Курс «Алгоритмы и алгоритмические языки»
1 семестр 2016/2017

Лекция 6

Старшинство операций

Операции	Ассоциативность
`! ++ -- + - sizeof (type)`	Справа налево
`* / %`	Слева направо
`+ -`	Слева направо
`== !=`	Слева направо
`&&`	Слева направо
`||`	Слева направо
`= += -= *= /= %=`	Справа налево
`,`	Слева направо

Операторы

- ***Выражение-оператор***: **`expression;`**
- ***Составной оператор***: **`{}`**
- ***Условный оператор***: **`if (expr) stmt; else stmt;`**
  - **`else`** всегда относится к ближайшему **`if`**:

```
if (x > 2)                    if (x > 2) {
  if (y > z)                    if (y > z)
    y = z;                        y = z;
  else                          }
    z = y;                      else
                                  z = y;
```

- ***Оператор выбора***:

```
switch (expr) {
        case const-expr: stmt;
        case const-expr: stmt;
        default: stmt;
}
```

  - Оператор **`break`** – немедленный выход из **`switch`**.

Операторы

- ***Цикл while*: `while (expression) stmt;`**
- ***Цикл for*:**

  `for (decl1; expr2; expr3)`
  `  stmt;`

  `decl1 -` возможно определение переменной с инициализатором

  ```
  decl1;
  while (expr2) {
    stmt;
    expr3;
  }
  ```

  - **`for ( ; ; ) stmt;`** – бесконечный цикл.
- ***Цикл do-while*: `do { stmt; } while (expression);`**
  - Проверка условия выхода из цикла после выполнения тела.
- ***Операторы break и continue*:** выход из внутреннего цикла и переход на следующую итерацию
- ***Оператор goto*:** переход по метке

  `goto label`
  …
  `label:`
  - Областью видимости метки является вся функция

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
- 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

Символьный тип данных (char)

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	

Символьный тип данных (char)

- 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 символов цифр **s** в целое число
  (верно для любой кодировки символов)

```
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;
}
```

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]; /* элементы 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***. Программа, подсчитывающая количество вхождений в строку (текст) каждой из десяти цифр (**`ndigit[10]`**), пробельных символов (**`nwhite`**) и остальных символов (**`nother`**).

Массивы

```
#include <stdio.h>
int main (void)
{
  int c, i, nwhite, nother, ndigit[10];
  nwhite = nother = 0;
  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;
  printf ("digits=");
  for (i = 0; i < 10; ++i)
    printf (" %d", ndigit[i]);
  printf (", white space=%d, other=%d\n”, nwhite, nother);
  return 0;
}
```