// Начало мантры

#include <stdio.h>

#include <stdlib.h>

// Конец мантры

struct Date

// здесь описываются атрибуты (поля) структуры

{

int year; // год даты

char month; // месяц даты

char day; // день даты

};

// описанная выше структура - это только тип переменной

// но не сама переменная

// а struct Date - это имя ТИПА

//использование

int main(){

//объявляем переменную типа struct Date

struct Date oneDay;

// Объявляем переменную-указатель на структуры типа Date

// Эта переменная занимает место только для хранения адреса!!

struct Date \*ptrDate;

// Переменной можно сразу задать начальное значение (инициализация)

int days = 0;

// получим значения для атрибутов переменной oneDay

scanf("%d%hhd%hhd",&(oneDay.year), &(oneDay.month), &(oneDay.day));

// Присвоим адрес переменной oneDay указателю

ptrDate = &oneDay;

//========= Обращение к атрибутам структуры через указатель ===========

/\*

Если мы используем имя переменной-структуы, то обращение к полям происходит

через ".": oneDay.year

Если указатель имеет значение реального адреса переменной-структуры,

то обращение к атрибутам этой структуры происходит через "->"

\*/

// обращение к атрибуту year

days += ptrDate->year \* 12 \* 30;

// обращение к атрибуту mount

days += ptrDate->month \* 30;

// обращение к атрибуту day

days += ptrDate->day;

// печать значений атрибутов переменной oneDay (для проверки)

printf("oneDay: %04d.%02hhd.%02hhd\n",ptrDate->year, ptrDate->month, ptrDate->day);

printf("дней с начала отсчета: %d\n",days);

return 0;

}

Запускаем работу программы

>./ptr\_st

2 9 10

oneDay: 0002.09.10

дней с начала отсчета: 1000

>

**Сложные структуры**

Структурные переменные еще называют **объектами**. При этом каждый такой **объект** может иметь в качестве атрибутов (полей) другие **объекты** - структурные переменные.

**Указатели на атрибуты (поля) объектов (структурных переменных)**

Каждый атрибут объекта - это тоже переменная, которая занимает место в памяти и имеет адрес. Значит можно получить адрес каждого атрибута и работать с ним также как и с указателями обычных переменных.

#include <stdio.h>

#include <stdlib.h>

// Допустим описана подобная структура

typedef struct Coord

{

int x; // координата по x

int y; // координата по y

} Point;

// Примеры работы с указателями на атрибуты

int main(){

// Инициализация (присвоение начальных значений)

// первое число - первому атрибуту (x)

// второе число - второму атрибуту (y)

Point coord = {7, 5}; // объект coord

Point \*directPoint; // уаказатель на объект типа Point

int \*px, \*py; // указатели на int

directPoint = &coord; // адрес coord присваивается directPoint

// Присваивание адресов атрибутов coord:

// Сначала получаем доступ к атрибуту,

// а затем вычисляем его адрес

px = &(coord.x);

py = &(coord.y);

// Отладочная наглядная печать

printf("Object! px: %p coord.x: %d\n", px, coord.x);

printf("Object! py: %p coord.y: %d\n", py, coord.y);

// Присваивание адресов атрибутов через указатель directPoint:

// Сначала получаем доступ к атрибуту,

// а затем вычисляем его адрес

px = &(directPoint->x);

py = &(directPoint->y);

// Отладочная наглядная печать

printf("Direct! px: %p coord.x: %d\n", px, coord.x);

printf("Direct! py: %p coord.y: %d\n", py, coord.y);

// Получение значений атрибутов через scanf

scanf("%d%d", &(coord.x), &(coord.y));

// Отладочная наглядная печать

printf("scanf! coord.x: %d\n", coord.x);

printf("scanf! coord.y: %d\n", coord.y);

return 0;

}

Запуск программы:

./direct

Object! px: 0x7ffe2c520d10 coord.x: 7

Object! py: 0x7ffe2c520d14 coord.y: 5

Direct! px: 0x7ffe2c520d10 coord.x: 7

Direct! py: 0x7ffe2c520d14 coord.y: 5

12 56

scanf! coord.x: 12

scanf! coord.y: 56

// Начало мантры

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

// Конец мантры

// Дадим структуре struct Coord синоним Point - новое имя типа

typedef struct Coord

// точка на плоскости

{

short int x; // координата точки x

short int y; // координата точки y

} Point;

// теперь вместо struct Coord a;

// можно написать просто Point a;

typedef struct LineSegment

{

// описание вершин отрезка

// Point - это структура struct Coord

Point a; // вершина a

Point b; // вершина b

float distance; // расстояние между вершинами (длина отрезка)

} Line; // новое имя типа Line

int main(){

//объявляем переменную типа Line

Line lin;

// Присвоим значения координатам вершин отрезка lin

lin.a.x = 0; // вершина a, координата x

lin.a.y = 0; // вершина a, координата y

lin.b.x = 3; // вершина b, координата x

lin.b.y = 4; // вершина b, координата y

// вычислим длину отрезка lin и присвоим это значение атрибуту distance

lin.distance = sqrtf((lin.a.x - lin.b.x) \* (lin.a.x - lin.b.x) + (lin.a.y - lin.b.y) \* (lin.a.y - lin.b.y));

// печать

printf("lin: a(%d, %d) b(%d, %d), %0.2f\n", lin.a.x, lin.a.y, lin.b.x, lin.b.y, lin.distance);

return 0;

}

>./line

lin: a(0, 0) b(3, 4), 5.00

>