

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Р.Е.АЛЕКСЕЕВА»

Институт радиоэлектроники и информационных технологий

Кафедра «Информационные радиосистемы»

Основы программирования на Си

Методические указания к лабораторным работам
по дисциплине «Информационные технологии» для студентов
направления подготовки бакалавра 11.03.01 «Радиотехника»
дневной формы обучения

Нижний Новгород 2015

Составители: Е.Н.Приблудова, С.Б.Сидоров

УДК 621.325.5-181.4

Основы программирования на Си: метод. указания к лаб. работам по дисциплине «Информационные технологии» для студентов направления подготовки бакалавра 11.03.01 «Радиотехника» дневной формы обучения / НГТУ; сост.: Е.Н.Приблудова, С.Б.Сидоров. Н.Новгород, 2015, 19 с.

Изложены краткие сведения о синтаксисе основных элементов языка Си. Сформулирован порядок выполнения лабораторных работ. Приведены контрольные вопросы для самопроверки.

Подп. к печ. Формат 60x84 1/16. Бумага газетная.

Печать офсетная. Печ.л. 1,25. Уч.-изд.л. 0,75. Тираж . Заказ .

Нижегородский государственный технический университет им. Р.Е.Алексеева
Типография НГТУ. 603950, Н.Новгород, ул.Минина, 24.

© Нижегородский государственный технический
университет им. Р.Е.Алексеева, 2015

© Приблудова Е.Н., Сидоров С.Б., 2015

1. Цель работы

Изучение синтаксиса языка Си и приобретение базовых навыков по разработке простых программ на нем.

2. Краткие сведения

2.1. Базовые типы данных

Наиболее часто используемыми типами данных являются целочисленные типы данных, к которым относятся:

char - символьный тип (размер, как правило, 1 байт);

int - целое (размер: 2 или 4 байта);

Значения перечисленных выше целочисленных типов являются числами со знаком. Если же необходимо иметь переменную, значения которой всегда неотрицательны, то перед именем типа ставится модификатор типа ***unsigned***: ***unsigned char***, ***unsigned int***.

Язык Си обеспечивает типы данных, представляющие числа с плавающей точкой:

float - одинарной точности (размер, как правило, 4 байта);

диапазон чисел: $\pm 10^{38}$;

double - двойной точности (размер, как правило, 8 байт);

диапазон чисел: $\pm 10^{\pm 308}$;

В языке Си введен специальный тип данных ***void***. Этот тип данных определяет пустое множество значений. Его особенностью является невозможность объявления переменной этого типа данных.

2.2. Определение переменной

Синтаксис определения переменной:

спецификатор_типа имя_переменной;

Здесь *спецификатор_типа* должен быть одним из базовых типов данных.

Пример определения переменной типа *int*.

int x;

2.3. Организация ввода-вывода

Язык программирования Си не содержит встроенных средств ввода-вывода. Для этих целей используются специальные функции, входящие в состав стандартной библиотеки языка.

Спецификаторы форматного ввода или вывода для значений различных типов данных

%d - целое значение;

%e - числа с плавающей точкой в экспоненциальной форме;

%f - числа с плавающей точкой в десятичной форме;

%lf - числа с плавающей точкой двойной точности;

%g - числа с плавающей точкой в экспоненциальной или десятичной форме;

%c - символ типа *char*;

%x - целое число в шестнадцатеричном представлении;

%s - последовательность символов, заканчивающаяся нулевым байтом;

%u – беззнаковое число.

Функция форматированного вывода – *printf*

Синтаксис функции *printf*:

printf(строка_форматирования, выдаваемые_значения_переменных);

У функции *printf* может быть один или более параметров. При этом первый параметр присутствует обязательно и является *строкой_форматирования*. Эта строка содержит с одной стороны выдаваемый текст, а, кроме того, в некоторых позициях текста помечаются места расположения значений выдаваемых переменных. Такая пометка называется *спецификатором_форматного_вывода* и задается указанием символа «%» с последующей буквой, определяющей тип выдаваемого значения. Остальные параметры функции *printf* являются *выдаваемыми_значениями_переменных*. При выдаче эти значения последовательно подставляются вместо спецификаторов форматной строки.

Пример.

```
unsigned int value=20;  
printf("здесь присутствует: %u человек\n",value);
```

Функция форматированного ввода – *scanf*

Синтаксис функции *scanf*:

```
scanf("спецификаторы_форматного_ввода", адреса_переменных);
```

Для ввода значений используют функцию *scanf*. При вводе в форматной строке указываются только *спецификаторы_форматного_ввода*, никакой текст там не должен присутствовать. Кроме этого, при вызове функции *scanf* в списке параметров каждой переменной, значение которой должно быть введено, должен предшествовать символ «&» – *адрес_переменной*.

Пример.

```
int value;  
float x;  
scanf("%d%f",&value,&x);
```

2.4. Пример программы на языке Си

Рассмотрим программу печати приветствия «*Hello, world*».

```
/* Программа на языке Си */  
#include <stdio.h>  
#include <stdlib.h>  
/* Основная функция программы: выполнение начинается с нее */  
int main(void)  
{  
    printf("Hello, world\n");     /* Выдача сообщения на стандартное  
                                  устройство вывода */  
    return EXIT_SUCCESS;     /* Возвращение из функции main */  
}
```

Для того чтобы научиться писать программы на конкретном языке программирования, надо как минимум знать *синтаксис* этого языка, т.е. правила написания текста.

Первая строка нашего примера содержит *комментарий*. Для пояснения отдельных фрагментов текста программы используется конструкция вида:

```
/* здесь располагается комментарий */  
/* пример комментария из  
   нескольких строк */
```

Следующие две строки содержат директивы **#include <stdio.h>** и **#include <stdlib.h>**, которые являются командами препроцессора (макропроцессора). Выполнение директивы **#include** состоит в подстановке на ее место содержимого указанного в качестве параметра файла, в нашем случае **stdio.h** и **stdlib.h**.

Далее программа содержит определение функции **main()**. Функция имеет имя, тип возвращаемого значения и список параметров, передаваемых ей при вызове.

Исполнение программы состоит в выполнении функции **main**. Все остальные функции могут быть вызваны либо из функции **main**, либо из какой-то другой функции. Каждая функция состоит из блока, который начинается с открывающей фигурной скобки «{» и завершается фигурной закрывающей скобкой «}». Блок представляет собой тело функции – определение данных и последовательность операторов.

В нашем примере тело функции **main** начинается с оператора выражения

```
printf("Hello, world\n");
```

Этот оператор вызывает функцию с именем **printf**. При выдаче два последних символа в строке «\n» будут восприняты как один управляющий символ перевода курсора на следующую строку. Сама функция **printf** такого перевода не осуществляет по окончании вывода сообщения. Другой пример управляющего символа: «\t» - символ табуляции.

Завершается тело функции оператором **return**. При этом он возвращает управление в то место программы, откуда была вызвана функция. В нашем случае выполнение оператора **return** приведет к завершению работы программы.

2.5. Арифметико-логические операции

Приведенные ниже операции упорядочены по убыванию приоритета.

I. Операции с наивысшим приоритетом

1. () вызов функции;
2. [] выбор элемента массива;

3. -> косвенный выбор элемента;
4. . прямой выбор элемента.

II. Унарные операции

1. ! логическое отрицание (NOT);
2. ~ побитовая инверсия;
3. - унарный минус;
4. ++ увеличение на единицу;
5. -- уменьшение на единицу;
6. & получение адреса переменной;
7. * получение значения по адресу;
8. sizeof размер операнда в байтах.

III. Мультипликативные

1. * умножение;
2. / деление;
3. % вычисление остатка от деления (для целочисленных операндов).

IV. Аддитивные

1. + бинарный плюс;
2. - бинарный минус.

При равном приоритете бинарные операции группируются слева направо.

V. Сдвиги

1. >> сдвиг вправо;
2. << сдвиг влево.

VI. Сравнения

1. < меньше;
2. <= меньше или равно;
3. > больше;
4. >= больше или равно.

VII. Равенство

1. == равно;
2. != не равно.

Результатом выражений с данными операциями является истина (ненулевое значение) или ложь (0).

VIII. Побитовые

1. & побитовое И (AND);
2. ^ побитовое исключающее ИЛИ (XOR);
3. | побитовое ИЛИ (OR).

IX. Логические

1. && логическое И (AND);
2. || логическое ИЛИ (OR).

Логические операции служат для связывания выражений, включающих в себя операции отношения.

X. Условное выражение (тернарная операция)

выражение1 ? выражение2 : выражение3.

Если значение ***выражения1*** отлично от нуля, то результатом выполнения тернарной операции является значение ***выражение2***, в противном случае – значение ***выражения3***.

XI. Присваивание

1. = простое присваивание;
2. *= присвоить произведение;
3. /= присвоить частное;
4. %= присвоить остаток от деления;
5. += присвоить сумму;
6. -= присвоить разность;
7. &= присвоить побитовое AND;
8. ^= присвоить побитовое XOR;
9. |= присвоить побитовое OR;
10. <<= присвоить сдвинутое влево;
11. >>= присвоить сдвинутое вправо.

XII. Запятая

, последовательное вычисление.

2.6. Блок операторов

Синтаксис блока операторов:

```
{  
    оператор1;  
    оператор2;  
    ...  
    операторN;  
}
```

Фигурные скобки «{» и «}» используются для объединения операторов в блок, чтобы с точки зрения синтаксиса эта новая конструкция воспринималась как один оператор.

Пример.

```
if(x>0)
{
    y=sqrt(x); /* Блок */
    x++;      /* операторов */
}
```

2.7. Операторы, управляющие потоком выполнения (*if*, *switch*, *for*, *while*, *do...while*)

Синтаксис первой формы оператора *if*:

```
if ( выражение )
    оператор
```

Схема выполнения первой формы оператора **if**:

- вычисляется **выражение** в круглых скобках;
- если значение **выражения** отлично от нуля (истина), выполняется **оператор**, если **выражение** равно нулю (ложь), то управление передается на оператор, следующий за оператором **if**.

Синтаксис второй формы оператора *if*:

```
if (выражение)
    оператор1;
else
    оператор2;
```

Схема выполнения второй формы оператора **if**:

- вычисляется **выражение** в круглых скобках;
- если значение **выражения** отлично от нуля (истина), выполняется **оператор1**, если **выражение** равно нулю (ложь), то выполняется **оператор2**.

Синтаксис оператора переключателя *switch*:

```
switch ( выражение )
{
    case константное_выражение1: последовательность_
                                операторов
    case константное_выражение2: последовательность_
                                операторов
    ...
    case константное_выражениеN: последовательность_
                                операторов
    default: последовательность_
                операторов
}
```

Схема выполнения оператора **switch** следующая:

- вычисляется **выражение** в круглых скобках;
- полученное значение последовательно сравнивается с **константными_выражениями**, следующими за ключевыми словами **case**;
- если одно из **константных_выражений** совпадает со значением **выражения**, то управление передается на соответствующие ему **операторы**;
- если ни одно из **константных_выражений** не равно **выражению**, то управление передается на **операторы**, помеченные ключевым словом **default**, а в случае его отсутствия управление передается на следующий после **switch** оператор;
- для завершения работы оператора **switch** после выполнения **последовательности_операторов** нужно обязательно использовать оператор **break**.

Синтаксис оператора цикла **for**:

for(выражение1 ; выражение2 ; выражение3)
оператор;

Выражение1 обычно используется для установки начального значения переменных, управляющих циклом. **Выражение2** - это выражение, определяющее условие, при котором тело цикла будет выполняться. **Выражение3** определяет изменение переменных, управляющих циклом после каждого выполнения тела цикла.

Схема выполнения оператора **for**:

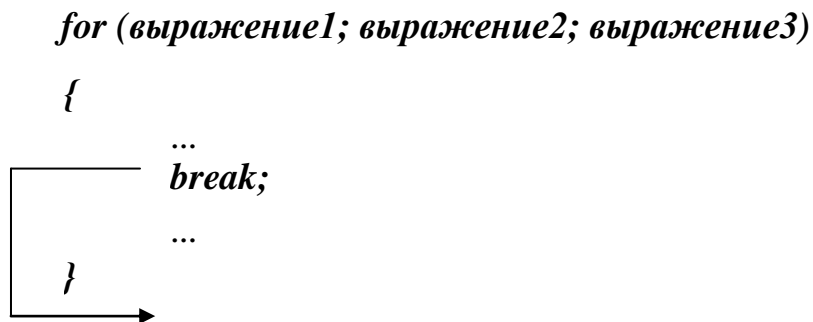
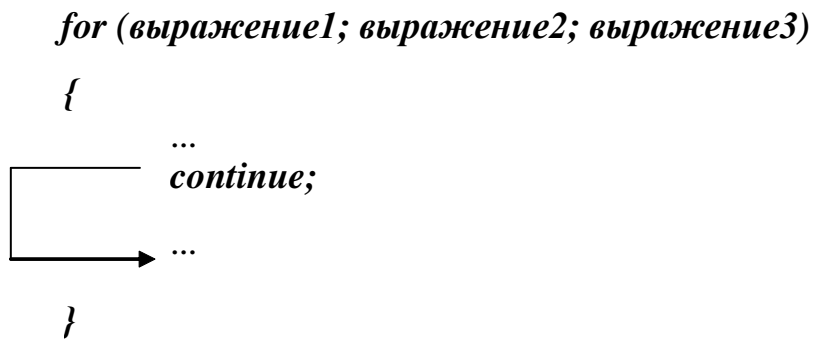
1. Вычисляется **выражение1**.
2. Вычисляется **выражение2**.
3. Если значения **выражения2** отлично от нуля (истина), выполняется **оператор**, вычисляется **выражение3** и осуществляется переход к пункту 2, если **выражение2** равно нулю (ложь), то оператор цикла завершается и управление передается на оператор, следующий за оператором **for**.

Действия операторов **break** и **continue**

Последовательность выполнения оператора **for** может быть изменена с помощью операторов **break** и **continue**.

Оператор **break** вызывает завершение работы цикла.

Оператор **continue** вынуждает цикл **for** не выполнять оставшиеся операторы цикла и перейти к вычислению **выражения3**, т.е. к следующей итерации.



Синтаксис оператора цикла **while**:

```

while (выражение)
    оператор;

```

Схема выполнения оператора **while**:

1. Вычисляется **выражение**.
2. Если **выражение** равно нулю (ложь), то выполнение оператора **while** заканчивается и выполняется следующий по порядку оператор. Если **выражение** отлично от нуля (истина), то выполняется **оператор**, являющийся телом цикла.
3. Процесс повторяется с пункта 1.

Синтаксис оператора цикла **do...while**:

```

do
    оператор
while (выражение);

```

Схема выполнения оператора **do...while**:

1. Выполняется **оператор** тела цикла.
2. Вычисляется **выражение**.
3. Если **выражение** равно нулю (ложь), то выполнение оператора **do...while** заканчивается и выполняется следующий по порядку оператор. Если **выражение** отлично от нуля (истина), то выполнение оператора продолжается с пункта 1.

2.8. Функции

В виде функции оформляются вычисления, преобразования, другие действия, которые затем могут быть выполнены неоднократно с различными значениями **аргументов (параметров)**.

С помощью функции большие вычислительные задачи подразделяют на более мелкие. Они позволяют воспользоваться тем, что уже сделано другими разработчиками, а не начинать создание программы каждый раз "с нуля". Язык проектировался так, чтобы функции были эффективным и простым в использовании средством программирования. Обычно программы на языке Си представляют собой набор небольших функций. Программу можно располагать в одном или нескольких исходных файлах. Эти файлы можно компилировать отдельно, а компоновку выполнять вместе, в том числе и с ранее откомпилированными библиотечными функциями.

Часто оказывается предпочтительным реализовать ряд операций один раз в виде некоторых модулей и в дальнейшем иметь возможность присоединять эти модули к своей программе.

Объявление, определение и вызов функций

Определение функции имеет следующий вид:

```
тип_возвращаемого_значения имя(список_формальных_параметров)  
{  
    последовательность_операторов;  
}
```

Определение функции задает: 1) тип возвращаемого функцией значения; 2) имя функции; 3) список формальных параметров; 4) последовательность операторов. Тело функции, заключенное в фигурные скобки, содержит объявления автоматических переменных и операторы, определяющие действия, выполняемые функцией.

Прототип функции имеет следующий вид:

```
тип_возвращаемого_значения имя(список_формальных_параметров);
```

Вызов функции имеет следующий вид:

```
имя(список_фактических_параметров );
```

Пример с использованием прототипа, вызова и определения функции.

```
/* Прототип функции */  
int volume(int a, int b, int c);
```

```

int main(void)
{
    int x=5, y=6, v;
    v=volume(x,y,8); /* Вызов функции */
    return 0;
}

/* Определение функции */
int volume(int a, int b, int c)
{
    return a*b*c;
}

```

Функции могут требовать указания параметров при своем *вызове*. Такие функции в заголовке содержат непустой список формальных параметров, который представляет собой последовательность объявлений формальных параметров, разделенных запятыми. *Формальные параметры* - это переменные, используемые внутри тела функции и получающие значение при вызове функции путем **копирования** в них значений соответствующих *фактических параметров*. *Фактические параметры* - это значения, передаваемые в функцию при ее вызове. В том случае, если функция не использует параметров, то список параметров оставляют пустым.

Важно, что *типы фактических параметров* при вызове функции должны быть совместимы с типами соответствующих *формальных параметров*.

Функция может возвращать значение любого типа данных, за исключением массива. Функция завершает работу при выполнении оператора **return**, имеющего следующий синтаксис:

return *выражение*;

Выражение должно соответствовать типу возвращаемого значения. Значение указанного выражения возвращается в точку вызова функции в качестве результата.

Память для параметров при вызове функции отводится компилятором. Это означает, что компилятор должен иметь информацию о списке передаваемых в функцию параметров, что обеспечивается *объявлением* функции путем указания ее прототипа.

Прототип должен обязательно предшествовать вызову функции. В нем указывается имя функции, тип возвращаемого значения и типы передаваемых в функцию аргументов.

Кроме того, с помощью прототипа компилятор осуществляет проверку соответствия типов передаваемых фактических параметров типам формальных параметров при вызове функции.

Область действия переменных и время жизни

Программа на языке Си обычно оперирует с множеством внешних объектов: переменных и функций.

Глобальные переменные объявляются вне функций и доступны для любой функции. Они определяются в одном из модулей, а в тех модулях, где к ним нужен доступ, они описываются с помощью модификатора *extern*.

Иногда требуется ограничить область видимости глобальной переменной только текущим модулем. В этом случае переменная определяется как *статическая* с помощью модификатора *static*.

Локальные (автоматические) переменные определяются внутри функций. Эти переменные действительны только в ней, создаются в момент входа в функцию и уничтожаются при выходе из нее.

Рассмотрим следующую таблицу:

Переменные	Область действия (видимости)	Время жизни
Глобальные	Программа	Программа
Статические	Модуль (отдельно откомпи- лированный файл)	Программа
Автоматические	Функция	Функция

2.9. Указатели

Указатель – это переменная, содержащая адрес другой переменной.
Синтаксис определения указателя:

*спецификатор_типа *имя переменной;*

Символ «*» означает "указатель на". *Спецификатор_типа* задает тип переменной, на которую указывает указатель.

Пример определения и инициализации указателя.

```
int c;  
int *p;      /* Определение указателя */  
p=&c; /* Инициализация указателя */
```

Результатом выполнения операции «&» является адрес объекта, так что в операторе

```
p=&c;
```

переменной *p* присваивается адрес ячейки *c* (говорят, что *p* указывает на *c* или *p* ссылается на *c*). Операция «&» применяется только к объектам, расположенным в памяти: к переменным и элементам массивов. Ее операндом не может быть ни выражение, ни константа.

Символ «*» перед именем переменной имеет разное значение в определении и в использовании. При определении он означает, что переменная является указателем, а при использовании – это получение значения переменной, на которую указывает указатель.

Пример с использованием указателя в операторах.

```
int x, y;  
int *ip;    /* ip – указатель на тип int */  
ip=&x;     /* Переменной ip присваивается адрес переменной x */  
y=*ip;     /* Значению y присваивается значение x */  
*ip=0;     /* Значению x присваивается значение 0 */
```

Допускаются следующие операции с указателями: присваивание значения указателя другому указателю того же типа, сложение и вычитание указателя и целого, вычитание и сравнение двух указателей, ссылающихся на элементы одного и того же массива, а также присваивание указателю нуля и сравнение указателя с нулем.

Нельзя складывать два указателя, перемножать их, делить, сдвигать, выделять разряды; указатель нельзя складывать со значением типа *float* или *double*; указателю одного типа нельзя присвоить указатель другого типа.

Указатели и массивы

Массив – это набор пронумерованных однотипных данных.

Синтаксис определения массива:

спецификатор_типа array[константное_выражение];

В определении массива *array* – имя массива, *константное_выражение* – количество элементов в массиве, *спецификатор_типа* – тип элементов в массиве. Для данного типа *T*, *T[size]* есть тип «массив из *size* элементов типа *T*». Все элементы массива пронумерованы, начиная с нуля.

При определении массива его элементы можно инициализировать следующим образом:

спецификатор_типа array[]={ значение1, значение2, ..., значениеN };

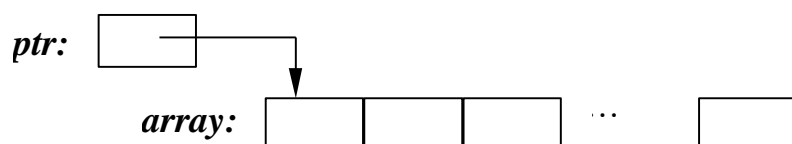
В фигурных скобках { } через запятую перечисляются значения элементов массива соответствующего типа.

В языке СИ между указателями и массивами существует некоторая связь. Когда объявляется массив, то его имя отождествляется с константой, значение которой равно адресу нулевого элемента массива.

Пример определения и использования массива.

```
int array[5]={2, 8, 45, 16, 12}; /* Определение и инициализация массива */  
int *ptr;                      /* Определение указателя */  
ptr=&array[0]; /* Присваивание адреса нулевого элемента массива  
                указателю ptr */
```

Указатель *ptr* устанавливается на адрес нулевого элемента массива, причем присваивание *ptr=&array[0]* можно записать в эквивалентной форме *ptr=array*.



Между именем массива и указателем, выступающим в роли имени массива, существует важное различие. Указатель – это переменная, поэтому допустима операция изменения ее значения: *ptr=array* или *ptr++*. Но имя массива не является переменной, и записи типа *array =ptr* или *array++* ошибочны.

2.10. Структуры

Структура - это составной объект, в который входят элементы любых типов, за исключением функций. В отличие от массива, который является однородным объектом, структура может быть неоднородной.

Синтаксис определения структурного типа данных:

```
struct имя_типа  
{  
    спецификатор_типа имя_элемента;  
    спецификатор_типа имя_элемента;  
    ...  
};
```


Служебное слово **struct** указывает начало определения структуры. За ним следует **имя_типа**, которое присваивается данному структурному типу; оно называется идентификатором структуры. Затем в фигурных скобках указывается список отдельных элементов, образующих структуру. Элементы структуры могут иметь любой тип, включая массивы, другие структуры.

Над структурами определены следующие операции: доступ к элементам структуры через «.» и через указатель на нее «→». Доступ к элементам структуры организован по имени элемента, а не по номеру как в массиве.

Пример с использованием типа данных **struct Student**, описывающего информацию о студенте.

```
struct Student
{
    char name[20];           /* Фамилия */
    unsigned int id;         /* Номер зачетной книжки */
    unsigned int age;        /* Возраст */
};

struct Student group[30];   /* Определение массива структур типа
                               struct Student */
struct Student starosta={"Иванов", 990345, 20}; /* Инициализация
                                                    переменной типа struct Student */
group[0]=starosta; /* Присваивание нулевому элементу массива
                     переменной типа struct Student */
```

Указатели на структуру

В выше указанном примере была определена переменная **starosta** структуры **Student**. Теперь определим указатель **ptr** на эту переменную. Для примера, иллюстрирующего доступ к структуре, определим еще одну переменную **unsigned int age_first**.

```
struct Student *ptr=&starosta; /* ptr – указатель на тип struct Student */
unsigned int age_first;
```

Синтаксис доступа к элементам структуры с использованием операций: «.», «→»:

```
имя_переменной.имя_элемента
имя_указателя->имя_элемента
```

Пример с использованием операций: «.», «→».

```
age_first=starosta.age;  
age_first=ptr→age;
```

Структуры и функции

Передача структур функциям в качестве аргументов и возврат их от функций в виде результата относятся к операциям копирования и присваивания.

Если функции передается структура, то рекомендуется передавать ее через указатель.

3. Порядок выполнения лабораторных работ

1. Подготовка к работе.

Изучите теоретическую часть по данной теме. Ознакомьтесь с вариантом задания на лабораторную работу, определяемым преподавателем.

2. Выполнение работы.

Выполнение работы состоит в следующем:

- анализ варианта задания;
- разработка и запись алгоритма;
- представление разработанного алгоритма преподавателю;
- составление программы на языке программирования Си;
- отладка полученной программы;
- внесение в программу изменений, предложенных преподавателем.

3. Отчетность лабораторной работы.

Отчет по лабораторной работе должен содержать:

- исходные данные;
- алгоритм, составленный с помощью блок-схемы;
- представление на экране исходного текста программы и результата ее работы, текст программы должен иметь комментарии.

4. Контрольные вопросы

1. Укажите базовые типы данных.
2. Перечислите арифметико-логические операции над переменными.
3. С помощью каких операторов можно организовать разветвляющийся алгоритм?
4. Приведите синтаксис операторов *if*, *switch*.
5. Приведите синтаксис операторов *for*, *while*, *do...while*.

6. Что такое функция?
7. Приведите синтаксис прототипа, определения, вызова функции.
8. Что такое формальные и фактические параметры?
9. Как определить и инициализировать указатели в программе?
10. Каким образом можно осуществить доступ к элементам массива?
11. Дайте определение массива и правила его инициализации.
12. Дайте определение структуры.
13. Как осуществить доступ к элементам структуры?

5. Список рекомендуемой литературы

1. Павловская Т.А. С/С++. Программирование на языке высокого уровня: Учебник / Т.А. Павловская.- Спб.: Питер, 2009. – 461 с.
2. Борисенко В.В. Основы программирования / В.В.Борисенко.- Москва: Интернет-Университет информационных технологий, 2012. – 314 с.
3. Костюкова Н.И., Калинина Н.А. Язык Си и особенности работы с ним.- Москва: Интернет-Университет информационных технологий, 2009. – 200 с.