Структуры в С++

Структура – это определяемый пользователем тип, который вы определяете, применяя ключевое слово *struct*, поэтому их часто называют *struct*.

Структуры впервые появились в языке С, и С++ включает и расширяет понятие *struct* из С. В С++ структуры функционально заменяются классами, в том смысле, что все, что вы можете делать со структурой, можно достичь, применяя классы.

Однако поскольку Windows была написана на С еще до того. как С++ достиг популярности, структуры широко распространены в программировании Windows.

Поэтому необходимо кое- что о них знать. Первым делом мы рассмотрим *struct,* прежде чем перейти к более развитым возможностям , предоставляемым классами.

Что такое структура?

Почти все переменные, которые вы видели до настоящего момента, могли хранить единственный тип сущности - число определенного вида, символ или массив элементов некоторого типа. Реальный мир сложнее модели и возникает необходимость его в, как минимум, нескольких единицах данных.

Подумаем об информации, которая может понадобиться, чтобы описать нечто простое, например, книгу. Можно принять во внимание название, автора, издательство, дату публикации, цену, количество страниц, тему классификации.

Список можно расширить. Можно определить отдельные переменные для определения каждого из параметров, необходимых для описания книги, но в идеале вам понадобится единый тип данных, скажем, BOOK, который включит в себя все эти параметры. Именно это и позволяет сделать структура.

Определение структуры

Остановимся на описании книги и предположим, что мы хотим включить в него заголовок, автора, издательство и год публикации. Чтобы достичь этого, можно объявить структуру следующего вида:

struct BOOK

{ char Title[80];

char Author[80];

char Publisher[80];

int Year;

};

Создается новый тип переменных по имени *BOOK*. Ключевое слово *struct* определяет тип как таковой, и элементы, составляющие объект этого типа, определены в фигурных скобках.

Каждая строка, определяющая отдельный элемент структуры, ограничена точкой с запятой, и точка с запятой появляется после закрывающей скобки.

Элементы структуры могут быть любого типа, за исключением самого типа структуры, внутри которой они определены. Нельзя иметь внутри определения структуры *BOOK* элемент типа *BOOK*.

Элементы *Title,* *Author*, *Publisher*, *Year*,заключенные могут быть названы членами или полями структуры *BOOK*. Каждый объект типа *BOOK* содержит поля

*Title*, *Author*, *Publisher*, *Year*. Можно создавать переменные типа *BOOK* точно таким же способом, как создается переменные любого другого типа.

*BOOK Novel; //Объявление переменной Novel типа BOOK*

Это объявляет переменную по имени *Novel*, которую теперь можно использовать для хранения информации о книге.

Рассмотрим , как установить данные в различных полях, составляющих переменную типа *BOOK*.

Инициализация структуры

Первый способ установить данные в члены структуры это определить их начальные значения в объявлении. Предположим, что вы хотите инициализировать переменную *Novel*, чтобы она содержала данные о книге Programming C++, выпущенной в 2005 году издательством Gutten Press.

Можно написать объявление переменной Novel следующим образом.

*BOOK Novel =*

*{*

*“Programming C++”, //Начальное значение Title*

*“Stiven Prata”, // Начальное значение Author*

*“Gutten Press”, // Начальное значение Publisher*

*2005 // Начальное значение Year*

*};*

Инициализирующие значения появляются между скобками и разделены запятыми – почти так же . как определяются начальные значения элементов массива.

Последовательность инициализирующих значений очевидно должна совпадать с последовательностью полей структуры в ее определении. Каждое поле структуры *Novel* имеет соответствующее значение. присвоенное ему, на что указывают комментарии.

Доступ к членам структуры

Чтобы обратиться к индивидуальным членам структуры, можно использовать операцию выбора члена (операцию доступа к члену), которая обозначается точкой. Чтобы сослаться на определенный член структуры, пишется имя переменной – структуры, за которым следует точка, а за ней – имя члена, к которому необходимо обратиться, для изменения поля *Year* структуры *Novel* можно написать так:

*Novel. Year = 2011;*

это установит значение поля *Year* равным 2011. Можно использовать поле структуры таким же образом, как любую другую переменную того же типа, что и это поле.

*Novel. Year += 2;*

Эта операция увеличит значение поля на 2.

Пример использования структуры.

Имеет место двор, бассейн. Поскольку они прямоугольны (на плане), можно определить тип структуры для определения любого из этих объектов.



*struct RECTANGLE*

*{*

*int Left; //Пара координат*

*int Top; //верхнего левого угла*

*int Right; //Пара координат*

*int Bottom; //нижнего правого угла*

*};*

Первые два поля структуры *RECTANGLE* соответствуют координатам верхней левой точки прямоугольника, а следующие две координаты – правой нижней точке. Можно использовать это в элементарном примере, имеющим дело с объектами двора.

//listing 33

//использование структур для описания двора

#include <iostream.h>

//определение структуры, описывающей прямоугольник

struct RECTANGLE

{

int Left; //Пара координат

int Top; //верхнего левого угла

int Right; //Пара координат

int Bottom; //нижнего правого угла

};

//прототип функции вычисления площади прямоугольника

long Area(RECTANGLE& aRect);

int main (void)

{

RECTANGLE Yard = {0, 0, 100, 120};

RECTANGLE Pool = {30, 40, 70, 80};

cout << endl

<<” Yard area:”

<< Area(Yard);

cout << endl

<<” Pool area:”

<< Area(Pool);

return 0;

}

// функция вычисления площади прямоугольника

long Area (RECTANGLE& aRect)

{

return (aRect. Right - aRect. Left)\*( aRect. Bottom - aRect. Top);

}

Результат

Yard area:12000

Pool area:1300

Описание полученных результатов

Здесь определена функция для обработки объектов *RECTANGLE – Area(),*которая вычисляет площадь объектов этого типа, переданного в аргументе ссылке, как произведение длины и ширины, где длина - разница между горизонтальными позициями конечных точек, а ширина – разница между их вертикальными позициями.

В функции *main ()* инициализируются переменные *Yard* и *Pool* типа *RECTANGLE* координатами их позиций.

Структура RECT

Прямоугольники в программах Windows используются очень широко. По этой при­чине существует предопределенная структура RECT в заголовочном файле windows. h. Ее определение, по сути, совпадает со структурой, которую вы видели в предыдущем примере:

*struct RECT*

*{*

*int left;*

*int top;*

*int right;*

*int bottom; } ;*

Эта структура обычно используется для определения прямоугольных областей на вашем дисплее для множества различных целей. Поскольку RECT используется столь интенсивно, windows. h также предоставляет функцию InflateRect () для увеличения размера прямоугольника и функцию EqualRect () для сравнения двух прямоугольников.

В библиотеке MFC также определен класс по имени CRect, который эквивалентен структуре RECT. После изучения классов вы отдадите ему предпочтение перед структурой RECT. Класс CRect предлагает широкий набор функций для манипулирования прямоугольниками, и вы используете многие из них, когда будете писать программы Windows с применением MFC.

**Использование указателей со структурами**

Как и следовало ожидать, вы можете создавать указатель на переменную типа структуры. Фактически многие функции, объявленные в windows .h, которые работают с объектами RECT, требуют указателей на RECT в качестве аргументов, поскольку это позволяет избежать копирования всей структуры при передаче в функцию аргумента типа RECT. Чтобы определить указатель на объект RECT, используется вполне предсказуемое объявление:

RECT\* pRect = NULL; // Определение указателя на RECT

Предполагая, что у вас определен объект типа RECT по имени aRect, вы можете установить указатель на адрес этой переменной обычным образом, используя опера­цию взятия адреса:

pRect = &aRect; //присвоить указателю адрес aRect

Как вы видели, когда была представлена идея структуры, struct не может содержать член того же типа, как она сама, однако она может содержать указатель на struct того же типа. Например, вы можете определить структуру примерно так:

struct ListElement {

RECT aRect; // Член структуры RECT

ListElement\* pNext; // Указатель на элемент списка

};

Первый элемент структуры ListElement имеет тип RECT, а второй - указатель на структуру типа ListElement - тот же тип, внутри которого он определен. (Следует подчеркнуть, что этот элемент не относится к типу ListElement; его типом является "указатель на ListElement".) Это позволяет связывать объекты типа ListElement в цепочки, в которых каждый ListElement может содержать адрес следующего объек­та ListElement в цепочке, а последний элемент цепочки будет иметь нулевой указа­тель.

**Доступ к членам структуры через указатель**

Рассмотрим следующие операторы:

RECT aRect = { 0, 0, 100, 100 };

RECT\* pRect = &aRect;

Первый оператор объявляет и определяет объект aRect типа RECT с первой па­рой членов инициализированных (0, 0) и второй парой (100, 100). Второй оператор объявляет pRect как указатель на тип RECT и инициализирует его адресом aRect.

Теперь вы можете обращаться к членам aRect через указатель, используя оператор вроде следующего:

(\*pRect) .Тор +=10; // Увеличит член Тор на 10

Скобки вокруг разыменования указателя здесь важны, потому что операция доступа к члену имеет более высокий приоритет, чем операция разыменования. Без скобок вы попытаетесь трактовать указатель как struct и разыменовывать ее член, так что этот оператор компилироваться не будет. После выполнения этого оператора член Тор бу­дет иметь значение 10, а остальные члены, конечно же, останутся без изменений.

Метод, используемый здесь для доступа к членам структуры через указатель, вы­глядит достаточно громоздко. Поскольку операции подобного рода часто встречаются в С++, язык включает специальную операцию, позволяющую вам выразить то же самое в намного более читабельной и интуитивной форме, поэтому давайте рассмотрим следующую операцию.

**Операция непрямого выбора члена**

**Операция непрямого выбора члена,** ->, специально предназначена для доступа к членам структур через указатель; эта операция также называется **операцией не­прямого доступа к члену.** Она выглядит как маленькая стрелочка (->) и состоит из знака "минус" (-), за которым следует символ "больше" (>). Вы можете использовать ее, чтобы переписать оператор доступа к члену Тор структуры aRect через указатель pRect, как показано ниже:

pRect->Top +=10; //Увеличить поле Top на 10