Функция возвращает структуру

FS4. Функция вычисления суммы времен.

Часы имеют 12-ти часовой циферблат. Человек вышел из дома когда часы показывали hour часов, min минут, sec секунд. Он вернулся через dhour часов (0 \le dhour \le 120≤dhour≤12), dmin минут, dsec.

Требуется вычислить показания часов когда человек вернулся.

Решение (1)

#include <stdio.h>

#include <stdlib.h>

// Допустим описана подобная структура

typedef struct Time

{

unsigned char hour; // часы

unsigned char min; // минуты

unsigned char sec; // секунды

} Timer;

// функция принимает объект struct Time

// вместо struct Time можно использовать

// синоним Timer

// Передаем указатель на объект типа Timer

// для передачи значений

void timePlus(Timer a, struct Time b, Timer \* tmp){

// Чтобы функция могла вернуть какой-либо объект,

// он должен быть в ее распоряжении

int sec, min, h;

sec = a.sec + b.sec;

min = sec / 60 + a.min + b.min;

h = min / 60 + a.hour + b.hour;

// заполнение атрибутов переданного объекта

tmp->sec = sec % 60;

tmp->min = min % 60;

tmp->hour = h % 12;

};

// функция печати времени в формате hh:mm:ss

void printTime(Timer a)

{

printf("%02hhu:%02hhu:%02hhu\n", a.hour, a.min, a.sec);

};

int main(){

Timer t1, t2; // объект дата

Timer res;

// получение значений атрибутов с консоли

// ввод в формате hh:mm:ss

scanf("%hhu:%hhu:%hhu", &(t1.hour), &(t1.min), &(t1.sec));

scanf("%hhu:%hhu:%hhu", &(t2.hour), &(t2.min), &(t2.sec));

// вызов функции timePlus(), c передачей

// указателя на res

timePlus(t1, t2, &res);

// вызов функции печати

printTime(res);

return 0;

}

struct Time и Timer используется в примере одновременно в аргументах функции. Чтобы показать, что они равноправны с точки зрения компилятора. Но не делайте так в настоящем коде! Только Timer.

Решение (2)

#include <stdio.h>

#include <stdlib.h>

// Допустим описана подобная структура

typedef struct Time

{

unsigned char hour; // часы

unsigned char min; // минуты

unsigned char sec; // секунды

} Timer;

// функция печати времени в формате hh:mm:ss

void printTime(Timer a)

{

printf("%02hhu:%02hhu:%02hhu\n", a.hour, a.min, a.sec);

};

// функция принимает объект struct Termin

// вместо struct Termin можно использовать

// синоним Date

Timer timePlus(Timer a, struct Time b){

// Чтобы функция могла вернуть какой-либо объект,

// он должен быть в ее распоряжении

// Создадим временный объект tmp типа Timer,

// в который запишем результаты вычислений

// после окончания работы функции все локальные

// переменные, в том числе и tmp прекратят свое

// существование

Timer tmp;

int sec, min, h;

sec = a.sec + b.sec;

min = sec / 60 + a.min + b.min;

h = min / 60 + a.hour + b.hour;

tmp.sec = sec % 60;

tmp.min = min % 60;

tmp.hour = h % 12;

// возврат объекта типа Timer

return tmp;

};

int main(){

Timer t1, t2; // объект дата

Timer res;

// получение значений атрибутов с консоли

// ввод в формате hh:mm:ss

scanf("%hhu:%hhu:%hhu", &(t1.hour), &(t1.min), &(t1.sec));

scanf("%hhu:%hhu:%hhu", &(t2.hour), &(t2.min), &(t2.sec));

// вызов функции timePlus() и получение копии

// возвращаемого значения

res = timePlus(t1, t2);

// вызов функции печати

printTime(res);

return 0;

}

SF5 Разность показания часов.

Часы имеют 24-часовой циферблат. Два события произошли в течение одних суток (не ранее 0 часов и не позднее 24) и зафиксированы показания часов: первого события и второго.

Требуется вычислить сколько времени прошло между событиями с точностью до секунд.

Решение

#include <stdio.h>

#include <stdlib.h>

// Допустим описана подобная структура

typedef struct Time

{

unsigned char hour; // часы

unsigned char min; // минуты

unsigned char sec; // секунды

} Timer;

// функция печати времени в формате hh:mm:ss

void printTime(Timer a)

{

printf("%02hhu:%02hhu:%02hhu\n", a.hour, a.min, a.sec);

};

// функция вычисления разницы между показаниями

// предполагаем, что первый параметр - более позднее событие,

// а второй параметр - более раннее

Timer timeMinus(Timer a, struct Time b){

// Чтобы функция могла вернуть какой-либо объект,

// он должен быть в ее распоряжении

// Создадим временный объект tmp типа Timer,

// в который запишем результаты вычислений

// после окончания работы функции все локальные

// переменные, в том числе и tmp прекратят свое

// существование

Timer tmp;

unsigned char sec, min, h;

h = a.hour - b.hour - 1;

min = 60 + a.min - b.min - 1;

sec = 60 + a.sec - b.sec;

tmp.sec = sec % 60;

tmp.min = (min + sec / 60 ) % 60;

tmp.hour = (h + min / 60 ) % 24;

// возврат объекта типа Timer

return tmp;

};

int main(){

Timer t1, t2; // объект дата

Timer res;

// получение значений атрибутов с консоли

// ввод в формате hh:mm:ss

scanf("%hhu:%hhu:%hhu", &(t1.hour), &(t1.min), &(t1.sec));

scanf("%hhu:%hhu:%hhu", &(t2.hour), &(t2.min), &(t2.sec));

// вызов функции timePlus() и получение копии

// возвращаемого значения

res = timeMinus(t1, t2);

// вызов функции печати

printTime(res);

return 0;

}

**Математическое отступление**

Допустим наши числа могут принимать некоторое максимальное (по модулю) значение max*max*, то есть, в нашем представлении, max + 1 = 0*max*+1=0. Обозначим z = 0*z*=0, если число number*number* у нас положительное и z = 1*z*=1 , если - отрицательное. Тогда: kode = max - number*kode*=*max*−*number* и z=1*z*=1 - число number<0*number*<0, представленное в дополнительном коде.

Например, если max = 1000*max*=1000, то дополнительный код для -17−17: kode=1000 - 17 = 9983, z=1*kode*=1000−17=9983,*z*=1 Имеется два числа a*a* и b*b* и неизвестно какое из них больше. Разность чисел a*a* и b*b* можено представить так:diff = a - b = a + (max - b) - max*diff*=*a*−*b*=*a*+(*max*−*b*)−*max*илиdiff= (a+kode)-max, z=1*diff*=(*a*+*kode*)−*max*,*z*=1

Рассмотрим сумму trans=a+kode*trans*=*a*+*kode*. - trans>max*trans*>*max*, тогда res=trans-max*res*=*trans*−*max* - получаем положительное число, в нормальном виде и z=0*z*=0. - trans<max*trans*<*max*, тогда res=trans-max=-(max-trans)*res*=*trans*−*max*=−(*max*−*trans*) - это отрицательное число в дополнительном коде и z=1*z*=1. Здесь число необходимо представить в нормальном виде

Необходимо научиться представлять число в дополнительном коде и производить с ним действия.

Рассмотрим простые пример для десятичных чисел, не превышающих 1000010000.

**Пример 1** a=287,\;b=170*a*=287,*b*=170 a-b=a+(-b)*a*−*b*=*a*+(−*b*) -b−*b* представляем в дополнительном коде. 10000-170=983010000−170=9830 Замечаем, что эту операцию удобно выполнять поразрядно. Но, младшие нули остаются нулями, а первый ненулевой разряд дополнительного кода - это дополнение до 1010: 10-710−7, а все остальные после первого ненулевого разряда - дополнение до 99: 9-19−1, 9-09−0 и т.д.

Таким образом для решения зади необходимо написать функцию преобразования числа в дополнительный код.

Представим десятичное число (-10000<number<10000−10000<*number*<10000) поразрядно и отдельно представим знак числа.

#include <stdio.h>

#include <stdlib.h>

typedef struct Num

{

char a0; // младший разряд числа

char a1; // второй разряд

char a2; // третий разряд

char a3; // четвертый разряд

char sign;

} Number;

// Функция представления десятичного числа в обратном коде

Number addCode(Number d){

Number tmp; // переменная для возврата значения

/\*====== Преобразование разрядов ==========================

для вычисления значения дополнительного разряда учитываем что

первый ненулевой младший разряд дополнительного числа вычисляется

как 10 - a0,

а все остальные после первого ненулевого вычисляются как

9 - a0

p "отвечает" за 10 - a0

z "отвечает" за 9 - a0

=========================================================== \*/

char z = 0; // самый младший разряд вычисляется как

char p = 1; // 10 - a0

//обратный код для младшего разряда

tmp.a0 = (10 - d.a0) % 10;;

// Если tmp.a0 > 0, то для всех последующих операций

// нужно сделать p=0

// !!tmp.a0 будет 1 если tmp.a0 > 0 и будет 0, если tmp.a0 = 0

// если !!tmp.a0 = 1, то (p + !!tmp.a0) % 2 будет 0

// если tmp.a0 = 0, то (p + !!tmp.a0) % 2 будет 1

// То есть значение всего выражения зависит от (p + !!tmp.a0) % 2

p = p \* ( (p + !!tmp.a0) % 2);

// z становится 1 как только p будет 0

z = !p;

printf("a0: %hhd p: %hhd z: %hhd\n", tmp.a0, p, z);

tmp.a1 = z \* ( 9 - d.a1) + (p \* ( 10 - d.a1)) % 10;

// После того как один раз p стало 0, значение выражения p \* ( (p + !!tmp.a1) % 2)

// больше не зависит от (p + !!tmp.a1) % 2, оно будет 0

p = p \* ( (p + !!tmp.a1) % 2);

// с этой поры z всегда 1

z = !p;

printf("a1: %hhd p: %hhd z: %hhd\n", tmp.a1, p, z);

tmp.a2 = z \* ( 9 - d.a2) + (p \* ( 10 - d.a2)) % 10;

p = p \* ( (p + !!tmp.a2) % 2);

z = !p;

printf("a2: %hhd p: %hhd z: %hhd\n", tmp.a2, p, z);

tmp.a3 = z \* ( 9 - d.a3) + (p \* ( 10 - d.a3)) % 10;

p = p \* ( (p + !!tmp.a2) % 2);

z = !p;

printf("a3: %hhd p: %hhd z: %hhd\n", tmp.a3, p, z);

tmp.sign = d.sign;

return tmp;

};

void prNum(Number tmp)

{

printf("%hhd %hhd %hhd %hhd %hhd \n", tmp.a3, tmp.a2, tmp.a1, tmp.a0, tmp.sign);

};

int main()

{

Number nb={0,0,5,1,1}, nz;

nz=addCode(nb);

prNum(nz);

}

Конечно, для работы с простыми числами проще было бы воспользоваться обычными средствами. Но часто приходится выполнять "обычные" математические функции для сложных величин, которые записаны в неоднородных системах счисления. Например, даты.

## Примечание о работе с числами в неоднородных системах счисления

Гораздо проще формулы для сложения или вычитания часов, минут и секунд получаются, если написать функции преобразования времени в виде Timer в количество секунд с 0:00 (назовем это usec) и обратно. Но они будут неверно срабатывать, если from > to. Подумайте, как можно изменить код, чтобы он работал для задачи "Электричка выехала во время from и приехала во время to, сколько времени была в пути электричка?" Электричка может выехать в 23:40:00 и приехать в 0:15:00, в пути 0:35:00.

int time2usec(Timer t) {

int res = t.hour \* 24 + t.min; // minutes since 0:00

res = res \* 60 + t.sec;

return res;

}

Timer usec2time(int sec) {

Timer t;

t.sec = sec % 60;

sec = sec / 60; // осталось минут

t.min = sec % 60;

sec = sec / 60; // осталось часов

t.hour = sec % 24;

return t;

}

Timer timeMinus(Timer from, Timer to){

Timer tmp;

int sec\_from = time2usec(from);

int sec\_to = time2usec(to);

tmp = usec2time(sec\_to - sec\_from);

// возврат объекта типа Timer

return tmp;

};