

1.7. Структуры данных.

Thursday, April 23, 2020 0:55

В уроке 1.5 (Массивы) мы познакомились с массивами, которые позволяют нам группировать элементы одного типа в один целостный объект. Чтобы обратиться к элементу массива, достаточно просто вызвать переменную массива и указать позицию элемента в массиве.

В языке C существует еще один способ группировки элементов, который называется *структурой*.

В этом уроке мы рассмотрим:

- Объявление структур
- Передачу структур в функции
- Массивы структур
- Структуры массивов

Понятие структуры

Допустим, нам нужно хранить значение текущей даты, например, 25/9/15, которое мы будем выводить на экран или проводить над ним некие вычисления. Очевидно, нам ничто не мешает хранить значение числа, месяца и года в отдельных переменных, вроде:

```
int day = 25, month = 9, year = 2015;
```

Но что если нам нужно хранить не одну дату, а несколько дат. Например, сегодняшнюю дату или даты, введенные пользователем. Придется создавать по три переменные для каждой даты. А еще помимо этого куча других переменных для других функций программы. Запутаетесь.

Структуры помогают нам группировать связанные между собой переменные в единый объект, чтобы с ними было проще работать. Рассмотрим, как с ними работать.

Структура для хранения даты

Начнем с объявления структуры с датой. Синтакс выглядит следующим образом:

```
1 struct date
2 {
3     int day;
4     int month;
5     int year;
6 };
```

Так, мы объявили структуру `date`, которая состоит из трех переменных типа `int`: `day`, `month` и `year`.

Фактически, созданная нами структура также является новым типом данных. И мы можем создавать переменные типа `date`, которые также будут состоять из трех "подпеременных", – `day`, `month` и `year`.

Синтакс для объявления переменных типа `date` будет выглядеть следующим образом:

```
struct date today, purchaseDate;
```

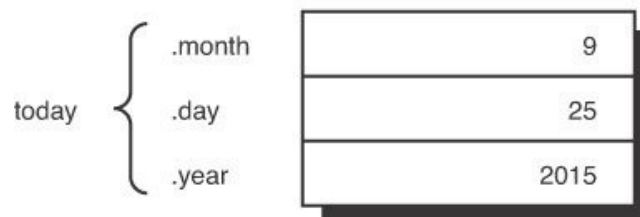
Легким движением руки мы теперь храним сегодняшнюю дату `today` и дату покупки `purchaseDate`.

Обращение к переменным структуры также немного отличается от обращения к обычным переменным. Например, если нам нужно установить значение `day` переменной `today`, нужно написать:

```
today.day = 25;
```

То есть, данные в переменной `today` типа `date` хранятся следующим образом:

```
today.month = 9;
today.day = 25;
today.year = 2015;
```



Напишем программу, в которой мы объявим структуру date, переменную типа date, заполним ее и выведем на экран.

```
1 // Program to illustrate a structure
2
3 #include <stdio.h>
4
5 int main (void)
6 {
7     struct date
8     {
9         int day;
10        int month;
11        int year;
12    };
13
14    struct date today;
15
16    today.day = 25;
17    today.month = 9;
18    today.year = 2015;
19
20    printf ("Today's date is %i/%i/%.2i.\n",
21           today.day, today.month, today.year % 100);
22
23    return 0;
24 }
```

Программа выдаст:

```
Today's date is 9/25/15.
```

Выражения работают со структурами точно так же, как и с обычными переменными. Например, если мы поделим элемент структуры типа int на целое число, то деление будет проводиться как целочисленное:

```
century = today.year / 100 + 1;
```

Структуры и функции

Напишем программу, которая рассчитывает дату следующего дня. Нужно учесть, что в месяцах разное количество дней, а также есть високосные годы.

Количество дней в месяцах мы занесем в массив констант и перед определением числа дней в феврале будем проверять текущий год на високосность.

```

1 // Program to determine tomorrow's date
2
3 #include <stdio.h>
4 #include <stdbool.h>
5
6 struct date
7 {
8     int day;
9     int month;
10    int year;
11 };
12
13 int main (void)
14 {
15     struct date today, tomorrow;
16     int numberOfDays (struct date d);
17
18     printf ("Enter today's date (dd mm yyyy): ");
19     scanf ("%i%i%i", &today.day, &today.month, &today.year);
20
21     if ( today.day != numberOfDays (today) ) {
22         tomorrow.day = today.day + 1;
23         tomorrow.month = today.month;
24         tomorrow.year = today.year;
25     }
26     else if ( today.month == 12 ) { // end of year
27         tomorrow.day = 1;
28         tomorrow.month = 1;
29         tomorrow.year = today.year + 1;
30     }
31     else { // end of month
32         tomorrow.day = 1;
33         tomorrow.month = today.month + 1;
34         tomorrow.year = today.year;
35     }
36
37     printf ("Tomorrow's date is %i/%i/%.2i.\n",
38             tomorrow.day, tomorrow.month, tomorrow.year % 100);
39
40     return 0;
41 }
42
43 // Function to find the number of days in a month
44
45 int numberOfDays (struct date d)
46 {
47     int days;
48     bool isLeapYear (struct date d);
49     const int daysPerMonth[12] =
50         { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
51
52     if ( isLeapYear (d) == true && d.month == 2 )
53         days = 29;
54     else
55         days = daysPerMonth[d.month - 1];
56
57     return days;
58 }
59
60 // Function to determine if it's a leap year
61
62 bool isLeapYear (struct date d)
63 {
64     bool leapYearFlag;
65
66     if ( (d.year % 4 == 0 && d.year % 100 != 0) ||

```

```

67         d.year % 400 == 0 )
68         leapYearFlag = true;    // It's a leap year
69     else
70         leapYearFlag = false;   // Not a leap year
71
72     return leapYearFlag;
73 }

```

Протестируем программу.

```

Enter today's date (dd mm yyyy): 28 2 2016
Tomorrow's date is 29/2/16.

```

Первое, на что обращаем внимание – структура `date` объявлена вне функций, – как глобальная переменная. Это позволяет нам создавать переменные типа `date` из любой функции. Структуры точно так же, как и переменные, бывают локальными и глобальными. То есть, если мы объявим структуру внутри функции, она будет локальной и будет доступна только внутри функции.

В функции `main()` прототип

```
int numberOfDays (struct date d);
```

сообщает компилятору, что функция `numberOfDays()` возвращает значение типа `int` и получает на вход аргумент типа `struct date`.

Для проверки окончания месяца мы используем строку

```
if ( today.day != numberOfDays (today) )
```

Здесь мы передаем в функцию переменную типа `date`. Чтобы функция поняла, что это за тип такой, нужно его указать при объявлении функции:

```
int numberOfDays (struct date d)
```

Функции работают с переданными в них структурами как с переменными, а не как с массивами. То есть, внутри функции создается копия структуры, над которой уже проводится дальнейшая работа. А переданная в функцию переменная остается без изменений.

Также обратите внимание на ввод даты – три цифры через пробел. Реализуется такой ввод следующим образом:

```
scanf ("%i%i%i", &today.day, &today.month, &today.year);
```

И еще маленькая, но важная деталь. Мы назвали функцию проверки високосного года `isLeapYear`, что сразу помогает понять, что функция возвращает бинарное значение ("да" или "нет"). Всегда стоит помнить про читабельность кода.

Перенесем код из функции `main()` в отдельную функцию, которая будет возвращать структуру.

```

1 // Function to calculate tomorrow's date
2
3 struct date dateUpdate (struct date today)
4 {
5     struct date tomorrow;
6     int numberOfDays (struct date d);
7
8     if ( today.day != numberOfDays (today) ) {
9         tomorrow.day = today.day + 1;
10        tomorrow.month = today.month;
11        tomorrow.year = today.year;
12    }
13    else if ( today.month == 12 ) {    // end of year
14        tomorrow.day = 1;

```

```

15         tomorrow.month = 1;
16         tomorrow.year = today.year + 1;
17     }
18     else {                                     // end of month
19         tomorrow.day = 1;
20         tomorrow.month = today.month + 1;
21         tomorrow.year = today.year;
22     }
23
24     return tomorrow;
25 }

```

Таким образом, мы можем не только передавать структуры в функцию, но и создавать функции, которые будут возвращать созданные нами структуры.

Структура для хранения времени

Теперь предположим, что нам нужно хранить время, т.е. часы, минуты и секунды. По аналогии с датой, воспользуемся структурой:

ы

По аналогии с датами, напомним программу, которая будет увеличивать текущее время на 1 секунду. Сначала рассмотрим, что для этого нужно.

1. Если количество секунд достигает 60, их нужно сбросить до 0 и увеличить количество минут на 1.
2. Если количество минут достигает 60, их нужно сбросить до 0 и увеличить количество часов на 1.
3. Если количество часов достигает 24, то часы, минуты и секунды нужно сбросить до 0.

Программа будет использовать функцию `timeUpdate`, которая в качестве аргумента получает текущее время и возвращает время, которое будет через секунду.

```

1  // Program to update the time by one second
2
3  #include <stdio.h>
4
5  struct time
6  {
7      int hour;
8      int minutes;
9      int seconds;
10 };
11
12 int main (void)
13 {
14     struct time timeUpdate (struct time now);
15     struct time currentTime, nextTime;
16
17     printf ("Enter the time (hh:mm:ss): ");
18     scanf ("%i:%i:%i", &currentTime.hour,
19             &currentTime.minutes, &currentTime.seconds);
20
21     nextTime = timeUpdate (currentTime);
22
23     printf ("Updated time is %.2i:%.2i:%.2i\n", nextTime.hour,
24             nextTime.minutes, nextTime.seconds );
25
26     return 0;
27 }
28
29 // Function to update the time by one second
30

```

```

31 struct time timeUpdate (struct time now)
32 {
33     ++now.seconds;
34
35     if ( now.seconds == 60 ) { // next minute
36         now.seconds = 0;
37         ++now.minutes;
38
39         if ( now.minutes == 60 ) { // next hour
40             now.minutes = 0;
41             ++now.hour;
42
43             if ( now.hour == 24 ) // midnight
44                 now.hour = 0;
45         }
46     }
47
48     return now;
49 }

```

Протестируем программу.

```

Enter the time (hh:mm:ss): 23:59:59
Updated time is 00:00:00

```

Снова обратите внимание на то, как реализован ввод. Выражение

```
"%i:%i:%i"
```

указывает на то, что программа будет ожидать три целых числа через двоеточие. В последующих уроках мы разберем ввод подробнее.

Инициализация структур

Как и с другими переменными, если структурная переменная автоматического типа, она будет инициализирована каждый раз при вызове функции, в которой она объявлена. Если же переменная статическая, она будет инициализирована только на старте программы.

Инициализация структур похожа на инициализацию массивов – элементы пишутся в фигурных скобках через запятую. Например, вот так инициализируется переменная `today` типа `date`:

```
struct date today = { 14, 2, 2015 };
```

Дата будет выставлена на 14 февраля 2015 года. Порядок чисел в скобках должен совпадать с порядком объявленных элементов в структуре.

Можно задавать не все значения, также по аналогии с массивами. Например, выражение

```
struct date today = { 14, 2 };
```

задаст только число и месяц, а год будет некоторым неопределенным значением (если переменная автоматического типа продолжительности).

Также можно задавать произвольные элементы в произвольном порядке:

```
struct date today = { .year = 2015 };
```

Составные литералы (compound literals)

В языке C можно присваивать структуре одно или несколько значений, используя *составные литералы*. Например, переменной `today` типа `date` можно присвоить значения следующим образом:

```
today = (struct date) { 14, 2, 2015 };
```

Да, это не объявление переменной, а присваивание ей значения, так что оно может быть в любом месте программы. И здесь работают такие же правила, как и при инициализации структур: если пишем просто через запятую, нужно соблюдать порядок.

Либо явно указывать каждый элемент структуры и присваивать элементам значения в произвольном порядке:

```
today = (struct date) { .month = 9, .day = 25, .year = 2015 };
```

Перепишем функцию `dateUpdate()`, чтобы она использовала для присваивания нового значения даты использовала составные литералы.

```
1 // Function to calculate tomorrow's date - using compound literals
2
3 struct date dateUpdate (struct date today)
4 {
5     struct date tomorrow;
6     int numberOfDays (struct date d);
7
8     if ( today.day != numberOfDays (today) )
9         tomorrow = (struct date) { today.day + 1, today.month, today.year };
10    else if ( today.month == 12 )        // end of year
11        tomorrow = (struct date) { 1, 1, today.year + 1 };
12    else                                // end of month
13        tomorrow = (struct date) { 1, today.month + 1, today.year };
14
15
16    return tomorrow;
17 }
```

Составные литералы используются по желанию. В примере выше они упростили чтение кода. Также мы можем их использовать в любых местах, где допустимо использование структур. Например, ничто нам не мешает вот так передать значение в функцию, без использования переменной:

```
nextDay = dateUpdate ((struct date) { 5, 11, 2004 } );
```

Но это часто непрактично.

Массивы структур

Структуры создали нам возможность группировать функционально связанные элементы. Так, если нам, например, нужно хранить 10 значений времени, благодаря структурам, мы можем использовать только 10 переменных вместо 30.

Но еще более оптимальным методом для такого случая является комбинация структуры и массива. Массивы в языке C не ограничены стандартными типами данных; ничто нам не мешает создавать массивы структур. Например,

```
struct date birthdays[15];
```

определяет массив `birthdays` из 15 элементов типа `date`.

Запись во второй элемент массива будет выглядеть вот так:

```
birthdays[1].month = 8;
birthdays[1].day   = 8;
birthdays[1].year  = 1986;
```

Дальше авторы учебника вместо продолжения примеров с датой, совершенно неважно

переключаются на примеры со временем. Надо будет потом исправить.

Для передачи в функцию `checktime()` элемента `experiments[4]` типа `time`, достаточно просто указать массив с индексом элемента:

```
checkTime (experiments[4]);
```

Инициализация массивов структур похожа на инициализацию многомерных массивов. Выражение

```
struct time runTime [5] =  
    { {12, 0, 0}, {12, 30, 0}, {13, 15, 0} };
```

устанавливает первым трем элементам массива `runTime` значения 12:00:00, 12:30:00 и 13:15:00. Внутренние скобки не обязательны, то есть можно и вот так:

```
struct time runTime[5] =  
    { 12, 0, 0, 12, 30, 0, 13, 15, 0 };
```

А выражение

```
static struct time runTime[5] = { [1].hour = 12, [1].minutes = 30 };
```

объявляет статический массив `runTime` и инициализирует только часы и минуты во втором элементе.

Напишем программу, которая нагляднее проиллюстрирует работу с массивами структур. Она будет увеличивать время в элементах массива на одну секунду, при помощи функции `timeUpdate()`, как это делалось в программе из предыдущей главы.

```
1 // Program to illustrate arrays of structures
2
3 #include <stdio.h>
4
5 struct time
6 {
7     int hour;
8     int minutes;
9     int seconds;
10 };
11
12 int main (void)
13 {
14     struct time timeUpdate (struct time now);
15     struct time testTimes[5] =
16         { { 11, 59, 59 }, { 12, 0, 0 }, { 1, 29, 59 },
17           { 23, 59, 59 }, { 19, 12, 27 } };
18     int i;
19
20     for ( i = 0; i < 5; ++i ) {
21         printf ("Time is %.2i:%.2i:%.2i", testTimes[i].hour,
22               testTimes[i].minutes, testTimes[i].seconds);
23
24         testTimes[i] = timeUpdate (testTimes[i]);
25
26         printf (" ...one second later it's %.2i:%.2i:%.2i\n",
27               testTimes[i].hour, testTimes[i].minutes, testTimes[i].seconds);
28     }
29
30     return 0;
31 }
32
33 struct time timeUpdate (struct time now)
34 {
35     ++now.seconds;
```



```

36
37     if ( now.seconds == 60 ) {      // next minute
38         now.seconds = 0;
39         ++now.minutes;
40
41         if ( now.minutes == 60 ) { // next hour
42             now.minutes = 0;
43             ++now.hour;
44
45             if ( now.hour == 24 ) // midnight
46                 now.hour = 0;
47         }
48     }
49     return now;
51 }

```

Программа выдаст:

```

Time is 11:59:59 ...one second later it's 12:00:00
Time is 12:00:00 ...one second later it's 12:00:01
Time is 01:29:59 ...one second later it's 01:30:00
Time is 23:59:59 ...one second later it's 00:00:00
Time is 19:12:27 ...one second later it's 19:12:28

```

Здесь массиву `testTimes` задаются начальные значения: 11:59:59, 12:00:00, 1:29:59, 23:59:59, и 19:12:27 соответственно. В оперативной памяти компьютера это будет выглядеть следующим образом:

testTimes[0]	{	.hour	11
		.minutes	59
		.seconds	59
testTimes[1]	{	.hour	12
		.minutes	0
		.seconds	0
testTimes[2]	{	.hour	1
		.minutes	29
		.seconds	59
testTimes[3]	{	.hour	23
		.minutes	59
		.seconds	59
testTimes[4]	{	.hour	19
		.minutes	12
		.seconds	27

Каждая структура `time` хранится в соответствующей ячейке массива `testTimes`. Доступ к индивидуальным элементам структуры осуществляется добавлением точки перед их названием. Понятие массивов структур очень важно при работе с языком C. **Убедитесь, что вы разобрались в этой теме прежде, чем продолжите курс.**

Структуры структур

Язык C позволяет также создавать структуры, содержащие другие структуры.

Мы уже видели, как можно сгруппировать в структуры дату и время. Но бывают случаи, когда дату и время удобнее представлять как единую структуру. Например, когда программа должна выполнять какое-то действие в определенное время и дату.

Создадим структуру `dateAndTime`, содержащую в себе элементы типа `date` и `time`:

```
struct dateAndTime
{
    struct date    sdate;
    struct time    stime;
};
```

То есть, нужно не просто указать сами структуры, а объявить их в виде переменных, в нашем случае мы решили их назвать `sdate` и `stime`.

Теперь можем объявить переменную `event` типа `dateAndTime`:

```
struct dateAndTime event;
```

Синтаксис обращения к элементу структуры остается неизменным:

```
event.sdate
```

Таким образом, мы можем обновлять дату и время, обращаясь к одной структуре, а не к нескольким:

```
event.sdate = dateUpdate (event.sdate);
event.stime = timeUpdate (event.stime);
```

А чтобы сослаться на какой-то элемент внутри вложенной структуры, нужно написать:

```
event.sdate.month = 10;
```

Можно и операторы применять:

```
++event.stime.seconds;
```

Инициализация выглядит так:

```
struct dateAndTime event =
    { { 1, 2, 2015 }, { 3, 30, 0 } };
```

Здесь задается дата 1 февраля 2015 и время 3:30:00.

Можно и не по порядку, только индивидуальные элементы:

```
struct dateAndTime event =
    { { .month = 2, .day = 1, .year = 2015 },
      { .hour = 3, .minutes = 30, .seconds = 0 }
    };
```

Можно создать массив структур структур:

```
struct dateAndTime events[100];
```

Обращение к элементу массива выглядит следующим образом:

```
events[i].sdate = dateUpdate (events[i].sdate);
```

Чтобы выставить время 12:00:00 в первом элементе массива, нужно написать:

```
events[0].stime.hour = 12;
```

```
events[0].stime.minutes = 0;
events[0].stime.seconds = 0;
```

Структуры, содержащие массивы

Как очевидно из заголовка, можно создавать структуры, содержащие массивы в качестве элементов этих структур. Одно из самых частых применений – знаковые массивы. Например, мы хотим создать структуру под названием "month", содержащую количество дней в каждом месяце и аббревиатуры названия каждого месяца. Такая структура будет выглядеть следующим образом:

```
struct month
{
    int    numberOfDays;
    char   name[3];
};
```

Мы задали массив name из трех знаков, потому что этого достаточно для аббревиатур.

Теперь можно создать переменную типа month:

```
struct month aMonth;
```

Можно задать элементам переменной значения последовательностью выражений:

```
aMonth.numberOfDays = 31;
aMonth.name[0] = 'J';
aMonth.name[1] = 'a';
aMonth.name[2] = 'n';
```

Или инициализировать переменную при объявлении:

```
struct month aMonth = { 31, { 'J', 'a', 'n' } };
```

Лучшим решением будет создать не отдельные переменные для каждого месяца, а массив из 12 элементов, – по одному на каждый месяц:

```
struct month months[12];
```

Напишем программу, в которой мы зададим значения массиву months и выведем их на экран.

```
1 // Program to illustrate structures and arrays
2
3 #include <stdio.h>
4
5 int main (void)
6 {
7     int i;
8
9     struct month
10    {
11        int    numberOfDays;
12        char   name[3];
13    };
14
15    const struct month months[12] =
16        { { 31, {'J', 'a', 'n'} }, { 28, {'F', 'e', 'b'} },
17          { 31, {'M', 'a', 'r'} }, { 30, {'A', 'p', 'r'} },
18          { 31, {'M', 'a', 'y'} }, { 30, {'J', 'u', 'n'} },
19          { 31, {'J', 'u', 'l'} }, { 31, {'A', 'u', 'g'} },
20          { 30, {'S', 'e', 'p'} }, { 31, {'O', 'c', 't'} },
21          { 30, {'N', 'o', 'v'} }, { 31, {'D', 'e', 'c'} } };
22
23    printf ("Month    Number of Days\n");
24    printf ("-----\n");
```

```

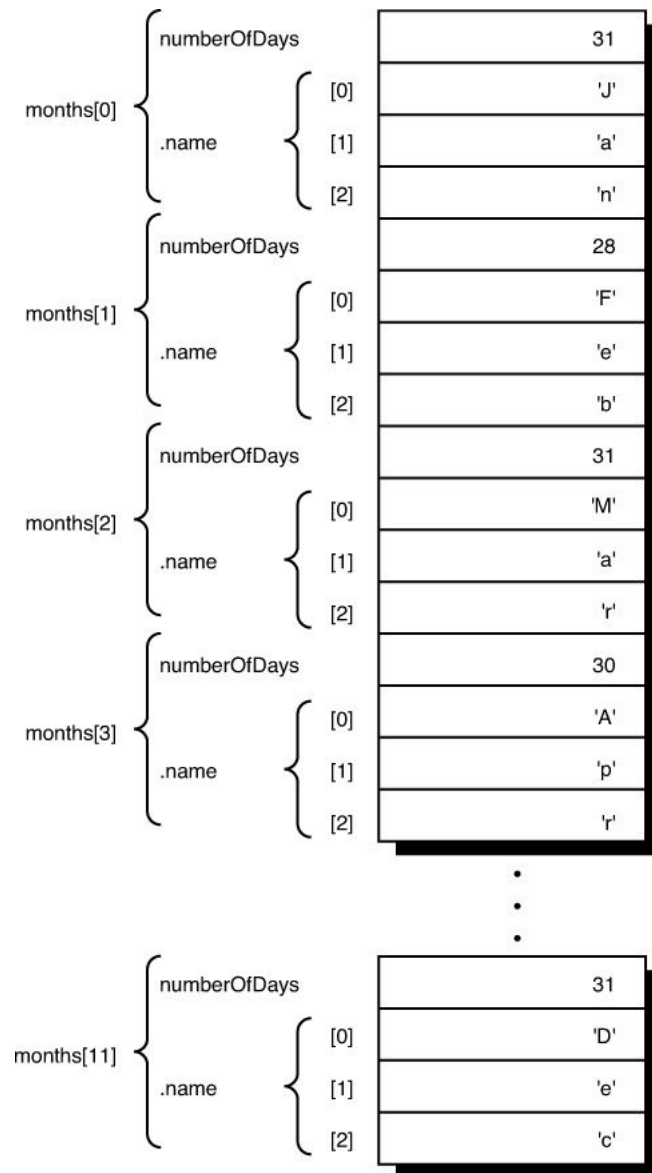
25
26     for ( i = 0; i < 12; ++i )
27         printf ( " %c%c%c          %i\n",
28                 months[i].name[0], months[i].name[1],
29                 months[i].name[2], months[i].numberOfDays);
30
31     return 0;
32 }

```

Программа выдаст:

Month	Number of Days
Jan	31
Feb	28
Mar	31
Apr	30
May	31
Jun	30
Jul	31
Aug	31
Sep	30
Oct	31
Nov	30
Dec	31

В памяти программы созданная нами структура выглядит следующим образом:



Как видно из рисунка, нотация `months[0]` обращается ко всем элементам структуры `month`, находящейся в первом элементе массива `months`. Таким образом, если мы вдруг будем передавать `months[0]` в функцию в качестве аргумента, то при объявлении функции нужно не забыть указать, чтоб она в качестве параметра принимала `struct month`.

Обращение к элементу структуры в первом элементе массива выглядит следующим образом:

```
months[0].numberOfDays
```

Этот элемент имеет тип `int`, об этом нужно помнить при объявлении функций и составлении выражений.

Можно даже обратиться к элементу массива внутри структуры:

```
months[0].name[0]
```

Здесь мы обратимся к первому элементу массива `name`, находящемуся в структуре, содержащейся в первом элементе массива `months`. То есть, символу `'J'`. Жесть...

Дополнительные свойства структур

Разберем дополнительные свойства структур.

Во-первых, можно задавать переменные типа структур сразу после объявления самих структур:

```
struct date
{
    int month;
```

```

    int day;
    int year;
} todaysDate, purchaseDate;

```

Здесь мы задали переменные todaysDate и purchaseDate типа date.

Можно также сразу на месте инициализировать переменные:

```

struct date
{
    int month;
    int day;
    int year;
} todaysDate = { 1, 11, 2005 };

```

Можно и массивы объявлять:

```

struct
{
    int month;
    int day;
    int year;
} dates[100];

```

Обратите внимание, что здесь мы не объявили название структуры. Так можно, но чтоб создавать другие структуры такого же типа, нужно будет заново объявлять структуру (читай: переписывать все 5 строк заново).

Упражнения

2. Write a program that permits the user to type in two dates and then calculates the number of elapsed days between the two dates.

```

1 #include <stdio.h>
2 #include <stdlib.h> // for abs() function
3
4 struct date
5 {
6     int day;
7     int month;
8     int year;
9 };
10
11 int main (void)
12 {
13     long int n (struct date dateIn);
14
15     struct date date1, date2;
16     long int numberOfDays;
17
18     printf("Enter first date (dd.mm.yyyy): ");
19     scanf ("%i.%i.%i", &date1.day, &date1.month, &date1.year);
20     printf("Enter second date (dd.mm.yyyy): ");
21     scanf ("%i.%i.%i", &date2.day, &date2.month, &date2.year);
22     numberOfDays = abs(n(date1) - n(date2));
23
24     printf("\nNumber of elapsed days: %li", numberOfDays);
25
26     return 0;
27 }
28
29 // Function to calculate the n parameter
30
31 long int n (struct date dateIn)
32 {
33     int f, g;

```

```

34     long int n;
35
36     if (dateIn.month <= 2)
37         f = dateIn.year - 1;
38     else
39         f = dateIn.year;
40
41     if (dateIn.month <= 2)
42         g = dateIn.month + 13;
43     else
44         g = dateIn.month + 1;
45
46     n = (1461 * f / 4) + (153 * g / 5) + dateIn.day;
47
48     return n;
49 }

```

В задании ошибка, вместо 8 авг 2004 и 22 фев 2005 авторы приводят пример для 3 авг 2004 и 21 фев 2005. Но это не помешало мне допустить ошибку с переполнением переменной. В примере показано, что возможные значения n могут выходить за максимальный предел переменной типа int, нужно внимательно за этим следить.

3. Write a function `elapsed_time` that takes as its arguments two time structures and returns a time structure that represents the elapsed time (in hours, minutes, and seconds) between the two times.

```

1 struct time elTime (struct time time1, struct time time2)
2 {
3     struct time result;
4     int totalSeconds1, totalSeconds2, n;
5     static int secsInDay = 86400;
6
7     totalSeconds1 = (time1.hrs * 3600) + (time1.mins * 60) + time1.secs;
8     totalSeconds2 = (time2.hrs * 3600) + (time2.mins * 60) + time2.secs;
9
10    if ( (totalSeconds2 - totalSeconds1) < 0 )
11        n = secsInDay - totalSeconds2 + totalSeconds1;
12    else if (totalSeconds1 == 0)
13        n = totalSeconds2;
14    else
15        n = totalSeconds2 - totalSeconds1;
16
17    result.hrs = n / 3600;
18    n -= 3600 * (n / 3600); // exploit the remainder cutoff on integer division
19    result.mins = n / 60;
20    n -= 60 * (n / 60);
21    result.secs = n;
22
23    return result;
24 }

```

Пришлось попотеть, в основном с математикой. Пока тяжело себе представлять в голове даже такие простые вещи.

4. Use the functions developed in the previous exercise to develop a program that displays the day of the week on which a particular date falls. Make certain that the program displays the day of the week in English (such as "Monday").

```

1 /*
2 4. Develop a program that displays the day of the week
3 on which a particular date falls.
4 */
5
6 #include <stdio.h>
7

```

```

8 struct date
9 {
10     int day;
11     int month;
12     int year;
13 };
14
15 int main (void)
16 {
17     static char days [7][3] = {'M','o','n'},
18                               {'T','u','e'},
19                               {'W','e','d'},
20                               {'T','h','u'},
21                               {'F','r','i'},
22                               {'S','a','t'},
23                               {'S','u','n'}};
24
25     long int currentDay (struct date dateIn);
26
27     struct date date1;
28     int i;
29
30     printf("Enter date (dd.mm.yyyy): ");
31     scanf ("%i.%i.%i", &date1.day, &date1.month, &date1.year);
32
33     i = currentDay(date1) - 1; // because array index starts from 0
34     printf("\n%c%c%c\n", days[i][0], days[i][1], days[i][2]);
35
36     return 0;
37 }
38
39 // Function to calculate the day of the week
40
41 long int currentDay (struct date dateIn)
42 {
43     int f, g;
44     long int n, result;
45
46     if (dateIn.month <= 2)
47         f = dateIn.year - 1;
48     else
49         f = dateIn.year;
50
51     if (dateIn.month <= 2)
52         g = dateIn.month + 13;
53     else
54         g = dateIn.month + 1;
55
56     n = (1461 * f / 4) + (153 * g / 5) + dateIn.day;
57     result = (n - 621049) % 7;
58
59     return result;
60 }

```

Ничего сложного, но не уследил за тем, что индекс массива начинается с 0, а не с 1.

5. Write a function called `clockKeeper()` that takes as its argument a `dateAndTime` structure as defined in this chapter. The function should call the `timeUpdate()` function, and if the time reaches midnight, the function should call the `dateUpdate` function to switch over to the next day. Have the function return the updated `dateAndTime` structure.

```

1 struct dateAndTime clockKeeper (struct dateAndTime now)
2 {
3     now.sdate = dateUpdate(now.sdate);

```



```
4     now.stime = timeUpdate(now.stime);
5
6     return now;
7 }
```

Странное задание, слишком простое.

6. *Replace the dateUpdate() function with the modified one that uses compound literals as presented in the text. Run the program to verify its proper operation.*

```
1 struct date dateUpdate (struct date today)
2 {
3     struct date tomorrow;
4     int numberOfDays (struct date d);
5
6     if ( today.day != numberOfDays (today) )
7         tomorrow = (struct date) {today.day + 1, today.month, today.year};
8
9     else if ( today.month == 12 )        // end of year
10        tomorrow = (struct date) {1, 1, today.year + 1};
11
12    else                                  // end of month
13        tomorrow = (struct date) {1, today.month + 1, today.year};
14
15    return tomorrow;
16 }
```

Толку с этого задания, если оно разбиралось в уроке.

Источники:

1. Stephen Kochan – Programming in C (4th Edition); chapter 8