**Część teoretyczna**

1. Jeśli w miejscu, gdzie program spodziewa się wyrażenia logicznego, program odczytuje int wartość 0 ( = ”fałsz”) lub 1 ( = ”prawda”), to kod działa. W przypadku innych wartości typu int program zwraca błąd.
2. Pierwszy else jest wewnątrz pierwszego if-a (warunek1), wykona się wtedy, gdy warunek2 i warunek3 nie spełniają się.

Drugi else wykona się wtedy, gdy nie spełnia się warunek1.

Trzeci else nie odnosi się do żadnego if-a. (brak nawiasów klamrowych w if (warunek4).

1. Pętle for i while do sprawdzają warunki pętli przed rozpoczęciem działania. Wyjątkiem z trzech poznanych pętli jest pętla do while, która wykona się co najmniej jeden raz.
2. Na przykład: for (int i = 7; i > 8; i++) {}

Pętla się nie wykona, bo warunek jest sprzeczny. W przypadku, gdy warunek byłby spełniony, for odczyta wartość początkową (int i = 7), wykona pętlę, inkrementuje (i++) i dopiero potem sprawdza, czy warunek jest nadal spełniony dla wykonywania następnego działania wewnątrz pętli.

1. Przykład deklaracji: int i.

Przykład definicji: i = 9.

a) Nie jest możliwe powtarzanie deklaracji w danym zakresie ważności, kompilator zwraca błąd.

b) Jest możliwe powtarzanie definicji w danym zakresie ważności.

c) Deklaracja powołuje do życia obiekt i=o nazwie nnn

1. Operator jednoargumentowy potrzebuje tylko jeden argument dla działania. Przykłady jednoargumentowych operatorów: -, !.

Operator dwuargumentowy stoi między dwoma argumentami. Przykłady dwuargumentowych operatorów: +, -, %, \*, +=, -=, \*=, /, /=.

Jednoargumentowy operator – zmienia znak zmiennej (liczba stanie ujemną).

Dwuargumentowy operator – oznacza odejmowanie liczb.

1. Kompilator nie uzna taką instrukcję za poprawną, bo nie będzie odczytywał case po default. Default musi być na końcu wewnątrz instrukcji switch.
2. Typ danych bool przechowuje wartość logiczną. Może przejmować dwie wartości: prawda (1) i fałsz (0). Nie można zastosować signed.
3. Tak, możemy. Definiując i deklarując zmienną wewnątrz pętli, zmienna „żyje” tylko w tym forze.
4. Pracuje na całkowitych liczbowych typach danych. Przesunięcie bitów w lewo zapełnia bity z prawego brzegu słowa zerami, żeby nie zmieniać wartości liczby binarnej.

Liczba a = 15, rezultat (po przesunięciu bitów liczby a o 2 w lewo) = 60.

1. \*wsk\_norm = \*wsk\_c
2. Znak null oznacza 0. Potrzebujemy w tablicach typu char. Znak null oznacza koniec symboli w tablicy.
3. Pętla nie będzie działała poprawnie, bo w warunku for nie ma inkrementacji lub dekrementacji dla i, więc nic się nie będzie działało wewnątrz pętli.
4. W if (a[6] = b[8]) jest przepisanie wartości tablicy b do tablicy a. Musimy użyć porównanie (==) dwóch elementów tablic.
5. \*(tab[x] + 6). Mamy dostęp do elementu poza zakresem tablicy, ale nie możemy wskazywać na niego.
6. Dowiadujmy się, jaki z elementów będzie bliżej do zerowego elementu tablicy.

**Część programistyczna**

*Zadanie 1*

#include <iostream>

using namespace std;

int main() {

setlocale(LC\_ALL, "polish");

cout << "Bardzo" << endl;

cout << "długi" << endl;

cout << "napis" << endl;

}

*Zadanie 2*

#include <iostream>

#include <conio.h>

#include <iomanip>

using namespace std;

int main() {

double x;

cin >> x;

cout << fixed << setprecision (2) << x + 1;

}

*Zadanie 3*

#include <iostream>

using namespace std;

int main () {

setlocale(LC\_ALL, "polish");

cout << "Napis zawierający różne dziwne znaczki // \ \ $ & %";

}

*Zadanie 4*

#include <iostream>

using namespace std;

int main () {

int x, y;

cin >> x >> y;

if (x > y) cout << x;

if (y > x) cout << y;

if (x == y) cout << x;

}

*Zadanie 5*

double function (unsigned int n) {

double \*w = new double[n];

return \*new double;

}

*Zadanie 6*

int function (unsigned int n, int tab[]) {

for (int i = 0; i < n ; n++) { [i] = 0;}

for (int i = 0; i < n; i++) {tab[i] = i;}

for (int i = 0; i < n; i++) {tab[i] = tab[i] + tab[i];}

for (int i = 0; i < n; i++) {

int a;

a += tab[i];

int srednia = a / n;

tab[i] = srednia;}}