Переменные, ввод/вывод переменных, операции (начало)

Структура простой программы

`#include <stdio.h>`
 	
`int main(void)`
`{`
`  puts("Hello, world!");`
 	
`  return 0;`
`}`
	`директивы`
	
	`int main(void)  // функция`
	`{`
	`  операторы`
	`}`

Даже в простой программе можно выделить черты, которые будут присущи любой программе на Си:

- директивы;
- функции;
- операторы.

Переменные

Переменная в программе представляет собой абстракцию ячейки памяти компьютера или совокупности таких ячеек.

Переменную можно охарактеризовать следующим набором атрибутов:

- имя (идентификатор);
- тип;
- адрес;
- значение;
- область видимости;
- время жизни.

Атрибуты переменной (1)

- *Имя* (идентификатор) – это строка символов, используемая для идентификации некоторой сущности в программе (переменной, функции и т. п.).

- *Тип* переменной определяет как переменная хранится, какие значения может принимать и какие операции могут быть выполнены над переменной.

- *Адрес* переменной – это ячейка памяти, с которой связана данная переменная. Адрес переменной иногда называют ее *левым значением* (*l-value*).

Атрибуты переменной (2)

- *Значение* переменной – это содержимое ячейки или ячеек памяти, связанных с данной переменной. Значение переменной иногда называется ее *правым значением* (*r-value*).

- *Область видимости* переменной – это часть текста программы, в пределах которой переменная может быть использована.

- *Время жизни* переменной – это интервал времени выполнения программы, в течение которого переменная существует (т.е. ей выделена память).

Имена в Си (1)

- Имя может содержать буквы, цифры и символ подчеркивания.
- Имя обязательно должно начинаться или с буквы, или с символа подчеркивания.
- Имя не должно совпадать с ключевыми словами языка.

- Имя чувствительно к регистру символов.
- Длина имени практически не ограничена.

Имена в Си (2)

Правильные имена	**Неправильные имена**
times10 get_next_char _done (!) puts (!)	10times get-next-char while

Согласно стандарту (7.1.3) не рекомендуется использовать имена

- которые начинаются с подчеркивания;
- которые совпадают с именами из стандартных заголовочных файлов, если подключен соответствующий заголовочный файл;
- ...

Типы в Си

В языке Си существует множество типов. Пока ограничимся только двумя числовыми типами: *int* и *float*.

Переменные типа int могут хранить целые числа (-25, 0, 157). Диапазон чисел ограничен (скорее всего от -2147483648 до 2147483647).

Переменные типа float могут хранить вещественные числа, гораздо большего значения чем числа, описываемые типом int. Кроме того, они могут хранить числа с дробной частью (-0.723, 265.12). НО арифметические операции для таких переменных выполняются медленнее, а сами числа представлены приближенно.

Определение переменных

До того, как переменная будет использована в программе, она должна быть *определена*. Чтобы определить переменную, необходимо сначала указать ее тип, а затем имя.

```
int width;
float speed;
```

Если несколько переменных имеют один и тот же тип, их определения можно объединять

```
int width, length, area;
float speed, acceleration;
```

Присваивание (1)

Переменной может быть «назначено» значение с помощью присваивания.

```
width = 5;
length = 4;
speed = 25.34;    // speed = 25.34f;
```

Целым переменным обычно присваивают целые значения, вещественным – вещественные. Смешение типов возможно, но не всегда безопасно.

```
width = 17.5;
speed = 100;
```

Присваивание (2)

После того как переменной присвоено значение, она может использоваться для вычисления значения другой переменной.

```
width = 5;
length = 4;
area = width * length;
```

Определение переменной можно совместить с присваиванием ей начального значения.

```
int width = 5;
int length = 4;
int area = width * length;
```

Структура простой программы с учетом определения переменных

```
директивы

int main(void)   // функция
{
    определения
    операторы
}
```

- Стандарт C89 требует, чтобы определения переменных располагались в начале функций.
- Стандарт C99 снимет это ограничение и позволяет смешивать определения переменных и операторов. Единственное ограничение – переменная должна быть описана перед своим использованием.

Вывод значения переменной

Для вывода текущего значения переменной используется (обычно) функция printf.

```
#include <stdio.h>

int main(void)
{
    int i = 157;
    float f = 3.14;

    // Первый аргумент – строка-форматирования. Она может
    // содержать спецификаторы (%d, %f) и esc-последовательности
    // (\n).
    printf("My favorite numbers are %d and %f\n", i, f);

    return 0;
}
```

Спецификаторы

В строке форматирования спецификатор обозначает место, в которое будет выведено соответствующее значение во время отображения строки.

Спецификаторы начинаются с символа “%”, а заканчиваются «символом преобразований», который определяет значение какого типа будет отображаться на экране.

Спецификатор	**Отображение аргумента**
%d	Целое число (100)
%f	Вещественное число (7.123000)
%e	Вещественной число в экспоненциальной форме (7.123000e+000)
%g	Что короче из %f или %e (7.123).

Esc-последовательности

Предназначены для отображения непечатных или «трудно» выводимых символов. Обычно эти символы управляют расположением текста на экране.

Символ	**Esc-последовательность**
beep	\a
Новая строка	\n
Гор. табуляция	\t
Возврат каретки	\r
Обратный слеш	\\
Двойная кавычка	\”

Ошибки при использовании printf

```
int i = 5;
float f = 2.5;

// Переменных меньше, чем спецификаторов
printf("%d %d\n", i);        // 5 2293664

// Переменных больше, чем спецификаторов (не критично)
printf("%d\n", i, i);        // 5

// Спецификаторы не соответствуют типам переменных
printf("%d %f\n", f, i);     // 0 0.000000
```

Спецификаторы (1)

В общем виде спецификатор записывается следующим образом:

`%[флаги][ширина][.точность][размер]тип`

Квадратные скобки означают, что соответствующий элемент может отсутствовать.

Пока мы ограничимся только такими спецификаторами

- **`%[ширина][.точность]тип`**
- **`%-[ширина][.точность]тип`**

Спецификаторы (2)

Параметр «ширина» задает минимальную ширину поля вывода.

```
printf("%4d", 123);    // •123
printf("%3d", 12345);  // 12345
printf("%-4d", 123);   // 123•
```

Параметр «точность» задает точность. Его назначение зависит от параметра «тип».

```
int i = 40;
printf("|%d|%5d|%-5d|%5.3d|\n", i, i, i, i);
// |40|•••40|40•••|••040|

float f = 839.21f;
printf("|%10.3f|%10.3e|%-10g|\n", f, f, f);
// |•••839.210|•8.392e+02|838.21••••|
```

Ввод значения переменной

Для вывода текущего значения переменной используется (обычно) функция scanf.

```
#include <stdio.h>

int main(void)
{
  int i;
  float f;

  printf("What are your favorite numbers?\n");
  printf("(Enter an integer and a float.)\n");

  // NB: символ & перед именем переменной!
  scanf("%d%f", &i, &f);

  printf("\nYour favorite numbers are %d and %f!\n", i, f);

  return 0;
}
```

Алгоритм работы scanf (1)

Работа функции scanf управляется строкой форматирования.

Для каждого спецификатора scanf пытается выделить данные соответствующего типа во входных данных. Функция останавливается на символе, который не относится к очередному вводимому значению.

Если значение успешно прочитано, scanf продолжает обрабатывать строку. В противном случае - прекращает работу.

Алгоритм работы scanf (2)

```
int i, j;
float x, y;
scanf(“%d%d%f%f”, &i, &j, &x, &y);
```

Ввод пользователя

```
•1⇐
-20•••.3⇐
•••-4.0e3⇐
```

scanf «видит» его как последовательность символов

```
•1⇐-20•••.3⇐•••-4.0e3⇐
```

scanf обрабатывает последовательность следующим образом

```
•1⇐-20•••.3⇐•••-4.0e3⇐
```

Черные символы пропускаются, красные читаются.

Алгоритм работы scanf (3)

Первый символ, который не соответствует текущему считываемому значению, «возвращается» обратно для последующего анализа.

Ввод пользователя

```
1-20.3-4.0e3
```

В нашем случае (scanf(“%d%d%f%f”, &i, &j, &x, &y)) корректный ввод :)

Спецификаторы %e, %f, %g взаимозаменяемы, когда используются в функции scanf (всем троим соответствуют одним и тем же правилам распознавания вещественных значений).

Строка форматирования scanf

Строка форматирования может содержать как обычные символы, так и спецификаторы. В большинстве случаев строка форматирования содержит только спецификаторы.

```
// пробельный символ в строке
scanf("%d %d", &i, &x);

4SPACE5ENTER
4TAB5ENTER
4ENTER5ENTER
```

```
// обычный символ в строке
scanf("%d,%d", &i, &x);

4,5ENTER
4,TAB5ENTER
4ENTER,5ENTER   // ОШИБКА
4n5ENTER        // ОШИБКА
```

Ошибки при использовании scanf

- См. слайд 16 «Основные ошибки при использовании printf».
- Кроме того, часто забывают указывать символ & перед именем переменной.

```
int i, j;
scanf("%d%d", i, j);        // предупреждение компилятора
printf("%d%d\n", i, j);
```

Программа, скорее всего, работать не будет.

Пример

```
#include <stdio.h>

int main(void)
{
    float tf, tc;

    printf("Enter Celsius temperature: ");
    scanf("%f", &tc);

    tf = (9.0f / 5.0f) * tc + 32.0f;

    printf("Fahrenheit equivalent: %.1f\n", tf);

    return 0;
}
```

Пример (предыдущий)

```
#include <stdio.h>

#define SCALE_FACTOR    (9.0f / 5.0f)
#define FREEZING_PT     32.0f

int main(void)
{
    float tf, tc;

    printf("Enter Celsius temperature: ");
    scanf("%f", &tc);

    tf = SCALE_FACTOR * tc + FREEZING_PT;

    printf("Fahrenheit equivalent: %.1f\n", tf);


    return 0;
}
```

Выражения

Любой язык программирования включает в себя средства описания данных и средства описания действий.

Средства описания действий указывают действия, исполняемые программой. Простейшим средством описания действий является выражение.

Выражение задает действия, которые вычисляют единственное значение. Оно состоит из констант, переменных, а также знаков операций и скобок.

Элементы данных, к которым применяется операция, называют *операндами*.

Назначение операции и побочный эффект

- Каждая операция имеет операнды определенных типов и задает способ получения по значениям этих операндов нового значения определенного типа.

- Некоторые операции имеют *побочный эффект* (*side effect*). При выполнении этих операций, кроме основного эффекта - вычисления значения - происходят изменения объектов или файлов.

Классификация операций

В языке Си операции по способу записи подразделяются на

Постфиксные	`<операнд><операция>`	`p[i]`
Префиксные	`<операция><операнд>`	`&speed`
Инфиксные	`<операнд><операция><операнд>`	`width * length`

по количеству операндов на

Унарные	Один операнд	`-a`
Бинарные	Два операнда	`width * length`
Тернарные	Три операнда	`(a > b) ? a : b`

Приоритет и ассоциативность операций

- Правила приоритетов операций при вычислении выражений определяют порядок, в котором выполняются операции, имеющие разные приоритеты.

```
a + b * c   =>      a + (b * c)
```

- На вопрос о том, какой из операторов, имеющих равный приоритет, выполняется первым, отвечают правила ассоциативности. Операции одинакового приоритета либо левоассоциативны, либо правоассоциативны.

```
a - b + c   =>      ((a - b) + c)
```

Операция присваивания

После вычисления значения выражения нужно сохранить результат в переменной. Для этих целей используется операция присваивания.

v = e

Формально алгоритм работы операции присваивания в языке Си можно описать следующим образом:

- Вычислить l-значение первого операнда (v) операции.
- Вычислить r-значение второго операнда (e) операции.
- Присвоить вычисленное r-значение вычисленному l-значению объекта данных;
- Возвратить вычисленное r-значение как результат выполнения операции.

Операция присваивания: примеры

```
int i, j;

i = 5;              // Значение i равно 5
j = i;              // Значение j равно 5
k = 10 * i + j;     // Значение k равно 55
```

```
int i;
float f;

i = 72.99f;         // Значение i равно 72
f = 136;            // Значение f равно 136.0
```

Операция присваивания: особенности

Во многих языках программирования присваивание – это оператор. В языке Си присваивание – это операция такая же как, например, сложение.

Значением выражения a = b будет значение переменной b , приведенное к типу переменной a, и в качестве побочного эффекта это значение будет присвоено переменной a.

Благодаря тому, что присваивание - это операция, несколько присваиваний могут быть объединены в «цепочку»:

```
i = j = k = 0;
```

Операция присваивания: особенности

Операция присваивания правоассоциативна:

```
i = (j = (k = 0));
```

Присваивание в форме v = e разрешено везде, где допустимо использовать значение типа v. Например,

```
int i,j,k;

i = 1;
k = 1 + (j = i);
printf("%d %d %d\n", i, j, k);    // 1 1 2
```

Составное присваивание (1)

В операции присваивания «старое» значение переменной часто используется для вычисления «нового» значения этой же переменной:

```
i = i + 2;
```

Операции составного присваивания позволяют нам короче записать этот оператор:

```
i += 2;
```

Операция составного присваивания обладает теми же свойствами, что и обычная операция присваивания. Например, она обладает правой ассоциативностью:

```
i += j += k;   =>      i += (j += k);
```

Составное присваивание (2)

`v += e`	Прибавить e к v, сохранить результат в v
`v -= e`	Вычесть e из v, сохранить результат в v
`v *= e`	Умножить v на e, сохранить результат в v
`v /= e`	Разделить v на e, сохранить результат в v
`v %= e`	Вычислить остаток от деления v на e, сохранить результат в v

Составное присваивание (3)

Обратите внимание, что v += e не эквивалентно v = v + e.

- Причина 1: приоритете операции составного присваивания:

  i *= j + k НЕ i = i * j + k

  i *= j + k ЭТО i = i * (j + k)

- Причина 2: случай когда при вычислении v есть побочный эффект.

Составное присваивание

Операция	Название	Нотация	Класс	Приоритет	Ассоциат.
`=`	Присваивание	`x = y`	Инфиксные	2	Справа налево
`*=`	Присваивание с умножением	`x *= y`			
`/=`	Присваивание с делением	`x /= y`			
`%=`	Присваивание с остатком	`x %= y`			
`+=`	Присваивание со сложением	`x += y`			
`-=`	Присваивание с вычитанием	`x -= y`			