Учреждение образования

«Белорусский государственный технологический университет»

**Реферат**

по дисциплине «Основы алгоритмизации и программирования»

по теме «Условные операторы»

Работу выполнил:

Студент ИСиТ 1-1

**Палазник А.В.**

Минск, 2021

**Задача**: познакомиться с условными операторами, узнать для чего они нужны, рассказать о каждом из них и в заключение сделать определенные выводы.

**Алгоритм решения:**

1. Что такое условные операторы, для чего они нужны и их виды

2. Оператор if

3. Операторы if-else

4. Операторы if-else-if

5. Вложенность операторов if-else

6. Условный оператор?

7. Оператор switch

8. Вложенность операторов switch

9. Оператор goto

10. Вывод

**Что такое условные операторы, для чего они нужны и их виды**

Часто бывает такое, что нам нужно, чтобы определенная часть кода выполнялась только при конкретном условии. В таких ситуациях нас и спасают условные операторы. При условии истинности команда (или набор команд) будут выполняться, при ложном условии – нет.

В C++ используются следующие условные операторы: if, if-else, if-else-if, ?, switch.

**Оператор if**

Оператор if –простейший из всех операторов ветвления

Синтаксис:

If (условие) выражение; // если выражение одно

If (условие) {

Много выражений; /\*если выражений несколько, мы помещаем их в фигурные скобки \*/

}

Если условие внутри скобок истинно, выражение (или блок выражений) выполняются, если нет – игнорируются

Пример:

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int age;

cout << "Введите значение переменной age: ";

cin >> age;

if (age < 0)

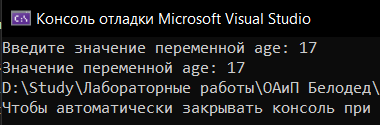
age = 0;

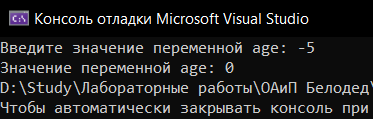
cout << "Значение переменной age: " << age;

}

При отрицательном значении переменной age, переменной будет присваиваться значение 0. При неотрицательном значении переменная остается прежней

Результат:





**Оператор if-else**

Принцип работы этого оператора заключается в выборе между двумя участками кода.

Синтаксис:

If (условие)

Выражение1;

Else

Выражение2;

*Или*

If (условие) {

Выражения\_если\_условие\_истинно;

}

Else {

Выражения\_если\_условие\_не\_истинно;

}

Если выражение внутри скобок истинно, выполняется строка после if (или блок кода), в противном случае – строка после else (или блок кода).

Пример:

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int age;

cout << "Введите значение переменной age: ";

cin >> age;

if (age < 0)

cout << "Вы еще не родились";

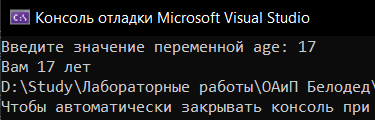
else

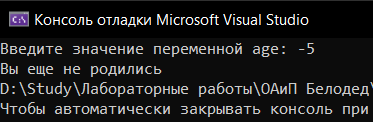
cout << "Вам " << age << " лет";

}

Если значение переменной age отрицательно, срабатывает первый cout, если нет – выполняется второй cout.

Результат:





**Оператор if-else-if**

Но что же делать, если нам нужно больше вариаций условий. Что делать, если оператор if-else дает нам выбор из двух фрагментов кода, а нам этого недостаточно? На помощь приходит оператор if-else-if. С помощью его мы можем записывать неограниченное количество условий и под каждое писать свои выражения.

Синтаксис:

If (условие1)

Выражение1;

Else If (условие2)

Выражение2;

Else if (условие3)

Выражение3;

Else if (условие4)

Выражение4;

//В конце можно также добавлять оператор else

В первом if идет условие на истинность. Если оно истинно – выражение1 выполняется, если нет – игнорируется. То же самое происходит и с последующими операторами if-else. Поэтому в вышеприведенной схеме может выполниться как все 4 выражения, так и ни одного.

Пример:

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int age;

cout << "Введите значение переменной age: ";

cin >> age;

if (age < 0)

cout << "Вы еще не родились";

else if (age == 0 || age > 4)

cout << "Вам " << age << " лет";

else if (age == 1)

cout << "Вам 1 год";

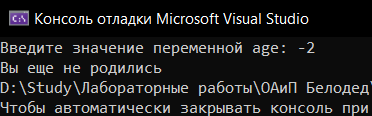
else {

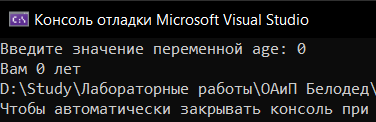
cout << "Вам " << age << " года";

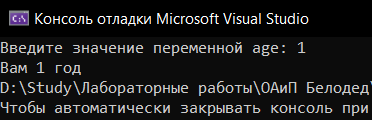
}

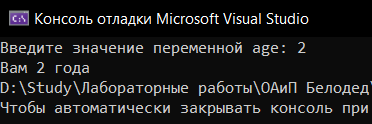
}

Результат:









**Вложенность операторов if-else**

Операторы if-else можно помещать в другие операторы if-else. Таким образом можно получить довольно интересные результаты.

Синтаксис:

If (условие1) {

If (условие2)

Выражение1;

Else

Выражение2;

}

Else

Выражение3;

Пример:

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int age;

cout << "Введите значение переменной age: ";

cin >> age;

if (age >= 0) {

if (age == 0 || age > 4)

cout << "Вам " << age << " лет";

else if (age == 1)

cout << "Вам один год";

else

cout << "Вам " << age << " года";

}

else

cout << "Вы еще не родились";

}

Результат тот же, что и в предыдущем примере.

**Условный оператор ?**

Условный оператор ? – это, можно сказать, просто краткая запись оператора if-else. Сначала мы записываем условие, после которого ставим вопросительный знак, а после пишем выражение при истине и ложное выражение, разделяя их двоеточием. В конце ставим точку с запятой.

Синтаксис:

Условие? Действие\_если\_истина : действие\_если\_нет;

Пример:

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int age;

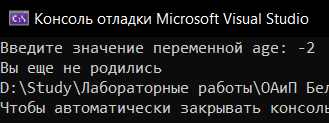
cout << "Введите значение переменной age: ";

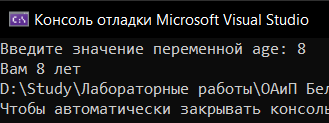
cin >> age;

age > 0 ? cout << "Вам " << age << " лет" : cout << "Вы еще не родились";

}

Результат:





**Оператор switch**

Как уже говорилось, если нам нужно больше двух вариаций условий, мы можем использовать оператор if-else-if, однако, когда этих вариаций слишком много, используя этот оператор наш код будет выглядеть очень неудобно и некрасиво. В таких ситуациям нужно прибегать к оператору switch.

Синтаксис:

Switch (общее\_выражение) {

case выражение\_подходящее\_под\_общее1:

операторы;

break;

case выражение\_подходящее\_под\_общее2:

операторы;

break;

case выражение\_подходящее\_под\_общее3:

операторы;

break;

…

case выражение\_подходящее\_под\_общееN:

операторы;

break;

default:

операторы;

}

На место общего выражения обычно вставляется переменная. Case начинает проверку на истинность. Если общее выражение является «выражением, подходящее под выражение1», выполняются операторы внутри этого case. Если нет – операторы внутри case игнорируются. Оператор break совершает переход к следующему case. В случае, если ни один case не подошел, выполняется код в содержимом default. Оператор default является необязательным.

Также операторы case можно совмещать друг с другом

Пример:

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int age;

cout << "Введите значение переменной age(до 9): ";

cin >> age;

switch (age) {

case 0: case 5: case 6: case 7: case 8: case 9:

cout << "Вам " << age << " лет";

break;

case 1:

cout << "Вам 1 год";

break;

case 2: case 3: case 4:

cout << "Вам " << age << " года";

break;

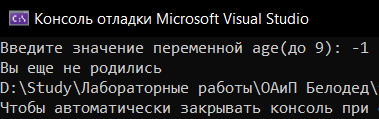
default:

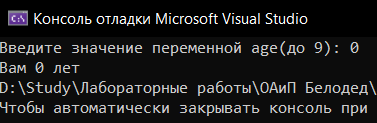
cout << "Вы еще не родились";

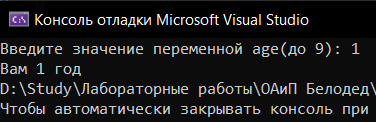
}

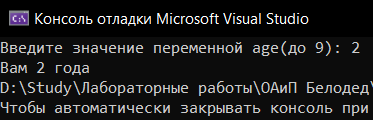
}

Результат:









**Вложенность операторов switch**

Вложенность операторов switch заключается в том, что мы можем засовывать новые операторы switch, в case первоначального switch

Пример:

int main()

{

setlocale(LC\_ALL, "Russian");

int c;

cout << "Здравствуйте, как вы себя чувствуете?(1-отлично; 2-бывало и лучше; 3-плохо" << endl;

cin >> c;

switch (c) {

case 1: {cout << "Прекрасно! Были ли какие-нибудь осложнения(1-да; 2-нет)" << endl;

cin >> c;

switch (c) {

case 1: cout << "Употребляйте эти таблетки еще несколько дней, организм должен привыкнуть" << endl; break;

case 2: cout << "Отлично, мы не будем прописывать вам новые таблетки" << endl; break;

}

break;

}

case 2: {cout << "Это из-за таблеток?(1-да; 2-нет)" << endl;

cin >> c;

switch (c) {

case 1: cout << "Употребляйте эти таблетки еще несколько дней, организм должен привыкнуть" << endl; break;

case 2: cout << "Мы запишем вас на прием к психологу" << endl; break;

}

break;

}

case 3: {cout << "Это из-за таблеток?(1-да; 2-нет)" << endl;

cin >> c;

switch (c) {

case 1: cout << "Мы пропишем вам новые таблетки" << endl; break;

case 2: cout << "Мы запишем вас на прием к психологу" << endl; break;

break;

}

break;

}

}

cout << "С вас 100 рублей" << endl;

}

Благодаря вложенности можно составлять интересные диалоги, например, как здесь.

**Оператор goto**

Оператор goto не является разветвляющим, однако его также можно добавить в этот реферат. Дело в том, что этим оператором не рекомендуется пользоваться и лучше всегда заменять его на «что-то». А условные операторы как раз и являются этим «что-то».

С помощью оператора goto мы можем пропускать фрагмент кода. Сам оператор состоит из ключевого слова goto и имени-метки.

Синтаксис:

Goto link;

Миллион выражений;

Link:

Когда компилятор доходит до строчки с оператором goto, он перестает считывать весь последующий код до момента, пока не дойдет до имени-метки, после которой будет стоять двоеточие.

Однако, оператором goto не рекомендуют пользоваться, так как он очень неудобен в использовании. Да и зачем он нужен, если многие операции с ним можно заметить на операции с условными операторами.

Например:

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int age;

cout << "Введите значение переменной age: ";

cin >> age;

int numb1;

if (age > 0);

goto a;

numb1 = 5;

goto b;

a: numb1 = 7;

b: int numb2 = numb1 + 10;

}

Можно заменить на:

int main() {

setlocale(LC\_ALL, "ru");

int age;

cout << "Введите значение переменной age: ";

cin >> age;

int numb1;

if (age > 0)

numb1 = 5;

else

numb1 = 7;

int numb2 = numb1 + 10;

}

ЕДИНСТВЕННАЯ ситуация, когда использование оператора goto является допустимым, – когда нам нужно выйти из вложенного цикла при получении какого-то определенного условия.

**Вывод:** благодаря условным операторам мы можем исходя из определенного условия выполнять определенный фрагмент кода, а их разнообразие поможет написать максимально грамотный и правильный код.