

Imię i nazwisko	Kierunek	Rok studiów i grupa
Patryk Kozłowski	Informatyka techniczna	1 rok grupa 2
Data zajęć	Numer i temat sprawozdania	
16.10.2024	3. Schematy blokowe (pętle, instrukcje warunkowe, funkcje)	

Logika jest podstawą informatyki i programowania, ponieważ operuje na wyrażeniach, które mogą przyjmować wartości logiczne. Zdanie logiczne jest stwierdzeniem, któremu można przypisać wartość logiczną: PRAWDA lub FAŁSZ. Zazwyczaj zdania logiczne oznacza się literami, np. pp, qq, rr. Przy użyciu spójników logicznych (jak koniunkcja, alternatywa, negacja, implikacja, równoważność), możemy łączyć zdania proste w bardziej złożone struktury.

Negacja ($\neg p$) odwraca wartość logiczną zdania. Jest prawdziwa, gdy zdanie początkowe jest fałszywe.
Koniunkcja ($p \wedge q$) jest prawdziwa tylko wtedy, gdy oba zdania pp i qq są prawdziwe.
Alternatywa ($p \vee q$) jest prawdziwa, gdy przynajmniej jedno ze zdań jest prawdziwe.
Implikacja ($p \Rightarrow q$) jest fałszywa tylko wtedy, gdy pp jest prawdziwe, a qq jest fałszywe.
Równoważność ($p \Leftrightarrow q$) jest prawdziwa, gdy oba zdania są jednocześnie prawdziwe lub fałszywe.

Tautologią nazywamy wyrażenie logiczne, które zawsze jest prawdziwe, niezależnie od wartości logicznych jego składników. Można ją sprawdzić metodą zero-jedynkową, tworząc tabele wartości logicznych.

Kwantyfikatory to symbole używane do opisywania własności zbiorów:

Kwantyfikator ogólny (\forall) wskazuje, że własność dotyczy wszystkich elementów danego zbioru.
Kwantyfikator szczegółowy (\exists) oznacza, że istnieje przynajmniej jeden element, który spełnia daną własność.

W programowaniu, zmienne logiczne (typu bool) mogą przyjmować wartości 0 (FAŁSZ) lub 1 (PRAWDA) i są często wykorzystywane w instrukcjach warunkowych (if), aby podejmować decyzje na podstawie wyników operacji logicznych.

Zadanie 1

```
In [ ]: #include <iostream>

using namespace std;

bool koniunkcja(bool p1, bool p2){
    return p1 && p2;
}
```

```
}

bool alternatywa(bool p1, bool p2){
    return p1 || p2;
}

bool negacja(bool p){
    return !p;
}

bool rownowaznosc(bool p1, bool p2){
    return p1 == p2;
}

bool implikacja(bool p1, bool p2){
    if (!p1 && p2){
        return true;
    }
    return p1 == p2;
}

int main(){
    bool p, q;

    cout << "podaj wartosc logiczna p (1/0):";
    cin >> p;

    cout << "podaj wartosc logiczna q (1/0):";
    cin >> q;

    cout << "koniunkcja wynosi: " << koniunkcja(p,q) << endl;
    cout << "alternatywa wynosi: " << alternatywa(p,q) << endl;
    cout << "negacja p wynosi: " << negacja(p) << endl;
    cout << "negacja q wynosi: " << negacja(q) << endl;
    cout << "implikacja wynosi: " << implikacja(p,q) << endl;
    cout << "rownowaznosc wynosi: " << rownowaznosc(p,q) << endl;

    return 0;
}
```

```
01.cpp zadanie1
➔ 01 git:(master) x ./zadanie1
podaj wartosc logiczna p (1/0):1
podaj wartosc logiczna q (1/0):0
koniunkcja wynosi: 0
alternatywa wynosi: 1
negacja p wynosi: 0
negacja q wynosi: 1
implikacja wynosi: 0
rownowaznosc wynosi: 0
➔ 01 git:(master) x |
```

Zadanie 2

```
In [ ]: // 02.cpp

#include <iostream>
#include "func.hpp"
```

```
using namespace std;

bool p = true;
bool q = true;
bool r = false;

int main(){

    bool a = koniunkcja(p,q);
    bool b = alternatywa(p,q);
    bool c = koniunkcja(negacja(p), alternatywa(p,q));
    bool d = implikacja(koniunkcja(p,r), q);
    bool e = negacja(rownowaznosc(p, alternatywa(q,r)));
    bool f = rownowaznosc(alternatywa(implikacja(p,r),negacja(q)),implikacja(p,r));
    bool g = implikacja(alternatywa(alternatywa(negacja(r), q), negacja(koniunkcja(p,q))), negacja(p));

    cout << "zdanie a wynosi: " << a << endl;

    cout << "zdanie b wynosi: " << b << endl;

    cout << "zdanie c wynosi: " << c << endl;

    cout << "zdanie d wynosi: " << d << endl;

    cout << "zdanie e wynosi: " << e << endl;

    cout << "zdanie f wynosi: " << f << endl;

    cout << "zdanie g wynosi: " << g << endl;

    cout << "podaj p (1/0): ";
    cin >> p;

    cout << "podaj q(1/0): ";
    cin>> q;

    cout << "podaj r(1/0): ";
    cin>> r;

    a = koniunkcja(p,q);
    b = alternatywa(p,q);
    c = koniunkcja(negacja(p), alternatywa(p,q));
    d = implikacja(koniunkcja(p,r), q);
    e = negacja(rownowaznosc(p, alternatywa(q,r)));
    f = rownowaznosc(alternatywa(implikacja(p,r),negacja(q)),implikacja(p,r));
    g = implikacja(alternatywa(alternatywa(negacja(r), q), negacja(koniunkcja(p,q))), negacja(p));

    cout << "zdanie a wynosi: " << a << endl;

    cout << "zdanie b wynosi: " << b << endl;

    cout << "zdanie c wynosi: " << c << endl;

    cout << "zdanie d wynosi: " << d << endl;

    cout << "zdanie e wynosi: " << e << endl;
```

```
    cout << "zdanie f wynosi: " << f << endl;\n\n    cout << "zdanie g wynosi: " << g << endl;\n\n    return 0;\n}
```

```
In [ ]: // func.hpp\n\n#pragma once\n\nbool koniunkcja(bool p1, bool p2);\n\nbool alternatywa(bool p1, bool p2);\n\nbool negacja(bool p);\n\nbool rownowaznosc(bool p1, bool p2);\n\nbool implikacja(bool p1, bool p2);
```

```
In [ ]: // func.cpp\n\nbool koniunkcja(bool p1, bool p2){\n    return p1 && p2;\n}\n\nbool alternatywa(bool p1, bool p2){\n    return p1 || p2;\n}\n\nbool negacja(bool p){\n    return !p;\n}\n\nbool rownowaznosc(bool p1, bool p2){\n    return p1 == p2;\n}\n\nbool implikacja(bool p1, bool p2){\n    if (!p1 && p2){\n        return true;\n    }\n    return p1 == p2;\n}
```

```

→ 02 git:(master) × ls
@ 02.cpp @ func.cpp @ func.hpp @ zadanie2
→ 02 git:(master) × ./zadanie2
zdanie a wynosi: 1
zdanie b wynosi: 1
zdanie c wynosi: 0
zdanie d wynosi: 1
zdanie e wynosi: 0
zdanie f wynosi: 1
zdanie g wynosi: 0
podaj p (1/0): 1
podaj q(1/0): 1
podaj r(1/0): 0
zdanie a wynosi: 1
zdanie b wynosi: 1
zdanie c wynosi: 0
zdanie d wynosi: 1
zdanie e wynosi: 0
zdanie f wynosi: 1
zdanie g wynosi: 0
→ 02 git:(master) × |

```

p	q	r	$p \wedge q$	$p \vee q$	$\neg p$	$\neg p \wedge (p \vee q)$
1	1	1	1	1	0	0
$p \wedge r$	$(p \wedge r) \rightarrow q$	$p \leftrightarrow (q \vee r)$	$\neg(p \leftrightarrow (q \vee r))$	$p \rightarrow r$	$\neg q$	$[(p \rightarrow r) \vee \neg q]$
1	1	1	1	0	1	0
$(r \wedge \neg q)$	$[p \rightarrow (r \wedge \neg q)]$	$[(p \rightarrow r) \vee \neg q]$	$[(p \rightarrow r) \vee \neg q]$	$[(p \rightarrow r) \vee \neg q] \leftrightarrow [p \rightarrow (r \wedge \neg q)]$	$\neg r$	$(\neg r \vee q)$
0	0	1	1	1	1	1
$(q \wedge r)$	$\neg(q \wedge r)$	$[(\neg r \vee q) \vee \neg(q \wedge r)]$	$[\neg(q \rightarrow p)]$	$[(\neg r \vee q) \vee \neg(q \wedge r)] \rightarrow [\neg(q \rightarrow p)]$		
1	0	0	1	0		

Zadanie 3

```

In [ ]: // 03.cpp

#include <iostream>
#include "func.hpp"

using namespace std;

bool p;
bool q;
bool r;

int main(){

    cout << "podaj p (1/0): ";
    cin >> p;

    cout << "podaj q(1/0): ";
    cin >> q;

    cout << "podaj r(1/0): ";
    cin >> r;

```

```
bool z1 = implikacja(implikacja(p || q || r, !p), q || r) && !p;
bool z14 = implikacja(p, !p) || q;
bool z27 = implikacja(implikacja((implikacja(implikacja(implikacja(p, q),
    r), !p), q || r), q || r) && !p, q);

cout << "zdanie logiczne nr 1 wynosi: " << z1 << endl;

cout << "zdanie logiczne nr 14 wynosi: " << z14 << endl;

cout << "zdanie logiczne nr 27 wynosi: " << z27 << endl;

return 0;
}
```

```
In [ ]: // func.cpp

bool koniunkcja(bool p1, bool p2){
    return p1 && p2;
}

bool alternatywa(bool p1, bool p2){
    return p1 || p2;
}

bool negacja(bool p){
    return !p;
}

bool rownowaznosc(bool p1, bool p2){
    return p1 == p2;
}

bool implikacja(bool p1, bool p2){
    if (!p1 && p2){
        return true;
    }
    return p1 == p2;
}
```

```
In [ ]: // func.hpp

#pragma once

bool koniunkcja(bool p1, bool p2);

bool alternatywa(bool p1, bool p2);

bool negacja(bool p);

bool rownowaznosc(bool p1, bool p2);

bool implikacja(bool p1, bool p2);
```

```
@fedora → 03 rbenv:(system) git:(master) x ./zadanie3
podaj p (1/0): 1
podaj q(1/0): 1
podaj r(1/0): 1
zdanie logiczne nr 1 wynosi: 0
zdanie logiczne nr 14 wynosi: 1
zdanie logiczne nr 27 wynosi: 1
```

p	g	r	$p \vee g \vee r$	$p \vee g \vee r \Rightarrow \neg p$	$p \vee g \vee r \Rightarrow \neg p \vee (g \vee r)$	$p \vee g \vee r \Rightarrow \neg p \vee (g \vee r) \wedge \neg p$		
1	1	1	1	1	0	1	0	
1	1	0	1	1	0	1	0	
1	0	1	1	1	0	1	0	
0	1	1	1	1	1	1	1	
1	0	0	1	1	0	1	0	
0	1	0	1	1	1	1	1	
0	0	1	1	1	1	1	1	
0	0	0	0	0	1	0	0	
p	g	r	$p \Rightarrow \neg p$	$p \Rightarrow \neg p \vee g$				
1	1	1	0	1				
1	1	0	0	1				
1	0	1	0	0				
0	1	1	1	1				
1	0	0	0	0				
0	1	0	1	1				
0	0	1	1	1				
0	0	0	1	1				
p	g	r	$(p \Rightarrow g)$	$(p \Rightarrow g) \Rightarrow g$	$((p \Rightarrow g) \Rightarrow g) \Rightarrow r$	$((p \Rightarrow g) \Rightarrow g) \Rightarrow r \Rightarrow (r \Rightarrow p)$	$((p \Rightarrow g) \Rightarrow g) \Rightarrow r \Rightarrow (r \Rightarrow p) \Rightarrow g$	$((p \Rightarrow g) \Rightarrow g) \Rightarrow r \Rightarrow (r \Rightarrow p) \Rightarrow g \Rightarrow p$
1	1	1	1	1	1	1	1	1
1	1	0	1	1	0	1	1	1
1	0	1	0	0	1	1	0	1
0	1	1	1	1	1	0	1	0
1	0	0	0	0	0	1	0	1
0	1	0	1	1	0	1	1	0
0	0	1	1	1	0	1	1	0
0	0	0	1	0	1	0	1	0

Podsumowanie i Wnioski

Repozytorium github link: https://github.com/UmarlyPoeta/pi_sem_1_bin

Zadania wymagające zastosowania operacji logicznych i kwantyfikatorów są kluczowe w analizie i przetwarzaniu informacji w informatyce. Ćwiczenia te mają na celu rozwijanie zdolności do tworzenia i analizy wyrażeń logicznych oraz ich przekształcania w programy komputerowe. Rozwiązanie zadania 1 i zadania 2 rozwija umiejętność programowania operacji logicznych, dzięki czemu łatwiej jest zrozumieć działanie algorytmów warunkowych oraz manipulację danymi w oparciu o wartości logiczne. Z kolei sprawdzenie tautologii i prawdziwości zdań złożonych w zadaniu 3 pozwala zrozumieć istotę dowodzenia poprawności wyrażeń, co jest przydatne w dalszym rozwoju umiejętności analitycznych w informatyce.

Poprzez implementację funkcji, które realizują operacje logiczne, i wywoływanie ich z różnymi wartościami, pogłębiamy zrozumienie fundamentalnych mechanizmów programowania oraz nabywamy kompetencji w zakresie wykorzystania logiki matematycznej w praktycznych zastosowaniach.