Сложные типы языка Си

Шокуров Антон В.
shokurov.anton.v@yandex.ru
http://машинноезрение.рф

25 февраля 2017 г.

Версия: 0.10

Аннотация

Построение нового, собственного типа переменных.
Цель. Построить новый тип из уже имеющихся.
Предварительный вариант!

1 Структуры языка Си

В первом подразделе будет показано как создавать новый тип данных, являющийся составленым из ранее уже имеющихся типов данных.

1.1 Переменные

Допустим в программе нужно обрабатывать сложные объекты, например, точки. Под термином – сложный – понимается, что объект состоит из связанных между собой базовых типов языка Си. Например, точка будет состоять из её компонент, двумерная точка состоит из двух чисел с плавающей точкой.

Важным моментом связанным с объектами является его целостность. Так при выполнении вычислений с такими объектами использую только то, что было ранее введено (т.е. без понятия структур), возникнут ряд сложностей описанных ниже.

Объявление При объявлении сложность заключается в том, что необходимо объявить переменные так, чтобы потом было ясно какие из переменных образует единый объект. В частности, какие из переменных являются компонентами точки и какой у них порядок.

Так, если нужно объявить одну точку, то можно написать код и так:

```
1 double x,y;//одна точка
```

Но, если точек больше, например две, то придется например вводить индекс:

```
1 double x0,y0;//первая точка
2 double x1,y1;//вторая точка
```

В дальнейшем в программе необходимо помнить о том, что переменные с одним и тем же индексом (суффиксом) образуют единый объект, в рассматриваемом случае: первая точка задается как $(x0, y0)$, а вторая как $(x1, y1)$. При таком подходе сам пользователь отвечает за согласованность переменных: при инициализации, копировании, при вычислениях, при передачу в функции и тому подобное. Последнее не удобно и в большинстве случаев ведет к ошибкам.

Язык Си позволяет задавать такие объекты естественным образом, что возлагает большую часть согласованности при взаимодействии с объектом на сам язык. Последние позволяет пользователю сконцентрироваться на самой сути программы. Языковая конструкции языка Си заключается в следующем:

```
1 struct point_2d
2 {
3   double x, y;//составные части объекта
4 };
```

Внутри фигурных скобок пишутся объявления переменных (определять их там нельзя!), которые называются полями структуры. Поля описывают сложные объект. В данном случае они являются координатами точки.

Поля в обще говоря могут быть разных типов. Более того, они, как будет показано нижу, могу в свою очередь быть тоже типом некой структуры. Считается, что поля задают сам объект в целом.

Когда нужный тип данных определен объявления переменных можно записать следующим образом:

```
1 struct point_2d p;//одна точка
2 struct point_2d p0, p1;//ещё две точки.
```

В первой строчке была объявлена переменная p, а во второй строчке объявлено сразу две переменные p0 и p1. Все перечисленные переменные (p, p0 и p1) имеют тип struct point_2d, т.е. являются структурами (составными объектами) с именем (типом) point_2d.

Последнее очень похоже на объявления переменных стандартных типов. Разница как раз заключается в том, что тип в данном случае struct point_2d. Можно

такое название везде в программе и использовать, но это не всегда удобно (слишком длинно). Поэтому его принято сократить за счет использования конструкции *переопределения*.

Переопределение Заметим, что при объявлении приходится всегда писать вначале термин struct до название самого нашего типа (point_2d). Этого можно избежать использую другую языковую конструкцию языка Си: *typedef*. Она позволяет заменять любой сложный тип на более короткое название, сокращение:

```
1 typedef int myintarray[5];
2 //...
3 myintarray a;//У a тип int[5].
4 a[0]=5;
5 a[4]=a[0]+1;
6 myintarray b={5,9,2,-4,3};
```

В последнем примере сложный тип (целочисленный массив из 5 элементов) заменяется на некое обозначение (myintarray). Последнее позволяет далее в программе использовать именно его, а не громоздкое первоначальное (в котором легко ошибиться и в какой-то из очередных объявлений вместо 5 написать, например, 4). В частности, когда решено изменить размер массива (придется везде поменять 5 на, например, 4). Последнее означает, что хорошей практикой проектирования программного обеспечения является использование конструкции typedef.

По аналогии с выше написанном, рассматриваемую конструкцию можно применить и к структурам:

```
1 typedef struct point_2d point2d;
2 //Теперь можно так:
3 point2d p0, p1;//без термина struct.
```

Последнее явно читабельнее и удобнее ранее показанного способа объявления переменных с типом структура.

Вложенные объявления Объекты в Си можно определять не только посредством встроенных типов данных, но и через уже ранее определенные типы. Объект типа прямая (отрезок) можно задать как:

```
1 struct line_2d
2 {
3   //Начальная и конечная точка прямой (отрезка)
4   point2d p0, p1;
5   double weight;
```

```
6 };
7 typedef struct line_2d line2d;
```

В данном определении структуры задается объект отрезок двумя точками. Также добавлено ещё одно поле weight с отличным от предыдущих типом double. Последним показана возможность использования в структурах полей с разным типом данных.

Этим полем можно задавать, например, вес (важность данного отрезка). Последнее важно например в задачах на графах.

1.2 Вычисления

В предыдущем подразделе было показано как описывать свой тип данных и как потом создавать соответствующего типа переменные.

В первом параграфе будет показано как инициализировать (задавать первичное значение) переменным, во втором как выполнять простейшие вычисления над переменными имеющие тип структура.

Инициализация Переменные стандартных типов, как мы знаем, можно инициализировать в момент объявления:

```
1 double x0=1,y0=2;//первая точка
2 double x1=3,y1=y0+2.;//вторая точка
```

Переменные являющиеся структурами такая возможность тоже имеется. В конце концов все сводится к:

```
1 point2d p0={1,2}, p1={3,4};
2 line2d ll0={{1,2}, {3,4}, 3.0};
3 line2d ll1={p0, {3,4}, -2.0};
```

Таким образом значение рассматриваемых переменных задается списком значений, задаваемый парными фигурных скобок. Если какое либо из значений предполагает структуру, то оно задается соответствующим типом: либо ранее объявленной переменной (см. первое поле в стр. 3), либо как и ранее фигурными скобками (см. первое и второе поле в стр. 2, а также второе поле в стр. 3), последнее означает, что фигурные скобки будут вложенными.

Простые вычисления Для начала покажем как выполнять простейшие вычисления, а именно – вычисления над полями сложных переменных. Все конечно сводится к переменной имеющий встроенный тип, например, числовой тип.

Пусть, например, программа вычисляет параллельный сдвиг точки:

```
1 //Параллельный сдвиг:
2 x0 += x1;//Помним о том, что объект состоит
3 y0 += y1;//из  более чем одной переменной.
```

Как уже ранее отмечалось в данном случае пользователю самому приходится помнить из чего составлен каждый из объектов. В данном случае, что точки составлены из компонент.

В случае если данные уже определены, то можно переписать этот код как:

```
1 //Параллельный сдвиг:
2 p0.x += p1.x;//Указывается имя переменной, а далее через
3 p0.y += p1.y;//точку (.) имя поля.
```

Такой подход явно более удобен.

В ещё более явно это проявляется, когда нужно копировать весь объект. Например

```
1 point2d left_most;
2 if( p0.x < p1.x )
3   left_most = p0;
4 else
5   left_most = p1;
```

1.3 Ввод/вывод сложных типов

Помимо вычислений важным является и ввод вывод сложных объектов. Иначе как и ранее было уже отмечено, будет сложно увидеть результат работы программы. В принципе все что было уже сказано ранее достаточно для понимания того как это делать, так как когда происходит переход к полю с базовым типом оно уже ничем не отличается от обычной переменной. А для них все было уже рассказано.

Но тем не менее покажем как это делать.

Вывод данных Стандартные функции вывода (printf и тому подобное) естественно ничего не знают о нашем типе данных. Поэтому и вывести его самостоятельно не смогут:

```
1 //Так вывести компоненты точки не получится:
2 printf("(%f, %f)", p0);//Ошибка!!!
```

Для того чтобы они напечатались необходимо перейти к соответствующим полям самостоятельно:

```
1 //Выводим значения компонентов первой точки:
2 printf("(%f, %f)", p0.x, p0.y);//Печатаем поля первой
3 printf("(%f, %f)", p1.x, p1.y);//точки, а потом второй.
4 //Главное не ошибиться при копировании строчки:
5 printf("(%f, %f)", p0.x, p1.y);//Нет соответствия!
```

Ввод данных Продолжая предыдущее, такой же подход необходимо помнить при вводе данных:

```
1 //Считываем компоненты первой точки:
2 scanf("%lf%lf", &x0, &y0);//Соответствующие переменные
3 scanf("%lf%lf", &x1, &y1);//Считали вторую точку.
4 //Такой командой считается скорее всего не то, что хотели
5 scanf("%lf%lf", &x0, &y1);//Нет соответствия!
```

Упражнение. Реализовать сортировку по какому-либо из полей.

1.4 Функции

Вызов функций...

Вызов функции Допустим нужно вычислить расстояние до точки. В случае единственной точки:

```
1 //Расстояние до точки
2 double dist= sqrt( x * x + y * y );
```

В случае если их большей одной необходимо помнить какие переменные между собой соотносятся. Например расстояние до первой точки ищется как:

```
1 //Расстояние до первой точки
2 double dist = sqrt( x0 * x0 + y0 * y0 );
3 //Неправильным будет написать:
4 double dist2 = sqrt( x1 * x1 + y0 * y0 );
```

При вызове функции необходимо будет передать все поля объекта и убедиться что они относятся к одному объекту.

```
1 double length(double x, double y)
2 {
3   return sqrt( x * x + y * y);
```

```
4 }
5 //...идет некий код
6 //при вызове функции нужно передать соответствующие
7 double l = length(x0, y0);//поля объекта, компоненты.
```

Это уже создает нагромождение переменных, что увеличит шанс ошибки. Тем более, если нужно будет передать больше одной точки:

```
1  double dist(double x0, double y0, double x1, double y1)
2  {
3    double dx = x1 - x0;//разница между первых компонент
4    double dy = y1 - y0;//между вторыми. Аккуратно!
5    return sqrt( dx * dx + dy * dy);
6    //Вместо этого мы могли бы вызвать:
7    //return length( dx, dy);//Так правильнее.
8  }
9  //...идет некий код
10 //при вызове функции нужно передать соответствующие
11 double d = dist(x0, y0, x1, y1);//поля объектов, двух!.
```

Достаточно легко ошибиться даже если использовать структуры.

Выходом являются структуры. Они позволяют передать объект как единое целое, что избавляет от ошибок связанных с полями.

```
1 double length2(point2d p)
2 {
3   return sqrt( p.x * p.x + p.y * p.y);
4 }
5 //...идет некий код
6 //при вызове функции нужно передать просто
7 double l = length(p0);//соответствующий объекта.
```

```
1  double dist2(point2d p0, point2d p1)
2  {
3    double dx = p1.x - p0.x;//разница между полями
4    double dy = p1.y - p0.y;//между вторыми. Аккуратно!
5    return sqrt( dx * dx + dy * dy);
6    //Вместо этого мы могли бы вызвать:
7    point2d dd = {dx, dy};
8    return length( dd );//Так правильнее.
9  }
```

```
10 //...идет некий код
11 //при вызове функции нужно передать соответствующие
12 double d = dist( p0, p1);//два объекта.
```

Указатели Указатели нужны для сокращения объема передаваемой памяти:

```
1  double dist2(point2d *p0, point2d *p1)
2  {
3    double dx = p1->x - p0->x;//разница между полями
4    double dy = p1->y - p0->y;//между вторыми. Аккуратно!
5    return sqrt( dx * dx + dy * dy);
6    //Вместо этого мы могли бы вызвать:
7    point2d dd = {dx, dy};
8    return length( dd );//Так правильнее.
9  }
10 //...идет некий код
11 //при вызове функции нужно передать соответствующие
12 double d = dist( &p0, &p1);//два объекта.
```

Вместо точки (.) используется указатель (->).

Упражнение. Реализовать функцию swap.