Основы алгоритмизации и программирования

Лабораторная работа №3.

Цель работы - изучить инструкции условий и циклов в языке программирования Си.
Содержание отчёта:

1. Титульный лист
2. Формулировка цели и задач работы
3. Описание результатов выполнения
4. Выводы, согласованные с целью

Условные инструкции

Инструкция if-else используется для принятия решения. Формально ее синтаксисом является:

```
if (выражение)
        инструкция1
else
        инструкция2
```

причем else-часть может и отсутствовать. Сначала вычисляется выражение, и, если оно истинно (т. е. отлично от нуля), выполняется инструкция1. Если выражение ложно (т. е. его значение равно нулю) и существует else-часть, то выполняется инструкция2.

Так как if просто проверяет числовое значение выражения, условие иногда можно записывать в сокращенном виде. Так, запись

```
if (выражение)
```

короче, чем

```
if (выражение! = 0)
```

Иногда такие сокращения естественны и ясны, в других случаях, наоборот, затрудняют понимание программы.

Отсутствие else-части в одной из вложенных друг в друга if-конструкций может привести к неоднозначному толкованию записи. Эту неоднозначность разрешают тем, что else связывают с ближайшим if, у которого нет своего else. Например, в

```
if (n> 0)
        if (a > b)
                z= a;
        else
                z= b;
```

else относится к внутреннему if, что мы и показали с помощью отступов. Если нам требуется иная интерпретация, необходимо должным образом расставить фигурные скобки:

```
if (n > 0) {
        if (a > b)
                z= a;
} else
        z= b;
```

Кстати, обратите внимание на точку с запятой после z = a в

```
if (a > b)
        z = a;
else
        z= b;
```

Здесь она обязательна, поскольку по правилам грамматики за if должна следовать инструкция, а выражение-инструкция вроде z= a;всегда заканчивается точкой с запятой.

Конструкция

```
if (выражение)
        инструкция
else if (выражение)
        инструкция
else if (выражение)
        инструкция
else if (выражение)
        инструкция
else
        инструкция
```

встречается так часто, что о ней стоит поговорить особо. Приведенная последовательность инструкций if—самый общий способ описания многоступенчатого принятия решения. Выражения вычисляются по порядку;как только встречается выражение со значением "истина", выполняется соответствующая ему инструкция; на этом последовательность проверок завершается. Здесь под словом инструкция имеется в виду либо одна инструкция, либо группа инструкций в фигурных скобках.

Последняя else-часть срабатывает, если не выполняются все предыдущие условия. Иногда в последней части не требуется производить никаких действий, в этом случае фрагмент

```
else
        инструкция
```

можно опустить или использовать для фиксации ошибочной ("невозможной") ситуации.

Инструкция switch используется для выбора одного из многих путей. Она проверяет, совпадает ли значение выражениясо одним из значений, входящих в некоторое множество целых констант, и выполняет соответствующую этому значению ветвь программы:

```
switch (выражение) {
case конст-выр: инструкции
case конст-выр: инструкции
default: инструкции
}
```

Каждая ветвь case помечена одной или несколькими целочисленными константами или же константными выражениями. Вычисления начинаются с той ветви case,в которой константа совпадает со значением выражения. Константы всех ветвей case должны отличаться друг от друга. Если выяснилось, что ни одна из констант не подходит, то выполняется ветвь,помеченная словом default, если таковая имеется, в противном случае ничего не делается. Ветви case и default можно располагать в любом порядке.

Инструкция break вызывает немедленный выход из переключателя switch. Поскольку выбор ветви case реализуется как переход на метку, то после выполнения одной ветви case, если ничего не предпринять, программа провалится вниз на следующую ветвь. Инструкции break и return—наиболее распространенные средства выхода из переключателя. Инструкция break используется также для принудительного выхода из циклов while, for и do-while(мы еще поговорим об этом чуть позже).

"Сквозное" выполнение ветвей case вызывает смешанные чувства. С одной стороны, это хорошо, поскольку позволяет несколько ветвей case объединить в одну, как мы и поступили с цифрами в нашем примере. Но с другой это означает, что в конце почти каждой ветви придется ставить break,чтобы избежать перехода к следующей.Последовательный проход по ветвям —вещь ненадежная, это чревато ошибками, особенно при изменении программы. За исключением случая с несколькими метками для одного вычисления, старайтесь по возможности реже пользоваться сквозным проходом, но если уж вы его применяете, обязательно комментируйте эти особые места.

Циклические инструкции

В цикле

```
while (выражение)
        инструкция
```

вычисляется выражение. Если его значение отлично от нуля, то выполняется инструкция, и вычисление выражения повторяется. Этот цикл продолжается до тех пор, пока выражение не станет равным нулю, после чего вычисления продолжатся с точки, расположенной сразу за инструкцией.

Инструкция for

```
for (выр1;выр2;выр3)
```

```
        инструкция
```

эквивалентна конструкции

```
выр1;
while (выр2) {
        инструкция;
        выр3;
}
```

С точки зрения грамматики три компоненты цикла for представляют собой произвольные выражения, но чаще выр1 и выр3—это присваивания или вызовы функций, а выр2—выражение отношения. Любое из этих трех выражений может отсутствовать, но точку с запятой опускать нельзя. При отсутствии выр1 или выр3 считается, что их просто нет в конструкции цикла; при отсутствии выр2 предполагается, что его значение как бы всегда истинно. Например,

```
for(;;) {
        ...
}
```

есть "бесконечный" цикл, выполнение которого, вероятно, прерывается каким-то другим способом, например с помощью инструкций break или return.

Какой цикл выбрать: while или for—это дело вкуса. Так, в

```
while((c = getchar()) == ' ' ||c == '\n' ||c == '\t');
        /* обойти символы-разделители */
```

нет ни инициализации, ни пересчета параметра, поэтому здесь больше подходит while.

Там, где есть простая инициализация и пошаговое увеличение значения некоторой переменной, больше подходит цикл for, так как в этом цикле организующая его часть сосредоточена в начале записи. Например, начало цикла, обрабатывающего первые n элементов массива, имеет следующий вид:

```
for (i= 0; i< n; i++)
        ...
```

Это похоже на DO-циклы в Фортране и for-циклы в Паскале. Сходство,однако, не вполне точное, так как в Си индекс и его предельное значение могут изменяться внутри цикла, и значение индекса i после выхода из цикла всегда определено. Поскольку три компоненты цикла могут быть произвольными выражениями, организация for-циклов не ограничивается только случаем арифметической прогрессии. Однако включать в заголовок цикла вычисления, не имеющие отношения к инициализациии инкрементированию, считается плохим стилем. Заголовок лучше оставить только для операций управления циклом.

В циклах while и for проверка условия окончания цикла выполняется наверху. В Си имеется еще один вид цикла, do-while,в котором эта проверка в отличие от while и for

делается внизу после каждого прохождения тела цикла, т. е. после того, как тело выполнится хотя бы один раз. Цикл do-while имеет следующий синтаксис:

```
do
        инструкция
while (выражение);
```

Сначала выполняется инструкция, затем вычисляется выражение. Если оно истинно, то инструкция выполняется снова и т. д. Когда выражение становится ложным, цикл заканчивает работу. Цикл do-while эквивалентен циклу repeat-until в Паскале с той лишь разницей, что в первом случае указывается условие продолжения цикла, а во втором — условие его окончания.

Опыт показывает, что цикл do-while используется гораздо реже, чем while и for. Тем не менее,потребность в нем время от времени возникает.

Иногда бывает удобно выйти из цикла не по результату проверки, осуществляемой в начале или в конце цикла, а каким-то другим способом. Такую возможность для циклов for, while и do-while, а также для переключателя switch предоставляет инструкция break. Эта инструкция вызывает немедленный выход из самого внутреннего из объемлющих ее циклов или переключателей.

Инструкция continue в чем-то похожа на break, но применяется гораздо реже. Она вынуждает ближайший объемлющий ее цикл (for, while или do-while) начать следующий шаг итерации. Для while и do-while это означает немедленный переход к проверке условия, а для for—к приращению шага. Инструкцию continue можно применять только к циклам, но не к switch. Внутри переключателя switch, расположенного в цикле,она вызовет переход к следующей итерации этого цикла.

Вот фрагмент программы, обрабатывающий только неотрицательные элементы массива a(отрицательные пропускаются).

```
for ( i= 0 ; i < n; i++) {
        if (a[i] < 0) /* пропуск отрицательных элементов*/
                continue; /* обработка положительных элементов */
}
```

К инструкции continue часто прибегают тогда, когда оставшаяся часть цикла сложна, а замена условия в нем на противоположное и введение еще одного уровня приводят к слишком большому числу уровней вложенности.

В Си имеются порицаемая многими инструкция goto и метки для перехода на них. Строго говоря, в этой инструкции нет никакой необходимости, и на практике почти всегда легко без нее обойтись. До сих пор в нашей книге мы не использовали goto.

Однако существуют случаи, в которых goto может пригодиться. Наиболее типична ситуация, когда нужно прервать обработку в некоторой глубоко вложенной структуре и выйти сразу из двух или большего числа вложенных циклов. Инструкция break здесь не поможет, так как она обеспечит выход только из самого внутреннего цикла. В качестве примера рассмотрим следующую конструкцию:

```
for(...)
        for(...) {
                ...
                if(disaster) /* если бедствие */
```

```
                goto error; /* уйти на ошибку */
        }
...
error: /* обработка ошибки */
ликвидировать беспорядок
```

Такая организация программы удобна, если подпрограмма обработки ошибочной ситуации не тривиальна и ошибка может встретиться в нескольких местах.

Метка имеет вид обычного имени переменной, за которым следует двоеточие. На метку можно перейти с помощью goto из любого места данной функции, т. е. метка видима на протяжении всей функции.

В качестве еще одного примера рассмотрим такую задачу: определить,есть ли в массивах a и b совпадающие элементы. Один из возможных вариантов ее реализации имеет следующий вид:

```
for(i= 0 ; i< n; i++)
        for (j = 0; j < m; j++)
                if (a[i] == b[j])
                        goto found; /* нет одинаковых элементов */
...
found:
/* обнаружено совпадение: a[i] == b[j] */
...
```

Программу нахождения совпадающих элементов можно написать и без goto, правда, заплатив за это дополнительными проверками и еще одной переменной:

```
found = 0;
for (i = 0; i < n && !found; i++)
        for (j = 0; j < m && !found;j++)
                if (a[i] == b[j])
                        found = 1;
        if(found)
                /* обнаружено совпадение: a[i-1] == b[j-1] */
                ...
        else
                /* нет одинаковых элементов */
                ...
```

За исключением редких случаев, подобных только что приведенным,программы с применением goto, как правило, труднее для понимания и сопровождения, чем программы, решающие те же задачи без goto. Хотя мы и не догматики в данном вопросе, все же думается, что к goto следует прибегать крайне редко, если использовать эту инструкцию вообще.

Используемые источники:

1. Язык программирования Си. Авторы: Б. Керниган, Д. Ритчи.
2. Онлайн компилятор языка Си: https://www.onlinegdb.com/online_c_compiler
3. Онлайн блок-схемы: https://app.diagrams.net

Задание:

При решении каждой задачи также создавайте блок-схемы. Решением работы должен служить файл-отчёт, в котором будут располагаться скриншоты кода, выполнения программ, а также сами блок-схемы. Всего программ и блок-схем должно быть три.

1.

1.1. Сгенерировать случайное трехзначное число, оканчивающееся на ноль. Функции для генерации псевдослучайных чисел - srand, rand.

1.2. Написать программу, реализующая функцию y = f(x) следующего вида:

y = x^3, если x > 0,
y = 12, если x = 0,
y = -x * 3, если x < 0.

1.3. Существует натуральное число a. Найти сумму четных цифр, входящих в его состав.

2.

2.1. Из двух чисел (a, b) с разной четностью вывести на экран нечетное число.

2.2. Написать программу, реализующая функцию y = f(x) следующего вида:

y = x+1, если x > 0,
y = 0, если x = 0,
y = x-1, если x < 0.

2.3. Составить таблицу значений функции $y = 5 - x^2/2$ на отрезке [-5; 5] с шагом 0.5.

3.

3.1. Определить какое из трех чисел максимальное и вывести его на экран.

3.2. Написать программу, реализующая функцию y = f(x) следующего вида:

y = x + 23, если x > 0,
y = 0, если x = 0,
y = -6, если x < 0.

3.3. Написать программу, подсчитывающую количество четных и нечетных цифр в числе.

4.

4.1. Среди трех чисел найти среднее. Если среди чисел есть равные, вывести сообщение "Ошибка".

4.2. Написать программу, реализующая функцию y = f(x) следующего вида:

y = x * 3, если x > 0,
y = x, если x = 0,
y = x / 2, если x < 0.

4.3. Существует целое число. Преобразовать его в другое число, цифры которого будут следовать в обратном порядке по сравнению с введенным числом.

5.

5.1. В зависимости от того, в каких единицах измерения вводится значение, перевести его в другие единицы измерения. В данном случае переводится объем информации (байты, килобайты, мегабайты и гигабайты).

5.2. Написать программу, реализующая функцию y = f(x) следующего вида:

y = x^2, если x > 0,
y = 1, если x = 0,
y = -x, если x < 0.

5.3. Вывести на экран кубы чисел от a до b.

6.

6.1. Определить какое число - отрицательное или положительное.

6.2. Написать программу, реализующая функцию y = f(x) следующего вида:

y = x – 2, если x > 0,
y = 0, если x = 0,
y = |x|, если x < 0.

6.3. Существует целое число. Вывести на экран факториал этого числа.

7.

7.1. Существует целое число, обозначающее код символа по таблице ASCII. Определить, это код английской буквы или какой-либо иной символ.

7.2. Написать программу, реализующая функцию y = f(x) следующего вида:

y = x^2 + 3, если x > 0,
y = 1, если x = 0,
y = |x - 22|, если x < 0.

7.3. Существует целое число n. Вывести на экран ряд чисел фибоначчи, состоящий из n элементов.

8.

8.1. Существует два целых числа a,b. Проверить делится ли первое на второе. Вывести на экран сообщение об этом, а также остаток (если он есть) и частное (в любом случае).

8.2. Написать программу, реализующая функцию y = f(x) следующего вида:

y = x^3 - 1, если x > 0,
y = -33, если x = 0,
y = x / 3, если x < 0.

8.3. Возвести число a в степень b, не используя стандартных функций языка.