};
struct circle{
    point center;
    double radius;
};
```

Structures: another example - polynomial addition

- Consider a monomial in some variable (say) x: it contains a degree and a coefficient.
- We represent a monomial (in x) as a structure and a polynomial (in x) as an array of monomial structures, arranged in increasing order of degree, as follows:

```
struct monomial {
    float coeff;
    unsigned int degree;
};
```

```
monomial* add_polynomials(monomial* p1, monomial* p2, int n1,int n2, int& n3){
        int cp1, cp2, cp3;
        monomial *p3 = new monomial[n1+n2];  //dynamic memory allocation for an array of structures
        cp1 = cp2 = cp3 = 0;

        while (cp1 < n1 && cp2 < n2){
                if (p1[cp1].degree < p2[cp2].degree){
                        p3[cp3].degree = p1[cp1].degree;  p3[cp3].coeff = p1[cp1].coeff; cp3++; cp1++;
                }

                else if (p2[cp2].degree < p1[cp1].degree){
                        p3[cp3].degree = p2[cp2].degree;  p3[cp3].coeff = p2[cp2].coeff; cp3++; cp2++;
                }
                else{
                                p3[cp3].degree = p2[cp2].degree;
                        p3[cp3].coeff = p1[cp1].coeff+p2[cp2].coeff; cp3++; cp2++; cp1++;
                }
        } // close while

        while (cp1 < n1){
                p3[cp3].degree = p1[cp1].degree;  p3[cp3].coeff = p1[cp1].coeff; cp3++; cp1++;
        }
        while (cp2 < n2){
                p3[cp3].degree = p2[cp2].degree;  p3[cp3].coeff = p2[cp2].coeff; cp3++; cp2++;
        }

        n3 = cp3;
        return p3;
}
```

```
void display_poly(monomial *p, int n)
{
        for(int i = 0; i < n;i++)
        {
                cout << p[i].coeff << "*x^" << p[i].degree;
                if (i < n-1) cout << " + ";
        }
}
```

```
int main(){
        int n1, n2,n3,i;
        cout << "enter the number of terms of the first and second polynomial: ";
        cin >> n1 >> n2;
        monomial *p1, *p2, *p3;
        p1 = new monomial[n1];
        p2 = new monomial[n2];
        cout << "enter the terms of the first polynomial in increasing order: " << endl;
        for(i=0;i<n1;i++) { cout << "enter the coefficient and degree of the next term: " << endl;
           cin >> p1[i].coeff >> p1[i].degree;}
        cout << "enter the terms of the second polynomial in increasing order: " << endl;
        for(i=0;i<n2;i++) { cout << "enter the coefficient and degree of the next term: " << endl;
           cin >> p2[i].coeff >> p2[i].degree;}
        p3 = add_polynomials(p1,p2,n1,n2,n3);
        display_poly(p3,n3);
}
```