Московский государственный университет им. М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Алгоритмы и алгоритмические языки
Лекция 8

2 октября 2019 г.

Программа: количество дней между двумя датами

```
int main (void)
{
  while (1) {
    int m1, d1, y1, m2, d2, y2;
    int t1, t2;
    int days1, days2, total;

    if (scanf ("%d%d%d%d%d%d", &d1, &m1, &y1,
               &d2, &m2, &y2) != 6)
      break;
    t1 = check_date (d1, m1, y1);
    if (t1 == 1 || (t2 = check_date (d2, m2, y2)) == 1)
      break;
    else if (t1 == 2 || t2 == 2)
      continue;
```

Программа: количество дней между двумя датами

```
    <...>
    days1 = days_from_jan1 (d1, m1, y1);
    days2 = days_from_jan1 (d2, m2, y2);
    total = days_between_years (y1, y2)
            + (days2 - days1);
    printf ("Days␣between␣dates:␣%d,"
            "weeks␣between␣days:␣%d\n",
            total, total / 7);
  }
  return 0;
}
```

Программа: количество дней между двумя датами

```
#include <stdio.h>

static int check_date (int d, int m, int y)
{
  if (!d || !m || !y)
    return 1;
  if (d < 0 || m < 0 || y < 0)
    {
      printf ("%d␣%d␣%d:␣wrong␣date\n", d, m, y);
      return 2;
    }
  return 0;
}
```

Программа: количество дней между двумя датами

```
static int leap_year (int y) {
  return (y % 400 == 0) || (y % 4 == 0 && y % 100 != 0);
}

static int days_in_year (int y) {
  return leap_year (y) ? 366 : 365;
}

static int days_between_years (int y1, int y2) {
  int i;
  int days = 0;

  for (i = y1; i < y2; i++)
    days += days_in_year (i);
  return days;
}
```

Программа: количество дней между двумя датами

```
static int days_from_jan1 (int d, int m, int y) {
  int days = 0;
  switch (m) {

      case 12: days += 30;
      case 11: days += 31;
      case 10: days += 30;
      case 9: days += 31;
      case 8: days += 31;
      case 7: days += 30;
      case 6: days += 31;
      case 5: days += 30;
      case 4: days += 31;
      case 3: days += leap_year (y) ? 29 : 28;
      case 2: days += 31;
      case 1: break;

    }
    return days + d;
}
```

Символьный тип данных (char)

Программа подсчета числа строк во входном потоке

```
#include <stdio.h>
int main (void)
{
  int c, nl = 0;
  while ((c = getchar()) != EOF)
    if (c == '\n')
      ++nl;
  printf ("%d\n", nl);
  return 0;
}
```

Каков должен быть возвращаемый тип функции `getchar`?

Символьный тип данных (`char`)

Символьные данные представляются в некотором коде. Популярным кодом является ASCII (American Standard Code for Information Interchange).

Каждому символу сопоставляется его код — число типа `char`

Требуется, чтобы в кодировке присутствовали маленькие и большие английские буквы, цифры, некоторые другие символы

Требуется, чтобы коды цифр `0`, `1`, ..., `9` были последовательны

Символьный тип данных (`char`)

Символьные данные представляются в некотором коде. Популярным кодом является ASCII (American Standard Code for Information Interchange).

К символьным данным применимы операции целочисленных типов (но обычно — операции *отношения* и *сравнения*)

Каждый символ-литерал заключается в одинарные кавычки `'` и `'`

Последовательность символов (строка) заключается в двойные кавычки `"` и `"`

Специальные (управляющие) символы представляются последовательностями из двух символов. Примеры:

- `\n` переход на начало новой строки
- `\t` знак табуляции
- `\b` возврат на один символ с затиранием

Таблица ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	\0									\t	\n					
1												ESC				
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}		DEL

В коде ASCII буквы верхнего и нижнего регистра составляют непрерывные последовательности: между `a` и `z` (соответственно, между `A` и `Z`) нет ничего, кроме букв, расположенных в алфавитном порядке.

Это же верно и для цифр `0`, `1`, ..., `9`.

Преобразование строки цифр в целое число

```
int atoi (char s[]) {
  int i, n;

  n = 0;
  for (i = 0; s[i] >= '0' && s[i] <= '9'; ++i)
    n = 10 * n + (s[i] - '0');
  return n;
}
```

Верно для *любой* кодировки символов.

Массивы

Массивы позволяют организовывать непрерывные последовательности нескольких однотипных элементов и обращаться к ним по номеру (индексу).

- Элементы массивов располагаются в памяти последовательно и индексируются с 0:

```
int a[30];  /* элементы a[0], a[1], ... , a[29] */
```

- Все массивы — одномерные, но элементом массива может быть массив:

```
int b[3][3];  /*  b[0][0], b[0][1], b[0][2],
                  b[1][0], b[1][1], b[1][2],
                  b[2][0], b[2][1], b[2][2] */
```

- Контроль правильности индекса массива **не производится**!

Пример программы с массивом символов

```
#include <stdio.h>
int main (void) {
  int c, i, nwhite = 0, nother = 0, ndigit[10];
  for (i = 0; i < 10; ++i)
    ndigit[i] = 0;
  while (c = getchar ()) != EOF)
    if (c >= '0' && c <= '9')
      ++ndigit[c - '0'];
    else if (c == '␣' || c == '\n' || c == '\t')
      ++nwhite;
    else
      ++nother;
```

Пример программы с массивом символов

```
  printf ("digits=");
  for (i = 0; i < 10; ++i)
    printf (" %d", ndigit[i]);
  printf (", white space=%d, other=%d\n",
          nwhite, nother);
  return 0;
}
```

Инициализация массивов

```
type name[dim1]...[dimN] = {value list};
```

Можно не указывать размер массива — он будет вычислен по количеству элементов инициализатора

```
int sqrs[] = {1, 4, 9, 16, 25}; /* 5 элементов */
```

C99: инициализация лишь некоторых элементов (остальные инициализируются нулями)

```
int days[12] = {31, 28, [4] = 31,30,31, [1] = 29};
```

- При инициализации одного элемента дважды используется последнее значение
- После задания номера элемента дальнейшие инициализаторы присваиваются следующим по порядку элементам

Можно использовать модификаторы `const`, `static` и т.п.

Можно использовать любое *константное целочисленное выражение* для определения размера массива

- `const`-переменная не является константным выражением!

Строки

- Строка — это одномерный массив типа `char`.

  Объявляя массив, предназначенный для хранения строки, необходимо предусмотреть место для символа `'\0'` (конец строки).
- Строковая константа записывается как `"string constant"`.

  В конец строковой константы компилятор добавляет `'\0'`.
- Стандартная библиотека функций работы со строками `<string.h>`, в частности, содержит такие функции, как:

`strcpy(s1, s2)`	копирование `s2` в `s1`
`strcat(s1, s2)`	конкатенация `s2` и `s1`
`strlen(s)`	длина строки `s`
`strcmp(s1, s2)`	сравнение `s2` и `s1` в *лексикографическом порядке*: 0, если `s1` и `s2` совпадают, отрицательное значение, если `s1 < s2`, положительное значение, если `s1 > s2`
`strchr(s, ch)`	указатель на первое вхождение символа `ch` в `s`
`strstr(s1, s2)`	указатель на первое вхождение подстроки `s2` в строку `s1`

Дома. Прочитайте о функциях `strspn`, `strpbrk`.

Зачем нужна функция `stpcpy`?