Politechnika Opolska

Wydział Elektrotechniki, Automatyki i Informatyki

Instytut Elektroenergetyki i Energii Odnawialnej

PRACA INŻYNIERSKA

Studia pierwszego stopnia
Stacjonarne

Kierunek studiów
Elektrotechnika

TEMAT PRACY

Aktualizacja platformy programistycznej aplikacji komputerowej stosowanej do planowania pracy równoległej transformatorów

Promotor:
dr hab. inż. Sebastian Borucki, prof. PO

Autor:
Kacper Hoffman
nr albumu: 95009

Opole, Grudzień 2020

Aktualizacja platformy programistycznej aplikacji komputerowej stosowanej do planowania pracy równoległej transformatorów

Streszczenie

W poniższej pracy przedstawiony został proces wykonania aktualizacji programu służącego do obliczania warunków pracy równoległej. Głógnym celem jest stworzenie nowego programu, w którym usunięte zostaną wady starego programu oraz zostaną wprowadzone funkcje ułatwiające użytkowanie programu. W ramach pracy przedstawiono pełną analizę kodu źródłowego programu modernizowanego oraz przeniesienie go na nowy program. W trakcie aktualizacji dokonano zmiany języka programowania z Pascal na C#. Jako sprawdzenie działania nowego programu wykonano przykładowe zadanie, polegające na obliczeniu warunków pracy równoległej zależnie od zmieniającego się obciążenia na obu programach, a następnie porównaniu wyników ich operacji.

Updating the programming platform of a computer application used for planning parallel operation of transformers

Summary

In the following work is presented the process of updating a program used for the calculation of parallel operation conditions of transformers. The main goal is the creation of a new pogram, which will involve the removal of various flaws of the original progam and the introduction of functions that make the use of program easier. As part of the work full analysis of source code was performed, as well as its translation to the new program. During the update the programming language used was changed from Pascal to C#. The functioning of the new program will be tested by performing an example exercise, involving the parallel operation of transformers under changing load, on both programs and comparing the results of their calculations.

Zawartość

1 Wstęp

Systemy elektroenergetyczne są podstawą funkcjonowania wszystkich urządzeń elektrycznych współczesnego świata. Taki system jest definiowany jako zbiór urządzeń, które umożliwiają względnie wydajne wytwarzanie, przesyłanie, rozdzielanie, przekształcanie oraz przechowywanie energii elektrycznej dla wszystkich odbiorników podłączonych do niego. W związku z tym głównymi celami systemu jest dostarczenie wymaganej ilości energii elektrycznej w postaci odpowiedniej mocy, poziomu napięcia, zgodnej częstotliwości, niskiej zawartości wyższych harmonicznych oraz przy jak najkrótszych (idealnie zerowych) przerwach w przesyle. Poza kwestiami zachowania wysokiej jakości przesyłanej energii, głównym problemem systemu jest zapewnienie właśnie takiej niezawodności.

W tym celu stosowane są różne rozwiązania, na przykład systemy zasilania zastępczego, zdalne przełączanie aktywnych linii przesyłowych, przenoszenie obciążenia na wiele źródeł zasilania, czy też stosowanie układów pracy równoległej transformatorów. Sam termin „praca równoległa transformatorów" może być mylący, ponieważ w warunkach pracy prawidłowej nie występuje potrzeba ciągłego stosowania tego typu rozdziału energii, a zastosowanie go w niekorzystnych warunkach może wywołać poważne uszkodzenia urządzeń w takim fragmencie systemu. W praktyce jest to głównie metoda zapewnienia optymalnego zasilenia swojego fragmentu sieci w zależności od zmiennego obciążenia [1].

W trakcie dnia użytkownicy systemu elektroenergetycznego wymuszają na systemie dostarczenie różnych ilości mocy ze względu na uruchamianie i wyłączanie odbiorników. Różnica pomiędzy wymaganiami w środku dnia a w nocy jest bardzo duża, jednak ciągłe stosowanie największych transformatorów dla każdej chwili dnia powoduje duże straty mocy. W związku z tym stosuje się system przełączania zasilania transformatorów, ustawiony w zależności od przewidywanych poziomów obciążenia w danych fragmentach dnia [1]. Oczywiście, praca równoległa transformatorów może też zostać wykorzystana do zapewnienia zwiększenia ciągłości zasilania. Transformatory, tak jak pozostałe elementy systemu, mogą zostać uszkodzone w trakcie prawidłowej pracy. W takiej sytuacji wadliwy transformator musi zostać naprawiony, przez co fragment sieci zasilany przez niego zostałby odcięty od zasilania. Z tego powodu układ równoległy stosuje się również do zapewnienia niezawodności przesyłu energii elektrycznej.

Zapewnienie bezpiecznej pracy równoległej transformatorów polega na spełnieniu warunków pracy równoległej. Są one kolejno: odchyłka napięć pierwotnych i wtórnych mniejsza niż 0.5%, odchyłka napięć zwarcia mniejsza niż 10%, stosunek mocy mniejszy niż 3 (równoznacznie większy niż 1/3) oraz równoważność tzw. grup transformatorów. Te warunki zapewniają, że w trakcie pracy nie popłynie pomiędzy transformatorami szkodliwy prąd wyrównawczy przez nierówności napięć oraz nie nastąpi przeciążanie transformatorów poprzez niedopasowanie mocy znamionowych [2].

2 Cel i zakres pracy

Celem pracy jest wykonanie aktualizacji programu stosowanego w ramach laboratorium elektroenergetyki na Wydziale Elektrotechniki, Automatyki i Informatyki do obliczeń warunków pracy równoległej transformatorów, nazwany tran.exe. Aktualizacja wspomoże proces dydaktyczny na Wydziale poprzez wyeliminowanie znaczących wad aktualnej wersji programu. Program bazowy jest programem konsolowym, który stosuje język programowania Pascal. Programy wykonane w tym języku zwykle stosują jako podstawę oprogramowanie DOS dostępne na starszych systemach Windows. Jednak ze względu na zmiany dokonane w nowych wersjach Windows całkowicie zakończono wspieranie DOS. Efektem tego jest niemożliwość uruchomienia programu tran.exe na nowszych komputerach Wydziału.

Aktualizacja programu polega na pełnym przeniesieniu funkcjonalności programu tran.exe na nowy program o nazwie **Tran 2020**. Stworzony program, oprócz zdolności wykonywania tych samych obliczeń, co program oryginalny, będzie posiadał wiele dodatkowych cech ułatwiających użytkowanie programu. Do dokonanych zmian należy zastąpienie liniowego wprowadzania danych interfejsem oknowym, w którym istnieje możliwość wprowadzania danych w dowolnej kolejności. Kolejną ważną modernizacją jest usunięcie arbitralnego ograniczenia programu tran.exe, przez które niemożliwe jest wyświetlenie wyników obliczeń strat mocy bez drukowania wyników.

Przeniesienie funkcjonalności jest również zmianą używanego języka programowania. W nowszych komputerach często stosowanym językiem programowania jest C#. Posiada on wiele zalet w porównaniu z językiem Pascal. Jest on oficjalnie wspierany przez systemy Windows oraz zintegrowane środowiska programistyczne. Aktualizację ułatwi fakt, że podobnie do Pascalowego podziału programu na procedury C# dzieli program na funkcje. Dodatkowo wiele środowisk pozwala na graficzne modelowanie samego programu.

Analiza kodu programu tran.exe umożliwi pełne zrozumienie jego działania. Dzięki temu podejściu wykonanie programu Tran 2020 będzie znacznie ułatwione – wiele funkcji programu zmodernizowanego będzie analogicznych do procedur programu oryginalnego. Po wykonaniu nowego programu nastąpi przetestowanie jego zdolności. Dla porównania obu programów należy na nich wykonać to samo zadanie przykładowe. Jeśli wyniki obliczeń będą identyczne, można stwierdzić, że aktualizacja części obliczeniowej jest udana. Z kolej sprawdzenie nowych opcji umożliwianych przez Tran 2020 pokaże zwiększoną łatwość użytkowania i przystosowanie do działania na nowych komputerach z systemem Windows.

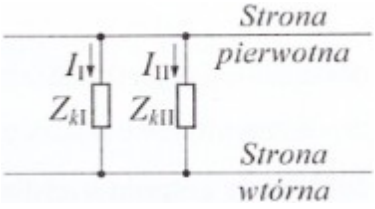
3 Praca równoległa transformatorów

Problematyka pracy równoległej transformatorów stosowana jest we wszystkich przypadkach, gdy używanie jednego transformatora nie jest wystarczające lub pożądane do wypełnienia potrzeb zasiłowych. Dobrym przykładem jest odniesienie do zmiennego obciążenia sieci. W praktyce występuje to we wszystkich sieciach, zwłaszcza miejskich. Do sieci podłączone są odbiorniki o pewnej sumie mocy znamionowych, która określa największą potrzebną moc transformatorów zasilających. Jednak nie wszystkie te odbiorniki są zasilane jednocześnie.

Rano występuje mały szczyt przed rozpoczęciem pracy w budynkach biurowych, przemysłowych itp. Potem następuje zwiększanie obciążenia sieci aż do szczytu południowego, gdy większość budynków pracy działa przy pełnej sprawności. Potem obserwuje się zmniejszenie obciążenia wraz z powrotem pracowników do domów. Wieczorny szczyt obciążenia wynika z wykonywaniu przez mieszkańców potrzebnych prac domowych, ale jest on bardzo mały w porównaniu z południem. W nocy z kolej obciążenie sieci osiąga minimum, ponieważ znaczna większość urządzeń jest wyłączona [1].

Zapewnienie bezpiecznej pracy równoległej transformatorów wymaga upewnienia się czy transformatory są wystarczająco zbliżone do siebie, czyli czy posiadają podobne parametry konstrukcyjne. Jeśli transformatory posiadają różne wartości napięć pierwotnych i wtórnych (w praktyce różnica większa niż 0.5%) lub przesunięcia godzinowego (nawet 1 godzina przesunięcia jest nieakceptowalna) to pomiędzy nimi będzie przepływał prąd wyrównawczy [2]. Taki prąd bardzo łatwo może uszkodzić transformatory ze względu na swoją duża wartość. Z drugiej strony, jeśli napięcia zwarć transformatorów różnią się o więcej niż 10% lub moce znamionowe różnią się bardziej niż trzykrotnie, to moc obciążenia sieci rozłoży się bardzo nierównomiernie na transformatorach. W najlepszym przypadku powoduje to zwiększenie strat mocy, a w najgorszym transformatory o mniejszej mocy zostaną przeciążone ponad próg uszkodzeń wewnętrznych [3].

Same obliczenia są kwestią zrozumienia układu połączenia równoległego ze strony teoretycznej [4]. Pracę równoległą transformatorów można analizować jako połączenie równoległe ich mocy znamionowych. Przez owe transformatory przepływają prądy zależne do ich mocy.



Rys. 3.1. Schemat teoretyczny pracy równoległej transformatorów [4]

Ze względu na to, że pracujące równolegle transformatory muszą mieć równe napięcia znamionowe, stosunek impedancji znamionowych transformatorów jest odwrotnie proporcjonalny do stosunku prądów przepływających przez nie [5].

$$\frac{Z_{k2}}{Z_{k1}}=\frac{I_1}{I_2} \tag{3.1}$$

Obliczenia kontynuuje się poprzez wprowadzenie po stronie mocy prądów znamionowych tych transformatorów. Iloczyn impedancji i prądów daje wartość napięć na tychże transformatorach.

$$\frac{I_1}{I_2}=\frac{Z_{k2}}{Z_{k1}}\frac{I_{N2}}{I_{N1}}\frac{I_{N1}}{I_{N2}} \tag{3.2}$$

$$\frac{I_1}{I_2}=\frac{U_{k2}}{U_{k1}}\frac{I_{N1}}{I_{N2}} \tag{3.3}$$

Z własności definicyjnej transformatorów stosunek prądów jest równy stosunkowi mocy [5]. Wprowadzenie wartości S równej sumie mocy znamionowej transformatorów możliwe jest przekształcenie wzoru na moc jednego z transformatorów.

$$\frac{S_1}{S_2}=\frac{U_{k2}}{U_{k1}}\frac{S_{N1}}{S_{N2}} \tag{3.4}$$

$$\frac{S_1}{S-S_1}=\frac{U_{k2}}{U_{k1}}\frac{S_{N1}}{S_{N2}} \tag{3.5}$$

$$\frac{S-S_1}{S_1}=\frac{U_{k1}}{U_{k2}}\frac{S_{N2}}{S_{N1}} \tag{3.6}$$

$$S-S_1=S_1\left(\frac{U_{k1}}{U_{k2}}\frac{S_{N2}}{S_{N1}}\right) \tag{3.7}$$

$$S=S_1\left(\frac{U_{k1}}{U_{k2}}\frac{S_{N2}}{S_{N1}}\right)+S_1 \tag{3.8}$$

$$S=S_1\left(\frac{U_{k1}}{U_{k2}}\frac{S_{N2}}{S_{N1}}+1\right) \tag{3.9}$$

$$S=S_1\left(\frac{U_{k1}S_{N2}}{U_{k2}S_{N1}}+\frac{U_{k2}S_{N1}}{U_{k2}S_{N1}}\right) \tag{3.10}$$

$$S=S_1\left(\frac{U_{k1}S_{N2}+U_{k2}S_{N1}}{U_{k2}S_{N1}}\right) \tag{3.11}$$

$$S_1=S\left(\frac{U_{k2}S_{N1}}{U_{k1}S_{N2}+U_{k2}S_{N1}}\right) \tag{3.12}$$

Podzielenie przez napięcia zwarcia pozwoli na rozdzielenie części zależnej od parametrów transformatora, dla którego moc obliczamy od parametrów pozostałych transformatorów. Jest to ważne, ponieważ powyższe operacje są prawdziwe dla połączeń równoległych wielu transformatorów. Wystarczy zastąpić fragment wzoru na sumę, a możliwe staje się obliczenie mocy obciążeniowych kolejnych transformatorów [4].

$$S_1=S\left(\frac{S_{N1}}{\frac{U_{k1}S_{N2}}{U_{k2}}+S_{N1}}\right) \tag{3.13}$$

$$S_1=S\left(\frac{\frac{S_{N1}}{U_{k1}}}{\frac{S_{N2}}{U_{k2}}+\frac{S_{N1}}{U_{k1}}}\right) \tag{3.14}$$

$$S_i=S\left(\frac{\frac{S_{Ni}}{U_{ki}}}{\sum_{j=1}^{n}\frac{S_{Nj}}{U_{kj}}+\frac{S_{Ni}}{U_{ki}}}\right) \tag{3.15}$$

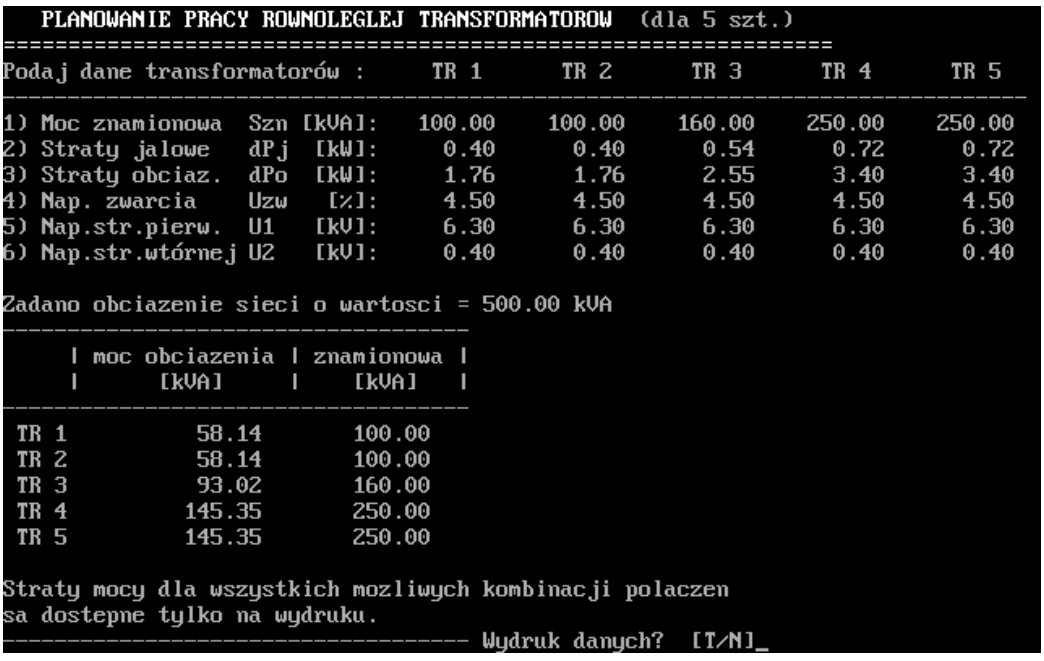
Obliczenia strat mocy są z jednej strony prostsze, ale z drugiej strony bardziej czasochłonne. Wartości strat pojawiają się w trakcie bilansu mocy transformatorów. Dodanie do siebie sumy strat jałowych transformatorów, które są relatywnie na stałym poziomie oraz sumy strat obciążeniowych odniesionych do obciążenia sieci otrzymuje się straty mocy czynnej dla danej kombinacji transformatorów. To jest główny cel wykonywania obliczeń cyfrowo – ilość kombinacji wzrasta ekspotencjalnie do ilości transformatorów. W przypadku dużych układów kombinacji może być nawet kilkaset.

$$dS_k=\sum_{i=1}^{n}P_{ji}+\sum_{i=1}^{n}P_{oi}\left(\frac{S_{zad}}{\sum_{i=1}^{n}S_{ni}}\right)^2 \tag{3.16}$$

4 Charakterystyka aktualnie stosowanego oprogramowania

Aktualnie w ramach zajęć laboratorium elektroenergetyki do analizy pracy równoległej transformatorów stosuje się program **tran.exe**. Jest to program konsolowy napisany w języku programowania Pascal. Jako program konsolowy wyświetla on dane w konsoli Windows w postaci tekstowej. Użytkowanie programu wymaga wpisywania danych w odpowiedniej kolejności, która została określona na stałe w kodzie programu. W związku z tym praca na programie tran.exe jest bardzo linearna i nie pozwala na szybką poprawę poprzednio wpisanych wartości.

Po uruchomieniu jedyną widoczną częścią interfejsu jest podkreślone wyrażenie „Planowanie Pracy Równoległej Transformatorów" oraz prośba o podanie ilości transformatorów. Po podaniu owej ilości zostaje wyświetlona górna połowa interfejsu programu – tabela, do której należy po kolej wpisywać dane transformatorów. Ta część funkcjonowania programu jest najbardziej ograniczona, ponieważ po wpisaniu danej niemożliwe jest zmienienie jej wartości. Wpisanie danych powoduje kolejną prośbę do użytkownika o podanie obciążenia sieci, wraz z wartością maksymalną. Wpisanie jej wyświetla dolną część interfejsu, czyli tabelę zawierającą porównanie mocy obciążeniowych transformatorów porównaną z ich mocami znamionowymi.



Rys. 4.1. Główny ekran programu tran.exe po wpisaniu danych i wykonaniu wstępnych obliczeń [opracowanie własne]

Użytkownik zostaje poinformowany o tym, że obliczone straty mocy są dostępne tylko na wydruku. To kolejna znacząca wada programu, ponieważ ten warunek jest nierealny z perspektywy wielu użytkowników – wymuszanie drukowania w celu dostępu do kluczowych danych oznacza marnowanie papieru przy każdym wykonanym obliczeniu. Wybranie opcji „T" powoduje wydrukowanie obliczonych danych, jeśli podłączona drukarka wspierają zastosowaną przez program metodę automatyzacji drukowania (szczegółowo wytłumaczona w rozdziale 4.3). Z drugiej strony opcja „N" umożliwia wykonanie kolejnego obliczenia przy zmienionych danych. Podstawową możliwością jest wpisanie nowej wartości obciążenia sieci. Druga możliwość to odłączenie transformatora od funkcjonowania. Odłączenie powoduje, że dany transformator zostaje zignorowany w trakcje kolejnych obliczeń. Zakończenie pracy programu polega na wejściu w opcje odłączania, a następnie wciśnięciu przycisku „Enter". Ta metoda jest bardzo niekonwencjonalna, przez co nie zostanie ona znaleziona przez użytkowników nieplanujących obliczeń z odłączaniem transformatorów.

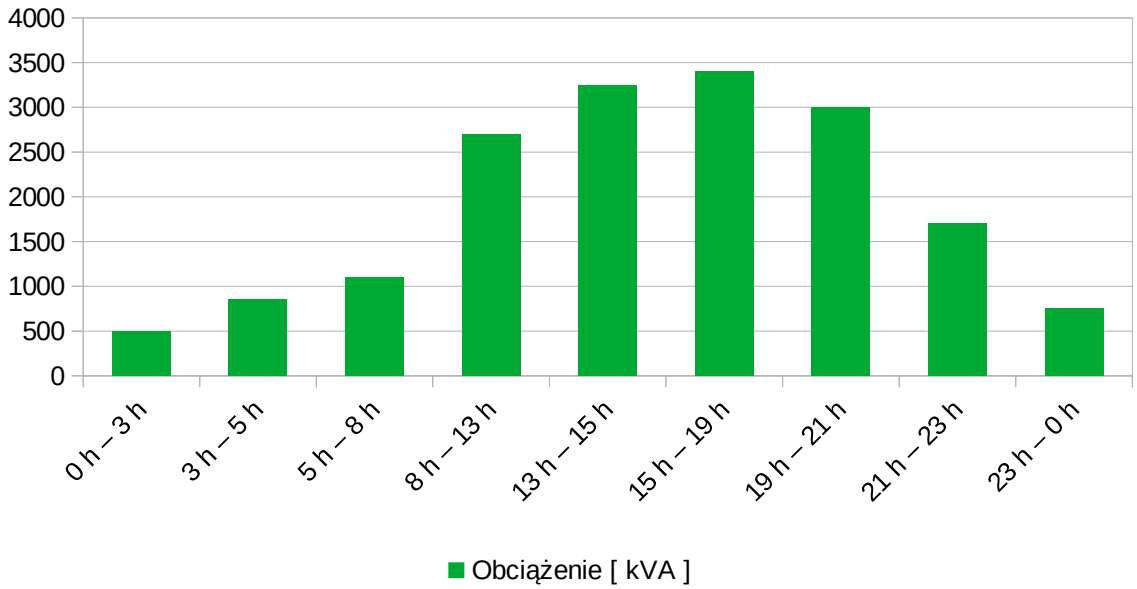
Z samego opisu jest oczywiste, że program wymaga aktualizacji, aby móc nadal wykonywać obliczenia warunków pracy równoległej na współczesnych komputerach. Przykładowo, Rys. 4.1. był możliwy do wykonania tylko przy pomocy zewnętrznego programu emulacyjnego DOS. Próba bezpośredniego uruchomienia programu na komputerach o systemach operacyjnych nowszych od Windows XP wywoła tylko komunikat błędu. Konsolowa natura programu wymusza na użytkowniku liniowe wpisywanie danych, co z kolej bardzo utrudnia poprawę wprowadzonych danych. Na koniec niemożliwość dostępu do strat mocy bez drukowania automatycznie zmniejsza wartość wykonanych obliczeń o koszty związane z zużyciem papieru (co i tak zakłada, że użytkownik posiada drukarkę zdolną do wykonania rozkazu drukowania przez program).

Poniżej przedstawiono przykład zastosowania programu do wykonania zadania analizy pracy równoległej przy pomocy programu tran.exe. W przyszłym rozdziale to zadanie zostanie wykonane na zaktualizowanym programie Tran 2020, aby porównać zdolności oraz łatwość obsługi obu programów.

4.1 Przykład pracy równoległej

Załóżmy, że chcemy wykonać analizę pracy równoległej transformatorów dla zmiennego obciążenia. Takie zagadnienie dotyczy przykładowo układu zasilającego miasto. W mieście obciążenie sieci zmienia się zależnie od pory dnia – rano obciążenie wynika głównie z porannego przygotowania do pracy, w godzinach roboczych osiąga maksimum przez zasilanie urządzeń wymaganych do funkcjonowania ekonomii, wieczorem spada do poziomu wymuszanego przez domy a w nocy osiąga minimum.

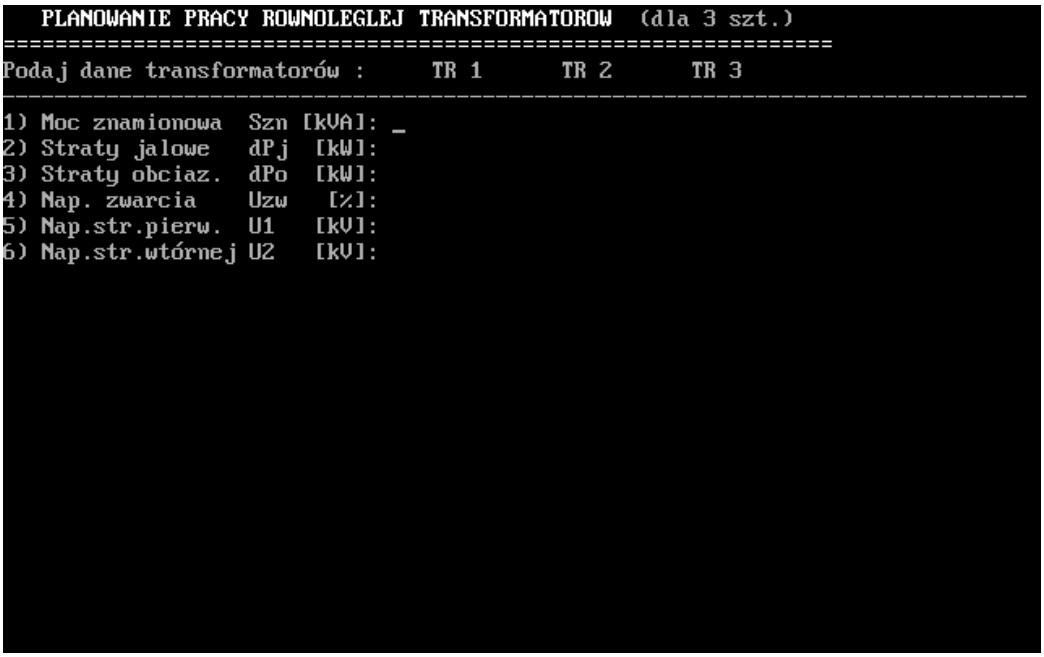
W związku z tym rozdzielnia przygotowuje transformatory, których suma mocy znamionowych jest większa od przewidywanego obciążenia maksymalnego. Jednak minimalizacja strat mocy dla każdej części dnia jest kwestią wymagającą obliczeń [1]. W tym zadaniu obciążenia sieci w kolejnych porach dnia zostały przedstawione na poniższym wykresie.



Rys. 4.1.1. Wykres obciążenia sieci w zależności od pory dnia [opracowanie własne]

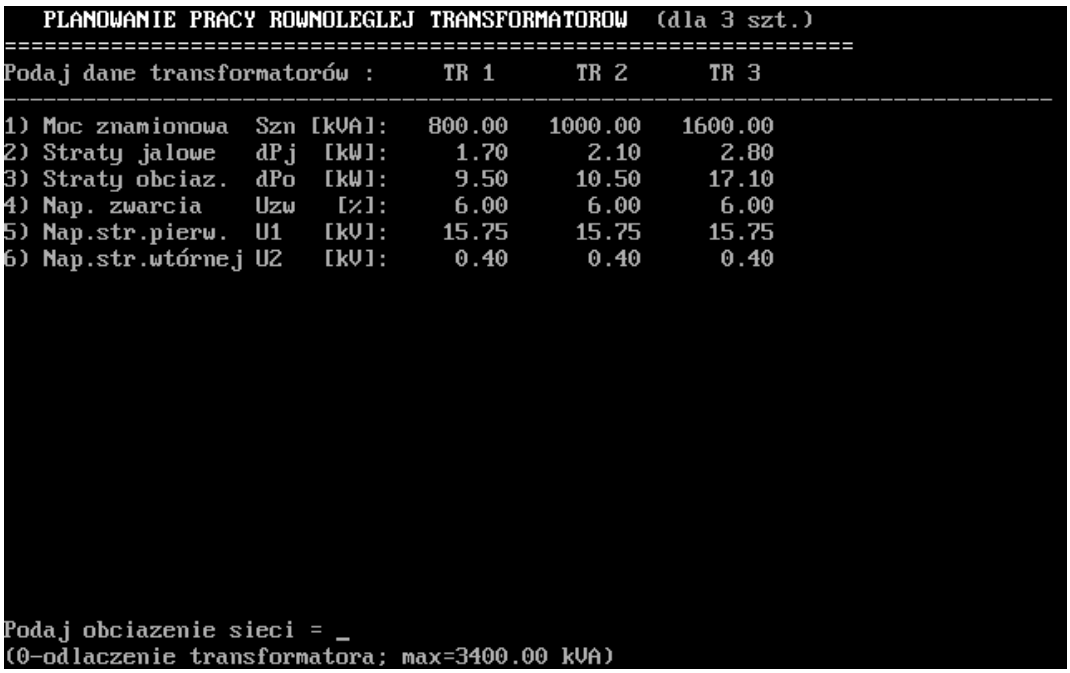
Tutaj maksymalne obciążenie wynosi 3400 kVA. W związku z tym w przykładowej rozdzielni zastosowano trzy transformatory: TAOb–800/15, TAOb–1000/15 oraz TAOb–1600/15. Ich suma mocy znamionowych to 3400 kVA, co jest wystarczające do zasilenia sieci. Pierwszą częścią zadania jest uruchomienie programu tran.exe oraz wpisanie wymaganej ilości transformatorów.

Rys. 4.1.2. Ekran programu tran.exe tuż po uruchomieniu [opracowanie własne]

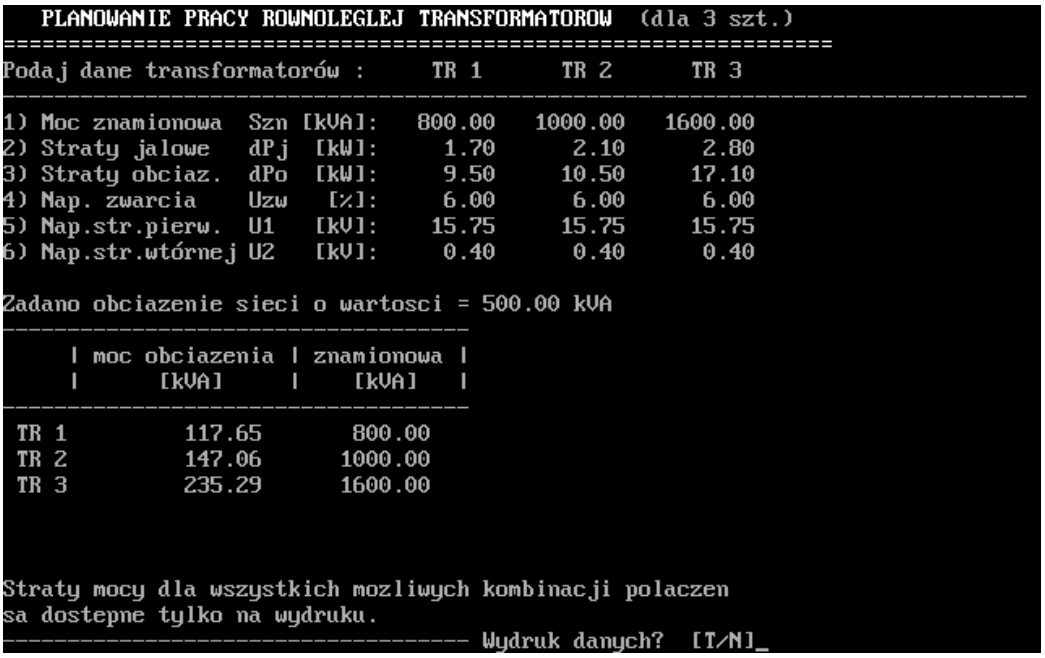


Rys. 4.1.3. Ekran programu po wpisaniu liczby transformatorów [opracowanie własne]

Program jest przygotowany do wpisania danych kolejnych transformatorów. Kolejność wpisywania jest wymuszona przez program – najpierw należy wpisywać poszczególne dane transformatora pierwszego od góry do dołu, potem drugiego a na koniec trzeciego. Wpisanie danych umożliwia wprowadzenie żądanego obciążenia. W tym przypadku pierwszą wartością obciążenia jest 630 kVA, co może zostać wypełnione przez jeden transformator TAOb-1000/15.



Rys. 4.1.4. Ekran programu po wpisaniu danych transformatorów [opracowanie własne]



Rys. 4.1.5. Ekran programu po wpisaniu pierwszego obciążenia [opracowanie własne]

Jak przewidywano, moc obciążenia sieci rozłożyła się proporcjonalnie do mocy znamionowych transformatorów. Dla przykładu prawie połowa obciążenia została na transformatorze TAOb–1600/15, ponieważ jego moc znamionowa jest bliska połowy mocy całkowitej. Wartości strat mocy są dostępne dopiero po wykonaniu wydruku, dlatego zostaną przedstawione w rozdziale 4.2. Na teraz możliwe jest wykonanie obliczeń mocy obciążeniowych dla pozostałych obciążeń sieci.

Tab. 4.1.1. Zestawienie wyników obliczeń mocy obciążeniowych [opracowanie własne]

Obciążenie Sieci	Obciążenie TAOb–800/15	Obciążenie TAOb–1000/15	Obciążenie TAOb–1600/15
[kVA]	[kVA]	[kVA]	[kVA]
500	117.65	147.06	235.29
850	200	250	400
1100	258.82	323.53	517.65
2700	635.29	794.12	1270.59
3250	764.71	955.88	1529.41
3400	800	1000	1600
3000	705.88	882.35	1411.76
1700	400	500	800
750	176.47	220.59	352.94

Wykonane obliczenia w pełni potwierdzają teoretyczne przypuszczenia o pracy równoległej tego układu. Transformatory przejmują moc obciążeniową proporcjonalnie do swoich mocy znamionowych przy każdym obciążeniu. Jednak określenie, która kombinacja daje najmniejsze straty dla danego obciążenia jest nieznana do czasu wykonania wydruku.

4.2 Obsługa drukowania

W programie tran.exe drukowanie bazuje na przestarzałej technice wykorzystującej nazwę pliku „PRN" [6]. Ta technika jest dokładniej wytłumaczona w kolejnym rozdziale, ale w skrócie polega ona na wykorzystaniu plików urządzenia. W komputerach zawierających stary DOS pozwala to na wysłanie rozkazu drukowania bezpośrednio do drukarki, ale w nowych systemach nie jest to możliwe. Funkcjonalność plików urządzeń została kilkukrotnie przerobiona w nowych systemach. Z tego powodu nawet, jeśli program zostanie otwarty dzięki programowy emulującemu DOS drukowanie działa tylko na komputerach nieaktualizowanych.

PLANOWANIE PRACY ROWNOLEGLEJ TRANSFORMATOROW

Dane transformatorów			TR 1	TR 2	TR 3
1) Moc znamionowa	Szn	[kVA]:	800.00	1000.00	1600.00
2) Straty jalowe	dPj	[kW]:	1.70	2.10	2.80
3) Straty obciaz.	dPo	[kW]:	9.50	10.50	17.10
4) Nap. zwarcia	Uzw	[%]:	6.00	6.00	6.00
5) Nap.str.pierw.	U1	[kV]:	15.75	15.75	15.75
6) Nap.str.wtórnej	U2	[kV]:	0.40	0.40	0.40

Zadano obciazenie sieci o wartosci = 500.00 kVA

	moc obciazenia [kVA]	znamionowa [kVA]
TR 1	117.65	800.00
TR 2	147.06	1000.00
TR 3	235.29	1600.00

MOZLIWE WARIANTY PRACY

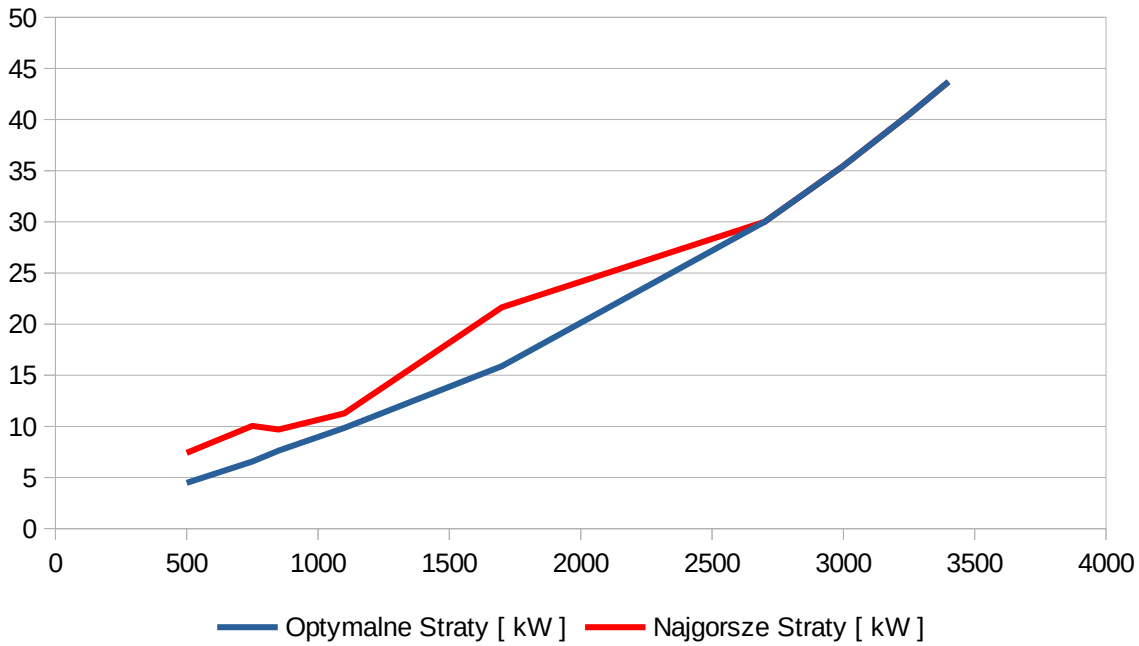
Kombinacje polaczen	Straty mocy [kW]	Moc znamionowa [kVA]
TR 1	5.41	800.00
TR 2	4.72	1000.00
TR 1 2	5.34	1800.00
TR 3	4.47	1600.00
TR 1 3	5.65	2400.00
TR 2 3	5.92	2600.00
TR 1 2 3	7.40	3400.00

Rys. 4.2.1. Fragment skanu wydrukowanych wyników obliczeń [opracowanie własne]

Wydruk pozwala na wykonanie analizy strat mocy poszczególnych kombinacji. W tym przypadku optymalne straty nastąpiły dla podłączenia transformatora TAOb–1600/15, a najwyższe straty odpowiadają włączeniu wszystkich trzech transformatorów jednocześnie. To wynik zrozumiały ze względu na niskie obciążenie sieci, ale dla pozostałych nie jest to oczywiste.

Tab. 4.2.1. Zestawienie wyników obliczeń start mocy [opracowanie własne]

Obciążenie Sieci	Optymalna Kombinacja	Optymalne Straty	Najgorsza Kombinacja	Najgorsze Straty
[kVA]	[–]	[kW]	[–]	[kW]
500	3	4.47	1 + 2 + 3	7.4
850	3	7.63	2	9.69
1100	2 + 3	9.84	1 + 2	11.27
2700	1 + 2 + 3	30	1 + 2 + 3	30
3250	1 + 2 + 3	40.5	1 + 2 + 3	40.5
3400	1 + 2 + 3	43.7	1 + 2 + 3	43.7
3000	1 + 2 + 3	35.48	1 + 2 + 3	35.48
1700	1 + 2 + 3	15.88	1 + 2	21.64
750	3	6.56	1	10.05



Rys. 4.2.2. Wykres porównawczy strat optymalnych i najgorszych zależnie od obciążenia sieci [opracowanie własne]

Straty mocy są kwestią bardziej skomplikowaną niż obciążenie transformatorów, co jest widoczne na uzyskanych wynikach. Dla mocy obciążeniowych większych od sumy mocy dwóch transformatorów możliwa jest tylko kombinacja wszystkich trzech, ale i wtedy strata mocy zależy od obciążenia. Przypadek obciążenia 1700 kVA pokazuje że nie zawsze dodanie transformatora do pracy równoległej zwiększa straty mocy. Jednak przede wszystkim wykonane obliczenia pokazują prawidłową kolejność załączania oraz odłączania transformatorów dla kolejnych pór dnia [1].

4.3 Analiza kodu programu tran.exe

Ponieważ Pascal jest językiem proceduralnym analiza kodu tego programu została podzielona na procedury zawarte w nim. Kod programu zawiera również komentarze, których znaczenie zostanie wyjaśnione w miarę potrzeby. Na koniec zostanie omówiony fragment wykonawczy, aby pokazać jak te procedury są wywoływane w samym programie. Na początku kodu znajduje się deklaracja programu i zmiennych:

Listing 4.3.1. Deklaracja programu i zmiennych

```
1    program TRAN;  {planowanie pracy rownoleglej transformatorow}
2
3    uses Crt;
4
5    const
6      iT: Byte = 5;  {ilosc transform. max 5}
7    var
8      dT: array[1..5,0..6] of Real; {dane transformatorow}
9      So: array[1..5] of Real;      {moc obciazenia}
10     K,          {przyrost mocy}
11     dPn,         {straty mocy dowolnej liczby transformatorow}
12     Szad,        {zadane obciazenie sieci}
13     Smax: Real;  {max moc wybr. ilosci transformatorow}
14     dr: Text;
```

Sam program jest deklarowany komendą *program TRAN;*. Komenda *uses Crt;* oznacza, że w tym programie została użyta biblioteka Crt. Ta biblioteka zajmuje się obsługą wydarzeń związanych z klawiaturą, kursorem, ekranem i głośnikiem. Innymi słowy jest to biblioteka wejścia/wyjścia. Funkcje należące do tej biblioteki pokażą się wielokrotnie w następnych procedurach. Maksymalna ilość transformatorów *iT* została zdefiniowana jako stały bajt. Odpowiada to dobrej metodyce programowania, żeby często używaną wartość zapisywać jako nazwaną stałą. Pozostałe zmienne są dosyć dobrze opisane komentarzami. Wymagane dane transformatorów *dT* można wydedukować z późniejszej procedury *WpiszDaneTr*; są to kolejno moc znamionowa, straty jałowe, straty obciążenia, napięcie zwarcia, napięcie strony pierwotnej oraz napięcie strony wtórnej. Moce obciążeniowe transformatorów *So* zostaną obliczone w ramach procedury *MocObc*. Przyrost mocy *K* oraz straty mocy *dPn* są pewną tajemnicą. Nie zostały one użyte w kodzie nawet raz. Być może jest to artefakt z poprzedniej wersji programu, która uwzględniała zmienne warunki obciążenia transformatorów. Zadane obciążenie sieci *Szad* jest używane do potwierdzenia dobrych warunków zasilania oraz obliczenia mocy obciążeniowej. Ta moc musi być mniejsza od sumy wszystkich mocy znamionowych badanych transformatorów, czyli mocy maksymalnej *Smax*. Na koniec mamy zmienną tekstową *dr*. Mimo że nie została skomentowana ona odpowiada za obsługę drukarki. Stare drukarki mogły bezpośrednio drukować zmienne Text, co będzie pokazane w procedurach *DrukDanychWe* oraz *MocObc*.

Listing 4.3.2. Funkcja UstalIloscTransf

```
1     function UstalIloscTransf: Byte;
2       var i: Byte;
3       begin
4       ClrScr;
5       HighVideo;
6       Writeln('   PLANOWANIE PRACY ROWNOLEGLEJ TRANSFORMATOROW');
7       Writeln('==================================================
      ============');
8       Writeln;
9       LowVideo;
10      Window(1,25,80,25);
11      Write('Podaj ilosc transformatorow (max 5): ');
12      Window(38,25,50,25);
13      {$I-}
14      repeat
15        ClrScr;
16        Readln(i);
17      until (IOResult=0) AND (i in[2..5]);
18      {$I+}
19      Window(1,25,80,25); ClrScr;
20      Window(1,1,80,25);
21      GoToXY(50,1); Write('(dla ',i,' szt.)');
22      UstalIloscTransf:=i;
23      end;
```

Funkcja *UstalIloscTransf* jest pierwszą wykonywaną funkcją w programie. Zaczyna się od deklaracji zmiennej *i* która zostanie użyta w późniejszej pętli. Komenda *ClrScr* służy do wyczyszczenia okna konsoli. Nie jest to pokazane bezpośredno, ale ta komenda również okno konsoli otwiera. Komenda *HighVideo* uruchamia tryb podkreślonego tekstu. Przez to kolejne teksty, wypisane przez *Writeln*, będą lepiej widoczne dla użytkownika. Po wyświetleniu nagłówka *LowVideo* wyłącza podświetlanie tekstu. Komenda *Window* tworzy nowe okno wewnątrz okna konsoli zaczynając od współrzędnych (1,25) do (80,25), co jest tak naprawdę linią podkreślającą nagłówek. Podobnie zostanie podkreślona prośba o podaniu ilości transformatorów. *{$I-}* oraz *{$I+}* są dyrektywami informującymi kompilator, żeby w danym fragmencie kodu nie sprawdzał stanu wejść i wyjść. Jest to krótsza alternatywa do dyrektyw *{$IOCHECKS OFF}* oraz *{$IOCHECKS ON}* [7]. Jest to wykonywane ze względu na pętlę, która się znajduje wewnątrz. Następuje w niej wpisywanie przez użytkownika ilości transformatorów. Gdyby sprawdzanie wejść/wyjść było włączone i użytkownik wpisałby coś nieprawidłowego, na przykład litery, znaki czy też kod Pascal, to zostałoby wyłapane przez system sprawdzania i nastąpiłoby wyłączenie programu. Natomiast bez sprawdzania każdy możliwy błąd zostanie wpisany w *IOResult*. Wartość *IOResult* wynosi 0 jeśli nie nastąpił żaden błąd we wpisywaniu wartości. Jeśli tak jest, a wartość wpisana jest w przedziale od 2 do 5, pętla się zamyka i sprawdzanie wejścia/wyjścia jest włączone. Ekran zostaje wyczyszczony i powstaje nowe okno od (1,1) do (80,25). Kursor przechodzi do punktu (50,1) za pomocą komendy *GoToXY* (jest to obok nagłówka), zostaje tam wpisany tekst „(dla i szt.)” i zwraca się wpisaną ilość transformatorów.

Listing 4.3.3. Procedura WpiszDaneTr

```
procedure WpiszDaneTr;

      function DobreDane(n: Byte): Boolean;
        var i: Byte;
        begin
        DobreDane:=FALSE;
        if n>1 then
          begin
                            if   dT[n-1,5]<>dT[n,5]   then   Exit;
{spr.nap.str.pierwotnej}
                            if   dT[n-1,6]<>dT[n,6]   then   Exit;
{spr.nap.str.wtornej}
          for i:=1 to n-1 do
            begin
             if (dT[i,1]/dT[n,1])>3 then Exit;        {stos.mocy
znamion.}
             if (dT[i,1]/dT[n,1])<(1/3) then Exit;
             end;
          end;
        DobreDane:=TRUE;
        end;

  var i: Byte;
  begin
  Window(1,3,80,11); ClrScr;
  Write('Podaj dane transformator¢w :');
  GoToXY(30,WhereY);
  for i:=1 to iT do Write('    TR ',i,'  ');
  Writeln;
  Writeln('---------------------------------------',
          '---------------------------------------');
  Writeln('1) Moc znamionowa  Szn [kVA]: ');  {dT[n,1]}
  Writeln('2) Straty jalowe   dPj  [kW]: ');  {dT[n,2]}
  Writeln('3) Straty obciaz.  dPo  [kW]: ');  {dT[n,3]}
  Writeln('4) Nap. zwarcia    Uzw   [%]: ');  {dT[n,4]}
  Writeln('5) Nap.str.pierw.  U1   [kV]: ');  {dT[n,5]}
  Writeln('6) Nap.str.wt¢rnej U2   [kV]: ');  {dT[n,6]}
  i:=1;
  repeat
    WpiszDaneNTr(i);
    if DobreDane(i) then
      Inc(i)
    else
      begin
      Window(1,24,80,25);
      ClrScr; Inc(TextAttr,Blink);
         Writeln('Nie sa spelnione warunki pracy r¢wnoleglej!
');
```

```
45          Dec(TextAttr,Blink);
46          Write('kontynuowac wprowadzanie danych [T/N]?');
47          if UpCase(ReadKey)='N' then Halt;
48          ClrScr;
49          Window(1,3,80,25);
50          end;
51      until i>iT;
52      end;
```

Następną w kolejności jest procedura *WpiszDaneTr*. Nietypowo, zaczyna się ona od deklaracji wewnętrznej funkcji *DobreDane*. Mogło to zostać wykonane by upewnić się, że nie zostanie ona wywołana poza tą procedurą. *DobreDane* jest funkcją typu Boolean, co oznacza, że zwraca wartość prawda/fałsz i służy przede wszystkim do jednoczesnego sprawdzenia kilku warunków. Definiuje się wewnętrzną zmienną *i*, po czym zakłada się, że domyślnym wynikiem tej funkcji jest fałsz. To podejście stosuje się, jeśli niespełnienie warunków skutkuje wyjściem z funkcji komendą *Exit*. Jest to równoznaczne z robieniem warunków *else*, które przypisują wartość fałsz, ale ta metoda minimalizuje ilość potrzebnych linii kodu. Pierwszy warunek sprawdza, czy podana wartość *n* jest większa niż 1. Wewnątrz funkcji *WpiszDaneTr* ta procedura jest wywoływana w pętli, dlatego pierwszy transformator nie wchodzi do pętli i od razu ma przypisaną prawdę jako wynik procedury *DobreDane*. Jednak każdy kolejny transformator musi być sprawdzony zgodnie z warunkami pracy równoległej. Jeśli posiadają one różne napięcia strony pierwotnej czy strony wtórnej, nie mogą one pracować równolegle i funkcja kończy się z wartością fałsz. Z drugiej strony stosunki mocy znamionowych muszą być w odpowiednich granicach. Stosunek mocy większej do mniejszej musi być mniejszy niż 3. Nie trzeba jednak sprawdzać, która moc jest większa, bo szybciej wychodzi dodać kolejny warunek: stosunek mocy znamionowych musi być większy od 1/3 i mniejszy od 3. Jeśli zostały one spełnione, *DobreDane* ustawiają się na prawdę i funkcja się kończy. Dopiero tutaj zaczyna się procedura *WpiszDaneTr*.

Tworzy się nowe okno dla wpisywania danych kolejnych transformatorów. Obok prośby o podanie danych transfornatorów, na poziomie kursora podanym przez *WhereY*, pokazują się kolejne numery transformatorów, a poniżej zostają wypisane kolejne nazwy danych które mają zostać wpisane przez użytkownika. W pętli zostaje wywołana procedura *WpiszDaneNTr* (opisana poniżej), a potem zostaje wywołana funkcja *DobreDane* by sprawdzić, czy wpisane dane transformatora pozwalają na pracę równoległą. Jeśli tak, *i* zwiększa się o 1 za pomocą *Inc* i zostaje sprawdzony kolejny transformator. Jeśli nie, tworzy się ostrzeżenie. Powstaje nowe okno o współrzędnych (1,24) i (80,25). Zwiększa się wartość atrybutu tekstowego *Blink* (jest on typu Boolean, więc to jest równoznaczne z uruchomieniem migania), wypisuje tekst informujący o niespełnionych warunkach, po czym miganie jest wyłączone zmniejszeniem atrybutu *Blink* [7]. Na koniec dodaje się pytanie o kontynuowanie wpisywania danych. Jeśli użytkownik naciśnie klawisz ‘n’, komenda *Halt* zakończy działanie

programu. W innym przypadku ekran zostanie wyczyszczony i pętla się powtarza dla następnego transformatora.

Listing 4.3.4. Procedura WpiszDaneNTr

```
1   procedure WpiszDaneNTr(n: Byte);
2     var i: Byte;
3     begin
4     Window(31+(n-1)*10,5,40+(n-1)*10,10); ClrScr;
5     for i:=1 to 6 do
6       begin
7       Window(31+(n-1)*10,4+i,40+(n-1)*10,4+i);
8       {$I-}
9       repeat
10        ClrScr; Readln(dT[n,i]);
11      until (IOResult=0) AND (dT[n,i]>0);
12      {$I+}
13      ClrScr; Write(dT[n,i]:8:2);
14      dT[n,0]:=1;   {1-transf. do obliczen, 0-transf.odlaczony}
15      end;
16    end;
```

Wewnątrz procedury *WpiszDaneTr* została wywołana procedura *WpiszDaneNTr*. To ona wykonuje operację pozwalającą na wpisywanie danych w odpowiednie pola „macierzy" utworzonej przez indeksy transformatorów oraz nazwy tychże danych. W tej przestrzeni tworzy się nowe okno, którego wielkość zależy od ilości transformatorów. Wzory na te współrzędne zostały najpewniej wykonane empirycznie, poprzez sprawdzanie, dla jakich kolejnych współrzędnych dobrze jest odzwierciedlone położenie tekstu. Wewnątrz tego okna tworzą się mniejsze okna dla każdej kolejnej danej transfrormatora. Po wpisaniu wartości jest ona wpisana do faktycznej macierzy danych *dT* i przechodzi się do następnej. Tutaj też jest potrzebne wyłączenie sprawdzania wejść/wyjść z tego samego powodu, co w funkcji *UstalIloscTransf*. Na koniec wpisane wartości są wypisane i transformatorom używanym przypisuje się wartość 1 do flagi w indeksie 0. To pozwala na rozróżnienie transformatorów używanych od odłączonych.

Listing 4.3.5. Funkcja ObcSieci

```
1   function ObcSieci: Boolean;  {okreslenie obciazenia sieci}
2     var i: Byte;
3     begin
4     ObcSieci:=FALSE;
5     Window(1,24,80,25); ClrScr;
6     Smax:=0;
7     for i:=1 to iT do
8       if dT[i,0]=1 then     {dla transf. podlaczonego}
9         Smax:=Smax+dT[i,1]; {liczenie mocy max}
10    Writeln('Podaj obciazenie sieci =');
11    Write('(0-odlaczenie transformatora; max=',Smax:0:2,' kVA)');
```

```
12      Window(26,24,80,24);
13      {$I-}
14      repeat
15        ClrScr; Readln(Szad);
16        if Szad=0 then Exit;
17      until (IOResult=0) AND (Szad>0) AND NOT(Szad>Smax);
18      {$I+}
19      Window(1,24,80,25); ClrScr;
20      Window(1,12,80,12); ClrScr;
21       Write('Zadano obciazenie sieci o wartosci = ',Szad:0:2,'
    kVA');
22      ObcSieci:=TRUE;
23      end;
```

ObcSieci jest kolejną funkcją prawda/fałsz. Ona jednak nie ma na celu potwierdzanie warunków pracy. Typ Boolean jest tutaj używany ze względu na zachowanie kolejności. W części wykonawczej jest ona umieszczona jako warunek pętli *while-do*. To wymusza, aby kolejna część kodu została wykonana tylko jeśli *ObcSieci* nie napotkało na swojej drodze żadnych błędów. Początkowo jest ustawiana na fałsz, po czym tworzy się nowe okno. W tym oknie pojawi się prośba o podanie obciążenia sieci wraz z wartością maksymalną obciążenia, ale najpierw ta moc maksymalna musi być obliczona. Jest ona obliczona za pomocą prostej pętli *for*. Najpierw *Smax* ustawia się na 0, a potem w pętli dodaje się do niej kolejne moce znamionowe podłączonych transformatorów. Tutaj sprawdza się też warunek z końca procedury *WpiszDaneNTr*, czyli czy posiadają one flagę 1 na zerowym indeksie. W kolejnym oknie użytkownik wpisuje swoją moc obciążenia. Jeśli wartość podana jest nieprawidłowa (*IOResult* $\neq$ 0) albo nie jest w przedziale od 0 do *Smax* pętla się powtarza. Jeśli jest prawidłowa, okna są wyczyszczane, wyświetla się moc zadaną i *ObcSieci* ustawia się na prawdę, co umożliwia pracę kolejnej części kodu.

Listing 4.3.6. Procedura MocObc

```
1   procedure MocObc;
2     var i: Byte;
3         suma: Real;
4     begin
5     suma:=0;
6     for i:=1 to iT do
7       if dT[i,0]=1 then     {dla transf. podlaczonego}
8         suma:=suma+dT[i,1]/dT[i,4];
9     Window(1,13,80,22); ClrScr;
10    Writeln('-----------------------------------');
11    Writeln('     | moc obciazenia | znamionowa |');
12    Writeln('     |      [kVA]     |    [kVA]   |');
13    Writeln('-----------------------------------');
14    for i:=1 to iT do
15      if dT[i,0]=1 then     {dla transf. podlaczonego}
16        begin
```

```
17          So[i]:=Szad/dT[i,4]*dT[i,1]/suma; {moc obciazenia}
18          Write(' TR ',i,'  ',So[i]:13:2,dT[i,1]:13:2);
19          if dT[i,1]<So[i] then
20            Writeln('  przekroczona moc znamionowa')
21          else Writeln;
22          end;
23      Window(1,23,80,25); ClrScr;
24        Writeln('Straty mocy dla wszystkich mozliwych kombinacji
    polaczen');
25      Writeln('sa dostepne tylko na wydruku.');
26        Write('----------------------------------  Wydruk danych?
    [T/N]');
27      if UpCase(ReadKey)='T' then
28        begin           (******************* wydruk danych*)
29        Window(1,23,80,25); ClrScr;
30        {$I-}
31        Window(1,25,80,25);
32        Assign(dr,'PRN');      {przydzielenie drukarki}
33        Rewrite(dr);
34        Inc(TextAttr,Blink);
35        Write('DRUKARKA NIE JEST PODLACZONA !!!! ---- PODLACZ');
36        repeat
37          Writeln(dr);
38        until IOResult=0;
39        ClrScr;
40        Dec(TextAttr,Blink);
41        {$I+}
42        DrukDanychWe;
43        StratyMocy;  {straty mocy dowolnej liczby pracujacych TR}
44        Writeln(dr,#12);
45        Close(dr);           {odlaczenie drukarki}
46        end;
47      Window(1,23,80,25); ClrScr;
48      Window(1,1,80,25);
49      end;
```

W kodzie jest to ostatnia procedua, bo zawiera ona odniesienia do wszystkich nieomówionych jeszcze procedur, ale w części wykonującej następuje ona tuż po spełnieniu warunku *ObcSieci*. Na początku wykonana zostaje pętla *for*, w której dla transformatorów podłączonych (flaga ustawiona w procedurze *WpiszDaneNTr*) oblicza się sumę stosunku mocy znamionowej do napięcia zwarcia. Jest to pierwszy fragment potrzebny do obliczenia mocy obciążeniowej transformatorów. Tworzy się nowe okno, w którym się wyświetla porównanie wartości mocy obciążeniowej do znamionowej. Tutaj na miejscu się oblicza moce obciążeniowe. W pętli sprawdza się, czy transformator jest używany, a potem moc obciążeniową wyznacza się jako iloraz mocy zadanej do napięcia zwarcia pomnożony przez iloraz mocy znamionowej i sumy. Jest to tak naprawdę jednoznaczne z przywołanym wcześniej wzorem, tylko przekształcone do postaci dwóch ułamków. Dla kolejnych

transformatorów wypisuje się ich moce obciążeniowe, moce znamionowe oraz, jeśli moc obciążeniowa jest wyższa od znamionowej, ostrzeżenie o przeciążeniu transformatora.

Następnie pokazuje się informacja, że straty mocy są dostępne tylko na wydruku, wraz z pytaniem o wykonanie tegoż wydruku. Jest nietypowe podejście, przez co niekoniecznie zrozumiałe. Trudno powiedzieć dlaczego dostęp do tych informacji jest tak ograniczony. Jeśli użytkownik naciśnie klawisz ‘t’, rozpocznie się obsługa drukowania. Powstają nowe okna, na wypadek wystąpienia błędu. Za pomocą komendy *Assign* przypisuje się zmiennej *dr* nazwę pliku ‘PRN’. Starsze drukarki rozpoznają tą nazwę jako plik z danymi o aktualnej sesji drukowania [6]. Dzięki temu taka drukarka wydrukuje wszystko, co zostanie wpisane do pliku *dr*. W czasach Windows XP była to pożyteczna sztuczka, ale w nowszych komputerach nie jest to już obsługiwane. Pierwotne otwarcie pliku za pomocą *Rewrite* sprawi, że informacja zostanie wysłana do drukarki o wejściu w stan uwagi. Oryginalnie zostało tutaj przygotowane, że zanim drukarka odpowie zostanie wyświetlona informacja o braku drukarki, co się będzie powtarzać aż drukarka zareaguje na komendę *Writeln*. Kiedy tak się stanie, wiadome jest, że drukarka działa prawidłowo i można przejść do drukowania wyników. Wywołane zostają prodecury *WydrukDanychWe* i *StratyMocy* (opisane poniżej), po czym zamyka się plik dr, co odłącza drukarkę, czyści okna i kończy procedurę.

Listing 4.3.7. Procedura WydrukDanychWe

```
1   procedure DrukDanychWe;
2     var i: Byte;
3     begin
4      Writeln(dr,'================================================
    ===============');
5           Writeln(dr,'           PLANOWANIE    PRACY    ROWNOLEGLEJ
    TRANSFORMATOROW');
6      Writeln(dr,'================================================
    ===============');
7     Writeln(dr);
8     Write(dr,'Dane transformator$w            ');
9     for i:=1 to iT do Write(dr,'    TR ',i,'  ');
10    Writeln(dr);
11    Writeln(dr,'---------------------------------------',
12           '---------------------------------------');
13    Write(dr,'1) Moc znamionowa  Szn [kVA]: ');  {dT[n,1]}
14    for i:=1 to iT do
15      Write(dr,dT[i,1]:10:2);
16    Writeln(dr);
17    Write(dr,'2) Straty jalowe   dPj  [kW]: ');  {dT[n,2]}
18    for i:=1 to iT do
19      Write(dr,dT[i,2]:10:2);
20    Writeln(dr);
21    Write(dr,'3) Straty obciaz.  dPo  [kW]: ');  {dT[n,3]}
22    for i:=1 to iT do
23      Write(dr,dT[i,3]:10:2);
```

```
24      Writeln(dr);
25      Write(dr,'4) Nap. zwarcia    Uzw   [%]: ');  {dT[n,4]}
26      for i:=1 to iT do
27        Write(dr,dT[i,4]:10:2);
28      Writeln(dr);
29      Write(dr,'5) Nap.str.pierw.  U1   [kV]: ');  {dT[n,5]}
30      for i:=1 to iT do
31        Write(dr,dT[i,5]:10:2);
32      Writeln(dr);
33      Write(dr,'6) Nap.str.wt¢rnej U2   [kV]: ');  {dT[n,6]}
34      for i:=1 to iT do
35        Write(dr,dT[i,6]:10:2);
36      Writeln(dr);
37      Writeln(dr);
38      Writeln(dr,'Zadano obciazenie sieci o wartosci = ',Szad:0:2,'
    kVA');
39      Writeln(dr,'-------------------------------------');
40      Writeln(dr,'     | moc obciazenia | znamionowa |');
41      Writeln(dr,'     |      [kVA]     |    [kVA]   |');
42      Writeln(dr,'-------------------------------------');
43      for i:=1 to iT do
44        if dT[i,0]=1 then     {dla transf. podlaczonego}
45          begin
46          Write(dr,' TR ',i,'  ',So[i]:13:2,dT[i,1]:13:2);
47          if dT[i,1]<So[i] then
48            Writeln(dr,'  przekroczona moc znamionowa')
49          else Writeln(dr);
50          end;
51      Writeln(dr);
52      end;
```

WydrukDanychWe jest procedurą długą, ale nie jest ona skomplikowana. Służy ona do wpisania wszystkich obliczonych danych do pliku *dr* (co zostanie automatycznie wydrukowane jak opisano w procedurze *MocObc*). Jako nagłówek zostaje wypisany temat pracy, a potem w pętli dla wszystkich transformatorów wypisuje się ich dane. Na koniec wpisuje się moce obciążeniowe i odpowiadające im moce znamionowe. Powtórzony jest warunek ostrzeżenia o przeciążeniu transformatora, po czym procedura się kończy i program przechodzi do *StratyMocy*.

Listing 4.3.8. Procedua StratyMocy

```
1     procedure StratyMocy;
2       var m,j: Byte;
3       begin
4       m:=Round(Exp(iT*Ln(2)));  {ilosc kombinacji  2^iT}
5       Writeln(dr,'MOZLIWE WARIANTY PRACY ');
6       Writeln(dr,'---------------------------------------');
7       Writeln(dr,'Kombinacje  Straty mocy  Moc znamionowa');
8       Writeln(dr,' polaczen       [kW]          [kVA]    ');
9       Writeln(dr,'---------------------------------------');
```

```
10      for j:=1 to (m-1) do
11           ObliczStratyMocy(j);     {liczenie i wydruk dla j-tej
    kombinacji}
12      end;
```

Jest to również krótka procedura służąca do wypisania wszystkich możliwych kombinacji połączeń, ale ona zawiera również ciekawe zagadnienie matematyczne. Ilość możliwych kombinacji transformatorów rzeczywiście wynosi 2^*iT*, ale wzór użyty do obliczenia tej wartości pokazuje podejście starszych programów do matematyki. W przeszłości nie warto było tworzyć osobnej operacji potęgowania i logarytmu (do dzisiaj niektóre języki ich nie posiadają). Zamiast tego tworzono tylko komendy *exp* i *ln*, odpowiadające funkcji ekspotencjalnej i logarytmicznej o podstawie liczby Eulera. Potęgowanie i logarytmowanie w tej bazie jest bardzo szybkie, a przekształcenia matematyczne pozwalają zamienić dowolną bazę potęgi i logarytmu na kombinację tych komend. W tym przypadku 2^iT jest równe e^(ln(2^iT)), co z kolej wynosi e^(iT*ln(2)), czyli exp(iT*ln(2)). Oczywiście, wynik takiego przekształcania nie jest liczbą całkowitą, więc trzeba dokonać zaokrąglenia. Poza tym pętla wypisuje straty mocy dla kolejnych mocy znamionowych kombinacji transformatorów, co zostało wykonane w procedurze ObliczStratyMocy.

Listing 4.3.9. Procedua ObliczStratyMocy

```
 1    procedure ObliczStratyMocy(j: Byte);
 2      type tab = array [1..5] of Boolean;
 3      var aT: tab;    {tablica TR do liczenia}
 4           i,k: Byte;
 5           sumaSzn, sumaPj, sumaPo: Real;
 6           s: String;
 7
 8             function Powtorka(TT: tab): Boolean;  {czy powtorzyla
    sie kombin.}
 9             var i,n,k,m: Byte;
10                 AA: tab;
11             begin
12             Powtorka:=TRUE;
13             for i:=1 to j-1 do
14               begin
15               for n:=1 to iT do
16                 begin
17                                        k:=Round(Exp((n-1)*Ln(2)));
    {ustalenie ktory liczyc}
18                  AA[n]:= ((i AND k)=k) AND (dT[n,0]=1);   {wybrany
    i nie odlaczony}
19                 end;
20               m:=0;
21               for n:=1 to iT do
22                 if AA[n]=TT[n] then Inc(m);  {jest powtorka}
23               if m=iT then Exit;
24               end;
25             Powtorka:= FALSE;
26             end;
27
```

```
28        begin
29        s:='';
30        sumaSzn:=0;
31        sumaPj:=0;
32        sumaPo:=0;
33        for i:=1 to iT do
34          begin
35           k:=Round(Exp((i-1)*Ln(2)));                {ustalenie ktory
      liczyc}
36            aT[i]:= ((j AND k)=k) AND (dT[i,0]=1);    {wybrany i nie
      odlaczony}
37          end;
38        s:='TR ';
39        for i:=1 to iT do
40          if aT[i] then
41            begin
42             s:=s+Char(48+i)+' ';                {wypisanie kombinacji
      TR}
43            sumaSzn:=sumaSzn+dT[i,1];
44            sumaPj:=sumaPj+dT[i,2];
45            sumaPo:=sumaPo+dT[i,3];
46            end
47          else s:=s+'  ';
48        if Szad>sumaSzn then Exit;
49        if (sumaSzn>0) AND (NOT Powtorka(aT)) then
50          Writeln(dr,s,'  ',
51                  (sumaPj+sumaPo*Szad/sumaSzn*Szad/sumaSzn):8:2,sumaSz
      n:15:2);
52        end;
```

W tej procedurze zawarto główną część obliczeniową programu, co wzmaga pytanie o jej obecność tylko w trakcie drukowania. W obliczeniach będą potrzebne zmienne pomocnicze. Pierwszą z tych „zmiennych" jest nowy typ danych *tab*, określony jako tablica pięciu wartości prawda/fałsz. Następnie powstaje zmienna *aT* typu *tab*, dwie zmienne pętlowe *i, k*, trzy liczby rzeczywiste *sumaSzn*, *sumaPj*, *sumaPo* oraz zmienna tekstowa *s*. Potem zdefiniowana jest funkcja wewnętrzna *Powtorka*. Wiadome jest, że jeśli dana kombinacja się powtórzy to wyniki będą identyczne, dlatego by nie marnować czasu została stworzona funkcja wykrywająca powtórzenia. Ona działa na zasadzie przeciwnej do *DobreDane* - tutaj najpierw się zakłada, że kombinacja się powtórzyła i należy udowodnić, że tak nie jest. Sprawdzenie polega na wykorzystaniu trzech pętli, z czego druga i trzecia jest zawarta w pierwszej. W drugiej pętli przypisuje się zmiennej *k* wartość odpowiadająca 2^(n-1) (na tej samej zasadzie co było przedstawione w *StratyMocy*) oraz na podstawie nietypowo zapisanego warunku *AA* otrzymuje wartość która sprawdza, czy dany transformator jest jednocześnie wybrany i używany.

To czy jest używany wynika z wcześniej ustawionej flagi, ale wybranie transformatora jest określane inaczej. W pętli trzeciej tablica *AA* jest porównywana s tablicą *TT*. Jeśli są one sobie równe w jakimkolwiek punkcie, to oznacza, że kombinacja się powtórzyła i należy ją ominąć. Wewnątrz samego ObliczStratyMocy dokonuje się najpierw inicjalizacji zmiennych

i podobnie jak w powtórce wyznacza się 2^(i-1) oraz tablicę *aT*. Kolejna pętla wypisuje kolejne kombinacje transformatorów. Jeśli kombinacja transformatorów jest wybrana i używana, jej indeksy są wypisywane (*Char(48)* oznacza ’0’, dlatego *Char(48+i)* daje wartość *i* jako znak), a ich moce znamionowe, straty jałowe i straty obciążeniowe są wpisywane odpowiednio do *sumaSzn*, *sumaPj* oraz *sumaPo*. Potem sprawdza się, czy moc zadana jest większa od sumy mocy znamionowych. Na koniec, po warunkiem, że suma mocy znamionowych jest dodatnia oraz kombinacja nie jest powtórką, oblicza się straty mocy wzorem równoznacznym do wcześniej wyprowadzonego, wypisując je wraz z mocami znamionowymi.

Listing 4.3.10. Funkcja OdlaczTr

```
1    function OdlaczTr: Boolean;   {odlaczanie ktoregos transform.}
2      var i: Byte;
3          s: String;
4      begin
5      OdlaczTr:=FALSE;
6      s:='';
7      for i:=1 to iT do
8        dT[i,0]:=1;   {na poczatku wszystkie sa podlaczone}
9      Window(1,24,80,25); ClrScr;
10     Write('Podaj transformatory, kt¢re chcesz odlaczyc [ ');
11     for i:=0 to iT do
12       Write(Char(48+i),' ');
13     Writeln(']: ');
14     Write('  (ENTER- koniec pracy)');
15     Window(65,24,80,24);
16     {$I-}
17     repeat
18       ClrScr; Readln(s);
19     until IOResult=0;
20     {$I+}
21     Window(1,24,80,25); ClrScr;
22     if s='' then Exit;   {koniec pracy}
23     for i:=1 to iT do
24       if Pos(Char(48+i),s)<>0 then
25         dT[i,0]:=0;   {odlaczony}
26     Window(1,1,80,25);
27     OdlaczTr:=TRUE;
28     end;
```

Ostatnią funkcją jest funkcja *OdlaczTr*. Ona pozwala na ponowne wykonanie obliczeń przy innych załączonych transformatorach. Najpierw zakłada się, że odłączenie nie nastąpi, po czym załącza się wszystkie transformatory i prosi użytkownika o podanie, które transformatory mają być odłączone (wraz z listą indeksów transformatorów). Jest też opcja wcisnąć „Enter”, co wyjdzie z funkcji i program się zakończy. Jest to sprawdzane zmienną tekstową *s*. Jeśli użytkownik naciśnie „Enter”, so *s*=’’, co jest dalej sprawdzane. Jeśli nie, użytkownik wpisuje numery transformatorów, które mają być odłączone. W ostatniej pętli *for* jest sprawdzane, które transformatory zostały wybrane. Komenda *Pos* sprawdza pozycję w której znajduje się *Char(48+i)*, czyli indeks kolejnych transformatorów, wewnątrz

zmiennej *s*. Jeśli nie zostanie znaleziony, to wynikiem jest 0 i pętla przechodzi do następnej iteracji. Jeśli zostanie znaleziony, to transformatorowi odpowiadającemu temu indeksowi ustawia się flagę 0 na indeksie zerowym, przez co kolejne obliczenia uznają ten transformator za odłączony. Potem OdlaczTr ustawia się na prawdę i funkcja się kończy.

Listing 4.3.11. Część wykonująca program

```
1    begin
2    iT:=UstalIloscTransf;
3    WpiszDaneTr;
4    repeat
5        while ObcSieci do     {obliczenie dla podanej wartosci
     obciazenia sieci}
6        MocObc;    {obliczanie mocy obciazenia i-tego TR}
7    until NOT OdlaczTr;    {powtorne obliczenia dla odlaczonych
     transf.}
8    Window(1,1,80,25); ClrScr;
9    end.
```

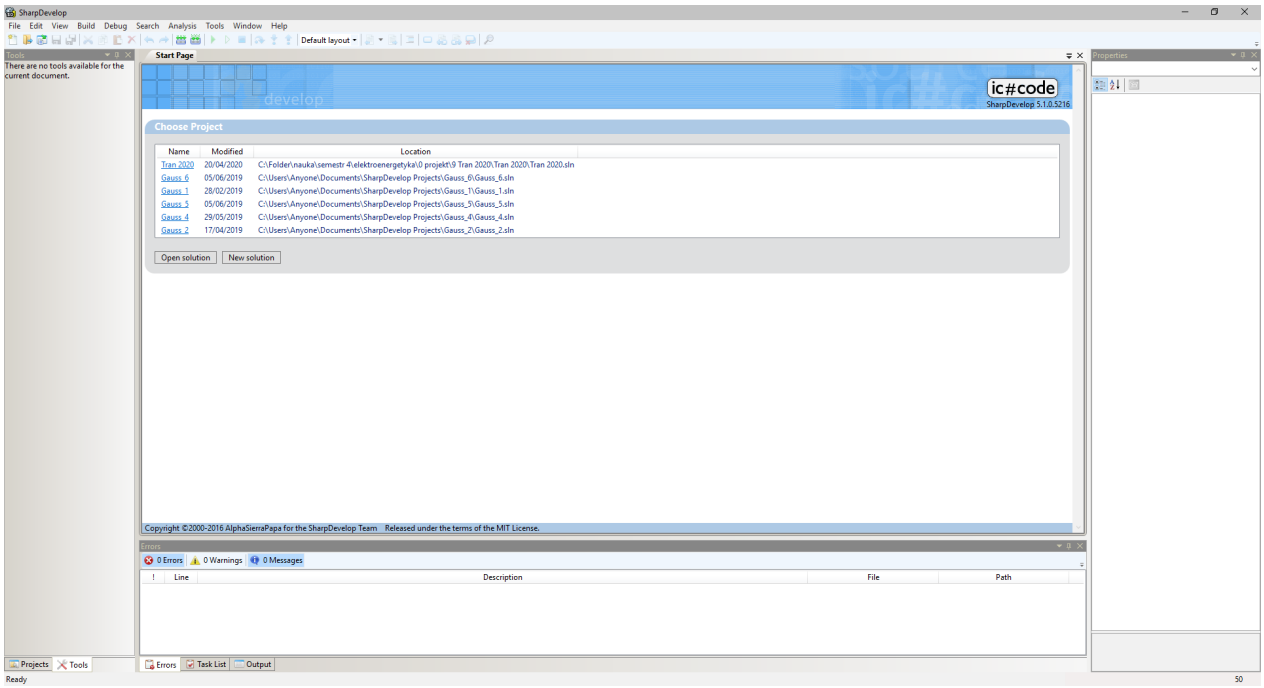
Na sam koniec pozostała część wykonująca program, czyli to co się faktycznie dzieje po uruchomieniu programu. Najpierw zostaje ustawiona ilość transformatorów *iT* za pomocą funcji *UstalIloscTransf*. Potem wykonuje się procedura *WpiszDaneTr* i użytkownik wpisuje dane transformatorów. Następnie w pętli wykonują się wszystkie obliczenia. Jeśli prawidłowo się ustali obciążenie sieci poprzez *ObcSieci*, następują obliczenia mocy obliczeniowej w *MocObc*. Pętla się powtórzy za każdym razem, gdy użytkownik poda nowe transformatory do odłączenia w funkcji OdlaczTr. Gdy przestanie, okno zostaje wyczyszczone i program się kończy.

5 Wybór i opis środowiska programistycznego

Jak wcześniej wspomniano, język Pascal jest ograniczony. Stosuje się go do wykonywania programów konsolowych bazujących na oprogramowaniu DOS. Z powodu unowocześnienia systemu Windows DOS przestał być wspierany, a Pascal stał się językiem mało przydatnym do programowania. Przez to nowy program powinien zostać napisany od zera w nowym języku programowania. Dobrym wyborem jest język C#. Jest to język bazowany ja C, ale znacznie zmodernizowany dla łatwości użytkowania [8].

Jest on oficjalnie wspierany przez nowe systemy Windows. Nawet programy dla najnowszych wersji Windows 10 są pisane w C#. Cieszy się on dużym wsparciem w zintegrowanych środowiskach programistycznych. Jego struktura pozwala na zautomatyzowanie i ukrycie wielu mniej istotnych procesów aplikacji, takich jak definiowanie komponentów graficznych. Zamiast tego środowiska programistyczne dają możliwość modelowania programu za pomocą elementów, które są już widoczne na ekranie użytkownika przed kompilacją programu. Poza tym C# jest językiem łatwiejszym w zrozumieniu. Jako język obiektowy możliwe jest odnoszenie się do istniejących komponentów, co niweluje potrzebę odnoszenia się do ezoterycznych komend.

Jako zintegrowane środowisko programistyczne zastosowano SharpDevelop. Jest to program Open Source, co oznacza, że użytkownicy mają dostęp do jego kodu źródłowego. Jest on darmową alternatywą do bardziej popularnych programów, jak Windows Visual Studio. Ma on te same możliwości projektowania co WVS, a potencjalne problemy programistyczne są kwestią poszukiwań rozwiązań w ramach języka C# [9].



Rys. 5.1. Ekran główny programu SharpDevelop [opracowanie własne]

Przeniesienie funkcjonalności programu tran.exe z Pascal na C# polega na dokładnym zrozumieniu działania kodu źródłowego, a następnie odtworzeniu go w wybranym języku. Dlatego analiza kodu wykonana w rozdziale 4.3 była ważną częścią procesu aktualizacji. Przykład modernizacji kodu można zaprezentować na procedurze *WpiszDaneTr*. Dużej części tej procedury nie trzeba wcale przenosić do C#, ponieważ wynikają z ograniczeń konsolowych programu bazowego. W nowym programie wpisywanie danych zostanie wykonane przez użytkownika w pola macierzy danych w oknie programu. Jednak podfunkcja *DobreDane* jest procedurą warunkową sprawdzającą, czy spełnione zostały warunki pracy równoległej. Jej kod może zostać przetłumaczony na funkcję w języku C#.

Listing 5.1. Podfunkcja *DobreDane*

```
1   function DobreDane(n: Byte): Boolean;
2           var i: Byte;
3           begin
4           DobreDane:=FALSE;
5           if n>1 then
6             begin
7                                 if   dT[n-1,5]<>dT[n,5]   then   Exit;
    {spr.nap.str.pierwotnej}
8                                 if   dT[n-1,6]<>dT[n,6]   then   Exit;
    {spr.nap.str.wtornej}
9             for i:=1 to n-1 do
10              begin
11               if (dT[i,1]/dT[n,1])>3 then Exit;         {stos.mocy
    znamion.}
12               if (dT[i,1]/dT[n,1])<(1/3) then Exit;
13               end;
14            end;
15          DobreDane:=TRUE;
16          end;
```

W języku C# funkcja posiada dwie części porównywalne z Pascalem – nagłówek i ciało. Nagłówek funkcji jest fragmentem tekstu, który definiuje rodzaj stworzonej funkcji. Podobnie jak wyżej trzeba zdefiniować, że funkcja jest typu Boolean, czyli zwraca jako swoją wartość prawdę/fałsz. Ciałem są nawiasy określające zasięg danej funkcji.

Listing 5.2. Początek nowej funkcji *checkData*

```
1   bool checkData()
2   {
3
4   }
```

Zadaniem nowej funkcji *checkData* (konwencją programowania jest utrzymywanie nazw funkcji w języku angielskim) jest sprawdzanie, czy zostały spełnione warunki pracy równoległej. Pierwszy warunkiem, aby w ogóle obliczenia były możliwe, jest sprawdzenie

mocy obciążenia sieci. Jeśli jest ujemna lub większa od maksymalnej, to obliczenia się nie powiodą. Jest to prosty warunek do sprawdzenia

Listing 5.3. Kolejna część funkcji *checkData*

```
1   bool checkData()
2   {
3         if(Szad<=0)
4         {
5               dataError="Obciazenie ujemne/zerowe!";
6               return false;
7         }
8         if(Szad>Smax)
9         {
10              dataError="Obciazenie wieksze od maksymalnego!";
11              return false;
12        }
13  }
```

Podobnie jak w kodzie źródłowym niespełnienie warunków zwraca fałsz. Dodatkowo jednak przypisuje się zmiennej *dataError* tekst odpowiadający błędowi który wystąpił. Ten błąd zostanie wyświetlony, gdy program wyjdzie z funkcji *checkData*. Kolejną częścią Programu jest sprawdzenie warunków pracy na poszczególnych transformatorach. Tutaj zostało to wykonane przy pomocy podwójnej pętli *for*. W nich dopiero definiuje się zmienne pętlowe. Tutaj następuje również sprawdzenie, czy transformator jest przyłączony. Jeśli flaga przyłączenia wynosi 1, to sprawdza się kolejne warunki pracy równoległej.

Listing 5.4. Kolejna część funkcji *checkData*

```
1   bool checkData()
2   {
3         if(Szad<=0)
4         {
5               dataError="Obciazenie ujemne/zerowe!";
6               return false;
7         }
8         if(Szad>Smax)
9         {
10              dataError="Obciazenie wieksze od maksymalnego!";
11              return false;
12        }
13        for(int i=0;i<iT;i++)
14        {
15              for(int j=i+1;j<iT;j++)
16              {
17                    if(dT[i,7]==1)
18                    {
19                    if(Math.Abs(dT[i,4]-dT[j,4])/dT[i,4]>0.005)
20                          {
```

```
21                              dataError="Odchylka            napiec
    pierwotnych wieksza niz 0.5%!";
22                              return false;
23                          }
24                      }
25                  }
26          }
27          return true;
28      }
```

Pętla podwójna wygląda skomplikowanie, ale jest łatwa do zrozumienia. Pierwsza pętla określa, że chcemy wykonać operacje na transformatorach od 0 do *iT*, a indeksy tych transformatorów to *i*. W drugiej pętli stwierdzamy, że chcemy również sprawdzić wszystkie transformatory o indeksach większych niż *i*, którym przypisujemy indeksy *j*. Sprawdzana jest flaga przyłączenia, a wewnątrz dopiero sprawdzamy warunek równości napięć pierwotnych. Tutaj istnieje większa wolność w wyborze napięć, ponieważ różnica mniejsza niż 0.5% nie jest groźna. Powtórzenie sprawdzenia dla kolejnych warunków działa tak samo jak przy napięciach pierwotnych, przez co nie ma potrzeby ich ponownego tłumaczenia.

Listing 5.5. Gotowa funkcja *checkData*

```
1   bool checkData()
2   {
3         if(Szad<=0)
4         {
5               dataError="Obciazenie ujemne/zerowe!";
6               return false;
7         }
8         if(Szad>Smax)
9         {
10              dataError="Obciazenie wieksze od maksymalnego!";
11              return false;
12        }
13        for(int i=0;i<iT;i++)
14        {
15              for(int j=i+1;j<iT;j++)
16              {
17                    if(dT[i,7]==1)
18                    {
19                    if(Math.Abs(dT[i,4]-dT[j,4])/dT[i,4]>0.005)
20                          {
21                                dataError="Odchylka            napiec
    pierwotnych wieksza niz 0.5%!";
22                                return false;
23                          }
24                    if(Math.Abs(dT[i,5]-dT[j,5])/dT[i,5]>0.005)
25                          {
26                                dataError="Odchylka  napiec  wtornych
    wieksza niz 0.5%!";
```

```
27                              return false;
28                          }
29                          if(Math.Abs(dT[i,3]-dT[j,3])/dT[i,3]>0.1)
30                          {
31                              dataError=”Odchylka  napiec  zwarcia
    wieksza niz 10%!";
32                              return true;
33                          }
34                          if(dT[i,0]/dT[j,0]>3)
35                          {
36                              dataError="Stosunek              mocy
    znamionowych wiekszy niz 3!";
37                              return false;
38                          }
39                          if(dT[j,0]/dT[i,0]>3)
40                          {
41                              dataError="Stosunek              mocy
    znamionowych mniejszy niz 1/3!";
42                              return false;
43                          }
44                          if(dT[i,6]!=dT[j,6])
45                          {
46                              dataError="Grupy             polaczen
    niezgodne!";
47                              return false;
48                          }
49                      }
50                  }
51              }
52              return true;
53          }
```

6 Implementacja programistyczna tworzonej aplikacji

Program Tran 2020, podobnie jak oryginalny program, został podzielony na funkcje, aby jego kod był łatwiejszy do zrozumienia. Poszczególne fragmenty zostaną dokładnie opisane i skomentowane w razie potrzeby.

Listing 6.1. Definicja zmiennych oraz konstruktor *MainForm*

```
1     int iT, iTmax, idSmin, idSmax, currentColumn;
2     double[,] dT;
3     double[] So, dS;
4     double dSmin, dSmax, Smax, Szad, sumaSnUk, sumaSzn, sumaPj,
      sumaPo;
5     string nl, tmpPath, dataError;
6     string[] library;
7     bool[] activeOrder;
8     StreamReader reader;
9     StringBuilder resultexport;
10    public MainForm()
11    {
12          InitializeComponent();
13          init();
14    }
```

Ważniejsze zmienne zastosowane w nowym programie zostały zdefiniowane na początku kodu. Zmienne, których nazwy się pokrywają z poprzednim programem (*iT*, *dT* itp.) mają taką samą rolę, a więc nie trzeba ich ponownie opisywać. Zmienna *iTmax* określa największą możliwą ilość transformatorów, którą można zdefiniować. W tym programie nie ma ograniczenia do pięciu, ponieważ w programach oknowych można zmieścić więcej kolumn z danymi. *idSmin* oraz *idSmax* określają kolejno indeks najmniejszej i największej wartości strat mocy czynnej dla danych transformatorów. Stosowane są przy obliczeniu strat mocy czynnej w różnych kombinacjach. Zmienna *currentColumn* to indeks kolumny, która została aktualnie wybrana. Analogicznie do *idSmin* oraz *idSmax*, *dSmin* i *dSmax* to najmniejsza i największa wartość strat moczy czynnej. Zmienne tekstowe *nl*, *tmpPath* oraz *dataError* zostały zdefiniowane dla wygody programowania – będą im przypisane wartości odpowiednio znaku nowej linii, adresowi pliku tymczasowego tworzonego w celu drukowania oraz tekstu informacji błędu. Tablica *library* odpowiada kolejnym wierszom biblioteki transformatorów. Z kolej tablica *activeOrder* jest alternatywnym rozwiązaniem problemu obliczenia wszystkich możliwych kombinacji transformatorów. Jest to tablica zmiennych logicznych prawda/fałsz i to od niej zależy czy transformator zostanie wliczony do aktualnej kombinacji. Zmienna *reader* została utworzona w celu ułatwienia drukowania, a *resultExport* – ułatwienie eksportowania wyników. Obsługa drukowania w tym programie polega na zapisie tymczasowego pliku na pulpicie, a następnie wydrukowaniu jego zawartości. Dzięki wybranemu środowiskowi programowania nie trzeba się zajmować bezpośrednio kodem uruchamiającym program. Wszystko potrzebne do tego zostało zamieszczone w funkcji wewnętrznej *InitializeComponent()*. Z drugiej strony jednak chcemy,

aby na samym początku zostały zdefiniowane wstępne wartości naszych zmiennych. Z tego powodu do konstruktora klasy *MainForm* wstawiono odniesienie do funkcji *init()*.

Listing 6.2. Funkcja *init()*

```
1    void init()
2    {
3         iT=2;
4         iTmax=10;
5         dSmin=0;
6         dSmax=0;
7         idSmin=1;
8         idSmax=1;
9         Smax=0;
10        Szad=0;
11        currentColumn=0;
12        nl=System.Environment.NewLine;
13   tmpPath=Environment.GetFolderPath(Environment.SpecialFolder.Des
     ktop)+"\\tmp.txt";
14        dataError="";
15        library=File.ReadAllLines(Application.StartupPath+"\\
     transformatory.txt");
16        buttonCopy.Enabled=false;
17        buttonSave.Enabled=false;
18        buttonExport.Enabled=false;
19        buttonPrint.Enabled=false;
20        saveDialog.Filter="Plik Tekstowy |*txt";
21        saveDialog.Title="Zapisz wyniki obliczen...";
22        saveDialog.FileName="Wyniki_Tekst.txt";
23        saveDialog.OverwritePrompt=true;
24        exportDialog.Filter="Plik Tekstowy |*txt";
25        exportDialog.Title="Eksportuj wyniki obliczen...";
26        exportDialog.FileName="Wyniki_Eksport.txt";
27        exportDialog.OverwritePrompt=true;
28        resultexport=new StringBuilder();
29        numericTransformers.Value=iT;
30        numericTransformers.Maximum=iTmax;
31        dataTransformers.RowCount=7;
32        for(int i=0;i<7;i++)
33        {
34   dataTransformers.Rows[i].Height=dataTransformers.Height/7;
35        }
36        initContext();
37        updateData(iT);
38   }
```

Wewnątrz funkcji *init()* następuje inicjalizacja zmiennych zdefiniowanych wyżej. Na początku przyjmuje się, że wybrane zostały dwa transformatory a maksymalna ilość wynosi 10. Ponieważ straty mocy nie zostały jeszcze obliczone to zmienne związane z nimi ustawiono na 0 lub 1 zależnie od kontekstu. Z przypisanych wartości *nl* i *tmpPath* widać dlaczego zostały użyte – ich wartości są dosyć długie i niewygodne w wielokrotnym stosowaniu. Do zmiennej *library* następuje automatyczne odczytanie pliku zawierającego dane transformatorów. Potem następuje ustawienie przycisków i dialogu, które zostały

dodane do programu graficznie. Na początku obliczenia nie zostały jeszcze wykonane, dlatego przyciski kopiowania, zapisu i drukowania wyników obliczeń są nieaktywne. Z kolej dialogowi zapisu i eksportu przypisuje się domyślne wartości dla plików tekstowych (filtr .txt, nazwa dialogu, domyślna nazwa pliku). Ustalenie danych wstępnych określa pierwotny wygląd macierzy danych przy otwarciu programu. Obiektowi tupu *numericUpDown* (dodany graficznie) przypisano domyślną wartość ilości transformatorów oraz wymuszono by maksymalna ilość transformatorów się zgadzała z *iTmax*. Macierzom z danymi transformatorów przypisano siedem wierszy (po jednym na wymaganą wartość definicyjną transformatora) i określono, by wysokość tychże wierszy była równa jednej siódmej wysokości całej macierzy. Ponieważ przypis danych transformatorów z *library* do programu jest wygodniejsze w osobnej funkcji, następuje odniesienie do funkcji *initContext()*. Ostatni fragment definicji macierzy, określenie kolumn, został przeniesiony do osobnej funkcji *updateData()*. Zostało to zrobione, ponieważ ta funkcja jest wywoływana wielokrotnie w różnych miejscach.

Listing 6.3. Funkcja *initContext()*

```
1    void initContext()
2    {
3         for(int i=1;i<library.Length;i++)
4         {
5              if(library[i-1].Contains("Transformator"))
6              {
7                   try
8                   {
9                        var                         transformer=new
     ToolStripMenuItem(library[i]);
10                       transformer.Name=library[i];
11                       transformer.Click+=new
     EventHandler(MenuItemClick);
12                       contextMenuStrip.Items.Add(transformer);
13                  }
14                  catch
15                  {
16                       MessageBox.Show("Zly                format
     biblioteki!","Blad
     biblioteki",MessageBoxButtons.OK,MessageBoxIcon.Error);
17                  }
18             }
19        }
20   }
```

Wewnątrz *init()* zmiennej *library* przypisano jako wartość wszystkie wiersze pliku bibliotecznego programu. Jednak w samym programie dane transformatorów mają być umieszczone na pasku narzędzi *contextMenuStrip* (dodany graficznie). Dzięki takiemu rozwiązaniu dane transformatorów będą dostępne po kliknięciu pola w macierzy danych prawym przyciskiem myszki. Dla wszystkich wierszy w *library* następuje wyszukanie słowa kluczowego „Transformator”, które oznacza początek danych nowego transformatora. Zmiennej tymczasowej *transformer* przypisuje się kolejne miejsce w pasku narzędzi. Tekst

wyświetlany definiuje jako pierwszą wartość w definicji bibliotecznej. Zachowanie po kliknięciu tej pozycji w pasku narzędzi określone jest dzięki funkcji delegacyjnej *MenuItemClick*. Na koniec dodaje się w pełni *transformer,* jako nową wartość w *contextMenuStrip*. Jeśli po drodze nastąpił jakikolwiek błąd, wyświetli się komunikat błędu o formacie biblioteki.

Listing 6.4. Funkcja *MenuItemClick()*

```
1     void MenuItemClick(object sender, EventArgs e)
2     {
3           for(int i=0;i<library.Length;i++)
4           {
5                 var item=(ToolStripMenuItem)sender;
6                 if(library[i]==item.Name)
7                 {
8                       for(int j=1;j<8;j++)
9                       {
10                            dataTransformers[currentColumn,j-
      1].Value=library[i+j];
11                      }
12                }
13          }
14    }
```

Chcemy, aby po kliknięciu nazwy transformatora jego dane zostały wpisane do wybranej kolumny w macierzy danych. Łatwym sposobem na uzyskanie tego jest przeszukanie zmiennej *library* w poszukiwaniu nazwy wybranego transformatora. Jeśli ich wartości się zgadzają, to dla kolejnych wierszy w wybranej kolumnie wpisuje się wartości z kolejnych wierszy w *library*.

Listing 6.5. Funkcja *updateData()*

```
1     void updateData(int n)
2     {
3                 dataTransformers.ColumnCount=n;
4                 for(int i=0;i<iT;i++)
5                 {
6     dataTransformers.Columns[i].Width=dataTransformers.Width/n;
7                 }
8     }
```

Ta funkcja jest krótka, ale niezbędna. Jest ona stosowana w celu dynamicznego ustawiania ilości oraz szerokości kolumn. Najpierw ilość wymagana jest wpisana bezpośrednio do macierzy, a potem szerokość każdej kolumny była równa jednej n–tej szerokości całej macierzy. Zostało to użyte i przy inicjalizacji macierzy, jak i przy każdej zmianie aktualnej wartości *iT*.

Listing 6.6. Wydarzenie *ButtonCalcClick()*

```
1     void ButtonCalcClick(object sender, EventArgs e)
2     {
3           insertData();
```

```
4           if(checkData())
5           {
6                 calculate();
7                 showResultsText();
8           }
9           else
10          {
11                string dataError="";
12    if(Szad<=0|Szad>Smax) dataError=" Nieprawidlowe obciazenie!";
13    else dataError=" Zle dane!";
14    MessageBox.Show("Warunki pracy rownoległej
      niespelnione!"+dataError,"Zle
      warunki",MessageBoxButtons.OK,MessageBoxIcon.Error);
15          }
16    }
```

Największą różnicą pomiędzy Pascalem a C# jest to, że nie trzeba osobno programować wszystkich zachowań programu. Wystarczy tylko określić odpowiedź programu na pewne wydarzenia. Po wpisaniu wszystkich danych transformatorów oraz określeniu obciążenia użytkownik naciska przycisk Oblicz. Kiedy to się stanie, program wywołuje wydarzenie *ButtonCalcClick()*. To jest odpowiednik części wykonującej z poprzedniego programu. Gdy naciśnięty zostanie ów przycisk, wywołana jest funkcja wpisująca dane transformatorów do odpowiednich zmiennych. Następnie sprawdzane jest, czy wpisane dane są prawidłowe. Jeśli tak, to program wykonuje obliczenia i wyświetla je w oknie tekstowym *textResults*. Jeśli nie, to oznacza, że nastąpił błąd. W tym punkcie kodu są możliwe tylko dwa błędy – złe obciążenie lub złe dane transformatorów. Krótki warunek sprawdza, czy moc zadana jest mniejsza lub równa 0 albo większa od sumy mocy transformatorów. Jeśli tak, to do tekstu błędu dodaje się informację o złym obciążeniu. W innym przypadku błąd jest w danych transformatora. Na koniec stosuje się wewnętrzną metodę do wywołania okna informacyjnego.

Listing 6.7. Wydarzenie *NumericTransfromersValueChanged()*

```
1     void NumericTransformersValueChanged(object sender, EventArgs
      e)
2     {
3           iT=(int)numericTransformers.Value;
4           labelState.Text="Wpisz dane transformatorow...";
5           updateData(iT);
6     }
```

Najlepiej jest to pokazane w wydarzeniu odpowiadającym za zmianę ilości transformatorów. Za każdym razem jak użytkownik wybierze nową ilość, wartość jest przypisana do *iT*, tekst stanu ustawia się na „Wpisz dane transformatorów...” oraz wywołana jest funkcja *updateData()*.

Listing 6.8. Funkcja *insertData()*

```
1    void insertData()
2    {
3          dT=new double[iTmax,7];
4          So=new double[iT];
5          activeOrder=new bool[iT];
6          for(int i=0;i<iT;i++)
7          {
8                for(int j=0;j<6;j++)
9                {
10                     dT[i,j]=0;
11               }
12               So[i]=0;
13         }
14         try
15         {
16               Szad=double.Parse(textBoxLoad.Text);
17         }
18         catch
19         {
20   MessageBox.Show("Podaj prawidłowe obciazenie sieci","Zle
     obciazenie",MessageBoxButtons.OK,MessageBoxIcon.Error);
21         }
22         for(int i=0;i<iT;i++)
23         {
24               if(!tryInsertData(i)) break;
25         }
26         Smax=dT[0,0];
27         for(int i=1;i<iT;i++)
28         {
29               Smax+=dT[i,0];
30         }
31   }
```

Wpisywanie danych transformatorów wykonano w funkcji *insertData()*. Na początku definiuje się wstępne wartości dla *dT*, *So* i *activeOrder* (nie dało się tego wykonać wcześniej ze względu na możliwość zmiany ilości transformatorów). Program próbuje zapisać wartość wpisaną do *textBoxLoad* jako moc zadaną na transformatory. Jeśli to się nie uda, to znaczy, że wpisana wartość jest nieprawidłowa i użytkownik zostanie o tym poinformowany przez okno informacyjne. Następnie program próbuje podobnie wpisać wszystkie dane transformatorów. Tutaj zastosowano pewną sztuczkę. Gdyby wykonano zwykłą pętlę podwójną i użyto jej do spisu danych, a w pewnym momencie pojawiłby się błąd, to nagle informacja o błędzie pojawiłaby się tyle razy ile pozostało iteracji w pętli (komenda *break* wychodzi tylko z jednego poziomu pętli). Dlatego wpis danych został podzielony na jedną pętlę, która obsługuje funkcję *tryInsertData()*. Jeśli ta funkcja napotka błąd, to zwróci wartość fałsz i program może wyjść jednocześnie z obu pętli, pokazując informację o błędzie tylko raz. Na koniec oblicza się sumę mocy znamionowych wszystkich transformatorów za pomocą zwykłej pętli *for*.

Listing 6.9. Funkcja *tryInsertData()*

```
1     bool tryInsertData(int i)
2     {
3           for(int j=0;j<6;j++)
4           {
5                 try
6                 {
7     dT[i,j]=double.Parse(dataTransformers[i,j].Value.ToString());
8     switch(dataTransformers[i,6].Value.ToString().Substring(0,2))
9                       {
10                            case "Yy":
11                                  dT[i,6]=100;
12                                  break;
13                            case "Yd":
14                                  dT[i,6]=200;
15                                  break;
16                            case "Dy":
17                                  dT[i,6]=300;
18                                  break;
19                            case "Dd":
20                                  dT[i,6]=400;
21                                  break;
22                            case "Yz":
23                                  dT[i,6]=500;
24                                  break;
25                            case "Dz":
26                                  dT[i,6]=600;
27                                  break;
28                      }
29    dT[i,6]+=int.Parse(dataTransformers[i,6].Value.ToString().Subst
      ring(2));
30                      dT[i,7]=1;
31                }
32                catch
33                {
34                      MessageBox.Show("Wypelnij     wszystkie     pola
      prawidlowymi                                  wartosciami!","Zle
      dane",MessageBoxButtons.OK,MessageBoxIcon.Error);
35                      return false;
36                }
37          }
38          return true;
39    }
```

Próba wpisu danych polega na tym, że dla każdego pola w *dataTransformers* program próbuje wpisać dane z niego do odpowiedniego pola w *dT*. Zwłaszcza jest to utrudnione przez grupę transformatora, ponieważ nie ma łatwego sposobu wpisać wartości tekstowej do macierzy typu *double*. Z tego powodu trzeba wpisać do wiersza grupy wartości odpowiadające – jeśli grupa to Yy, wpisuje się wartość 100, jeśli Yd, to 200 itd. Są to wielokrotności 100, bo na koniec do tej wartości charakterystycznej dodaje się faktyczną wartość przesunięcia godzinowego. Jeśli to się uda, to dodatkowo przypisuje się flagę 1 do ostatniego wiersza danej kolumny. To odpowiada fladze indeksu 0 z oryginalnego programu. Jeśli udało się wpisać dane transformatora, to można uznać go za podłączonego, funkcja

zwraca prawdę i program przechodzi do kolejnej kolumny. Jeśli jednak nastąpił błąd, zostaje wyświetlona informacja dla użytkownika, funkcja zwraca fałsz i program wychodzi z pętli.

Listing 6.10. Funkcja *checkData()*

```
1     bool checkData()
2     {
3           if(Szad<=0)
4           {
5                 dataError="Obciazenie ujemne/zerowe!";
6                 return false;
7           }
8           if(Szad>Smax)
9           {
10                dataError="Obciazenie wieksze od maksymalnego!";
11                return false;
12          }
13          for(int i=0;i<iT;i++)
14          {
15                for(int j=i+1;j<iT;j++)
16                {
17                      if(dT[i,7]==1)
18                      {
19                      if(Math.Abs(dT[i,4]-dT[j,4])/dT[i,4]>0.005)
20                            {
21                                  dataError="Odchylka           napiec
      pierwotnych wieksza niz 0.5%!";
22                                  return false;
23                            }
24                      if(Math.Abs(dT[i,5]-dT[j,5])/dT[i,5]>0.005)
25                            {
26                                  dataError="Odchylka  napiec  wtornych
      wieksza niz 0.5%!";
27                                  return false;
28                            }
29                            if(Math.Abs(dT[i,3]-dT[j,3])/dT[i,3]>0.1)
30                            {
31                                  MessageBox.Show("Warunki       pracy
      rownoległej niespelnione! Odchylka napiec zwarcia wieksza niz
      10%!","Zle
      warunki",MessageBoxButtons.OK,MessageBoxIcon.Warning);
32                                  return true;
33                            }
34                            if(dT[i,0]/dT[j,0]>3)
35                            {
36                                  dataError="Stosunek             mocy
      znamionowych wiekszy niz 3!";
37                                  return false;
38                            }
39                            if(dT[j,0]/dT[i,0]>3)
40                            {
41                                  dataError="Stosunek             mocy
      znamionowych mniejszy niz 1/3!";
42                                  return false;
43                            }
44                            if(dT[i,6]!=dT[j,6])
45                            {
```

```
46                                  dataError="Grupy              polaczen
    niezgodne!";
47                                  return false;
48                              }
49                          }
50                    }
51              }
52              return true;
53        }
```

Funkcja *checkData()* służy do sprawdzenia, czy dane wpisane są zgodne z warunkami pracy równoległej. Działa to prawie identycznie do funkcji *DobreDane* z oryginalnego programu, tylko jest ona ubogacona o określenie komunikatów błędów. Jeśli niespełnione są warunki pracy równoległej transformatorów (moc zadana zbyt niska/wysoka, nierówność napięć strony pierwotnej/wtórnej itp.) sprawdzane dla kolejnych par podłączonych transformatorów, to program zwraca fałsz i zostaje podana informacja o nieprawidłowych danych. Dopiero, gdy zostaną spełnione te warunki można przejść do obliczeń.

Listing 6.11. Funkcja *calculate()*

```
 1    void calculate()
 2    {
 3          labelState.Text="Obliczanie strat...";
 4          sumaSnUk=0;
 5          for(int i=0;i<iT;i++)
 6          {
 7                if(dT[i,6]==1)
 8                {
 9                      sumaSnUk+=dT[i,0]/dT[i,3];
10                }
11          }
12          for(int i=0;i<iT;i++)
13          {
14                if(dT[i,6]==1)
15                {
16                      So[i]=Szad*dT[i,0]/(dT[i,3]*sumaSnUk);
17                }
18          }
19          dS=new double[(int)Math.Pow(2,iT)];
20          for(int i=0;i<Math.Pow(2,iT);i++)
21          {
22                dS[i]=0;
23          }
24          for(int i=1;i<=Math.Pow(2,iT)-1;i++)
25          {
26                sumaSzn=0;
27                sumaPj=0;
28                sumaPo=0;
29                activeOrder=getOrder(i);
30                for(int j=0;j<iT;j++)
31                {
32                      if(activeOrder[j])
33                      {
34                            sumaSzn+=dT[j,0];
```

```
35                          sumaPj+=dT[j,1];
36                          sumaPo+=dT[j,2];
37                    }
38              }
39              dS[i]=sumaPj+sumaPo*Math.Pow(Szad/sumaSzn,2);
40              if(i==1)
41              {
42                    dSmin=dS[i];
43                    dSmax=dS[i];
44              }
45              if(dS[i]<dSmin)
46              {
47                    dSmin=dS[i];
48                    idSmin=i;
49              }
50              if(dS[i]>dSmax)
51              {
52                    dSmax=dS[i];
53                    idSmax=i;
54              }
55        }
56    }
```

Wszystkie obliczenia związane z pracą równoległą podłączonych transformatorów są wykonywane w funkcji *calculate()*. Na początku tej funkcji jednak wykonano pewien zabieg bezpieczeństwa – tekst stanu określony, jako *labelState* ustawiono na „Obliczanie strat…”. Mimo że w normalnej pracy ten tekst jest niewidoczny, to jednak zostało to wykonane ze względu na niezłapane błędy w obliczeniach. Jeśli nastąpi jakiś problem niezwiązany z programem i obliczenia się zawieszą, to przynajmniej użytkownik będzie wiedział, w którym punkcie nastąpił problem. Aby obliczyć moc obciążenia transformatorów, najpierw wyznacza się sumę ilorazów mocy znamionowych do napięcia zwarcia. Następnie, w osobnej pętli można wyznaczyć moce obciążeniowe na podstawie wcześniej określonego wzoru. Tak jak w poprzednim programie, ilość kombinacji transformatorów jest równa 2^*iT*, jednak tym razem nie ma potrzeby wykonywania operacji matematycznych – C# jest na językiem na tyle nowoczesnym, że posiada operację potęgowania, jako funkcja *Math.Pow()*. Na początku wpisuje się domyślną wartość 0 dla każdej kombinacji, a następnie przechodzi się do faktycznych obliczeń. W kolejnej pętli oblicza się potrzebne sumy dla aktualnej kombinacji. Ta kombinacja jest określana dzięki funkcji *getOrder()*, przez co same obliczenia są łatwe. Dla każdego transformatora z aktywnej kombinacji oblicza się sumę mocy znamionowych, strat jałowych i strat obciążeniowych. Samą stratę mocy oblicza się zgodnie ze wzorem. Na koniec następuje wyznaczenie minimalnych i maksymalnych strat mocy. Najpierw zakłada się, że pierwsza wartość jest i maksymalna i minimalna. Potem za każdym razem sprawdza się, czy strata mocy jest mniejsza od minimum lub większa od maksimum. Jeśli tak, to ustawia się ją, jako nowe maksimum czy minimum i zapisuje jej indeks. Będzie to przydatne w trakcie wypisania wyników tekstowych.

Listing 6.12. Funkcja *getOrder()*

```
bool[] getOrder(int i)
{
     string orderString="";
     var orderBool=new bool[iT];
     orderString=Convert.ToString(i,2).PadLeft(iT,'0');
     for(int j=0;j<iT;j++)
     {
          orderBool[j]=orderString[iT-j-1]=='1';
     }
     return orderBool;
}
```

Funkcja *getOrder()* to alternatywa do określania, jaka kombinacja jest aktualnie sprawdzana przez obliczenia. Tutaj zastosowano podejście binarne. Jeśli ustawimy pętlę, w której ostatnim indeksem jest 2^*iT*, to indeksy iteracji w kodzie dwójkowym są kolejnymi kolejnościami transformatorów do sprawdzenia. Wynika to z natury liczb binarnych, którą łatwo potwierdzić matematycznie. Indeks 1 tu oznacza to, że sprawdzamy tylko transformator 1. Indeks 2 to binarne 10, co odpowiada sprawdzaniu tylko transformatora 2. Indeks 3 to binarne 11, czyli sprawdza się transformatory 1 oraz 2. To podejście jest znacznie łatwiejsze od zastosowanego rozwiązania w kodzie oryginalnym.

Listing 6.13. Funkcja *showResultsText()*

```
void showResultsText()
{
     buttonCopy.Enabled=true;
     buttonSave.Enabled=true;
     buttonPrint.Enabled=true;
     labelState.Text="Obliczenia zakonczone...";
     textResults.Text=@"-------------------------------------
   Praca Rownolegla Transformatorow
-------------------------------------

--------------------------
   Dane Transformatorow
--------------------------

                                  |";
     textInsert1("------",-1);
textResults.Text+="|"+nl+"                                    |";
     for(int i=0;i<iT;i++)
     {
          textInsert1(("Tr #"+(i+1)).PadRight(6,' '),i);
     }
textResults.Text+="|"+nl+"|------------------------------|";
     textInsert1("------",-1);
     textResults.Text+="|"+nl;
     textInsert2("Moc Znamionowa [kVA]",0);
     textInsert2("Straty Jalowe [kW]",1);
     textInsert2("Straty Obciazeniowe [kW]",2);
     textInsert2("Napiecie Zwarcia [%]",3);
     textInsert2("Napiecie Strony Pierwotnej [kV]",4);
     textInsert2("Napiecie Strony Wtornej [kV]",5);
```

```
31          textResults.Text+="|------------------------------|";
32          textInsert1("------",-1);
33          textResults.Text+="|"+nl+nl;
34          textResults.Text+=@"--------------------------------
35        Obciazenie Transformatorow
36     --------------------------------";
37          textResults.Text+=nl+nl+"
       |---------------------------------------|"+nl;
38          textResults.Text+="            |Moc  Znamionowa  [kVA]  Moc
       Obciazenia [kVA]|"+nl;
39     textResults.Text+="|------|-----------------------------------
       -----|"+nl;
40          for(int i=0;i<iT;i++)
41          {
42               if(dT[i,6]==1)
43               {
44          textResults.Text+=("|Tr #"+(i+1)).PadRight(7,' ');
45          textResults.Text+="|"+dT[i,0].ToString().PadLeft(20,' ');
46     textResults.Text+="
       "+Math.Round(So[i],2).ToString().PadLeft(20,' ')+"|";
47                    if(dT[i,0]<So[i])
48                    {
49                         textResults.Text+=" <- Przeciazenie";
50                    }
51                    textResults.Text+=nl;
52               }
53          }
54     textResults.Text+="|------|-----------------------------------
       -----|"+nl+nl;
55     textResults.Text+=@"----------------------
56        Straty Kombinacji
57     ----------------------"+nl+nl;
58     textResults.Text+="|-----------------------------------------
       ---------------|"+nl;
59     textResults.Text+="|Kombinacja [-]         Moc Kombinacji [kVA]
       Straty Mocy [kW]|"+nl;
60     textResults.Text+="|-----------------------------------------
       ---------------|"+nl;
61          for(int i=1;i<Math.Pow(2,iT);i++)
62          {
63               textInsert3(i);
64          }
65     textResults.Text+="|-----------------------------------------
       ---------------|";
66     }
```

Aby tekst wpisany do *textResults* wyglądał dobrze należy jednak wykonać sporo operacji, które wydają się być losowe na pierwszy rzut oka. Na początku odblokowuje się przyciski odpowiadające za kopiowanie, zapis i drukowanie wyników. Tekst stanu zmienia się na „Obliczenia zakończone...”, aby użytkownik wiedział, że wyniki są gotowe do analizy. Następnie wykonuje się dużo operacji na tekście w *textResults*. Pierwsze ustawienie tekstu wykonano za pomocą literału – w tym kontekście symbol @ oznacza, że następujący tekst ma zostać zrozumiany dosłownie tak jak jest widoczny. To umożliwia w łatwy sposób napisać dokładnie, co chcemy aby się pokazało w polu tekstowym. Pierwsza część jest czysto

wprowadzająca, aby użytkownik wiedział, że jest to są wyniki obliczeń programu. Potem jednak następuje stworzenie tabel z danymi kolejnych transformatorów, porównaniem mocy znamionowych z obciążeniem oraz na końcu zapis strat kombinacji. W tym momencie dobrze przypomnieć, że nowe funkcje tworzy się nie tylko by podzielić program na łatwe do zrozumienia fragmenty, ale również po to, aby nie kopiować tego samego kodu wielokrotnie.

Dlatego na potrzeby wpisywania stworzono cztery funkcje: *textInsert1()*, *textInsert2()*, *textInsert3()* i *textInsert4()*. *textInsert1()* wpisuje zadany tekst do *textResults* tyle razy, co aktualna ilość transformatorów lub tylko dla podanego transformatora. *textInsert2()* wpisuje zadany tekst w bardzo specyficznej konfiguracji i służy tylko do stworzenia wnętrza tabeli. *textInsert3()* działa podobnie, ale on tylko tworzy wnętrze tabeli strat kombinacji. Funkcja *textInsert4()* działa tak samo jak *textInsert2()*, tylko przy eksporcie pliku. Poprzez odpowiednio wypracowane kombinacje tych funkcji i dodawaniu tekstu ręcznie kształtuje się kolejne tabele. Pierwsza tabela posiada jedną boczną kolumnę z opisem danych transformatora oraz górny wiersz z kolejnymi numerami transformatorów. Z tego powodu najpierw stosuje się *textInsert1()* by stworzyć górne odgraniczenie, potem dodaje się niżej numery transformatorów, a jeszcze niżej opisy danych wraz z samymi danymi. Po utworzeniu pierwszej tabeli przechodzi się dalej i drugim literałem dodaje się napis o obciążeniu transformatorów. W drugiej tabeli transformatory są w bocznej kolumnie a moc znamionowa i obciążenia jest w górnym wierszu. Na koniec identycznie dodaje się tabelę ze stratami.

Listing 6.14. Wydarzenie *ExportDialogFileOk()*

```
1   void              ExportDialogFileOk(object              sender,
    System.ComponentModel.CancelEventArgs e)
2   {
3        resultexport=new StringBuilder();
4        resultexport.Append("Praca                        Rownolegla
    Transformatorow"+nl+nl);
5        resultexport.Append("Dane Transformatorow"+nl+"*");
6        for(int i=0;i<iT;i++)
7        {
8             if(dT[i,7]==1)
9             {
10                 resultexport.Append(("Tr  #"+(i+1)).PadLeft(6,'
    ')+"*");
11            }
12       }
13       textInsert4("Moc Znamionowa [kVA]",0);
14       textInsert4("Straty Jalowe [kW]",1);
15       textInsert4("Straty Obciazeniowe [kW]",2);
16       textInsert4("Napiecie Zwarcia [%]",3);
17       textInsert4("Napiecie Strony Pierwotnej [kV]",4);
18       textInsert4("Napiecie Strony Wtornej [kV]",5);
19       textInsert4("Grupa Polaczen [-]",6);
20       resultexport.Append(nl+nl+"*Moc    Znamionowa    [kVA]*Moc
    Obciazenia [kVA]*Obciazenie Zadane [kVA]"+nl);
21       for(int i=0;i<iT;i++)
22       {
```

```
23              resultexport.Append("Tr                                  #"+
    (i+1)+"*"+Math.Round(dT[i,0],2)+"*"+Math.Round(So[i],2)+"*"+Sza
    d+nl);
24          }
25          resultexport.Append(nl+nl+"Straty
    Kombinacji"+nl+"Kombinacja [-]*Moc Kombinacji [kVA]*Straty Mocy
    [kW]"+nl);
26          for(int i=1;i<Math.Pow(2,iT);i++)
27          {
28              sumaSzn=0;
29              activeOrder=getOrder(i);
30              for(int j=0;j<iT;j++)
31              {
32                  if(activeOrder[j])
33                  {
34                      sumaSzn+=dT[j,0];
35                      resultexport.Append(j+1);
36                  }
37                  else
38                  {
39                      resultexport.Append(" ");
40                  }
41              }
42  resultexport.Append("*"+Math.Round(sumaSzn,2)+"*"+Math.Round(dS
    [i],2));
43              if(i==idSmax)
44              {
45                  resultexport.Append("*<-Max");
46              }
47              if(i==idSmin)
48              {
49                  resultexport.Append("*<-Min");
50              }
51              resultexport.Append(nl);
52          }
53  File.WriteAllText(exportDialog.FileName,resultexport.ToString()
    );
54  MessageBox.Show("Eksport udany. Plik moze zostac importowany do
    arkusza obliczeniowego przy odgraniczeniu znakiem
    *","Eksport",MessageBoxButtons.OK,MessageBoxIcon.Information);
55  }
```

Sprawa eksportu również jest kwestią skomplikowaną. Nie da się łatwo zamienić tekstu wyświetlanego na plik otwieralny w arkuszu kalkulacyjnym, w związku z czym potrzeba stworzyć nowy plik tekstowy. Kiedy użytkownik potwierdzi wybranie eksportu, to zaczyna się wykonanie wydarzenia *ExportDialogFileOk()*. Tworzy się zmienna typu *StringBuilder*, w którą łatwo można dodawać kolejne łańcuchy znaków. Podobnie jak przy tekście wyświetlanym wpisuje się teksty odpowiadające tabelom („Dane Transformatorów”, „Straty Kombinacji” itp.), ale tutaj zamiast tabulatorów stosuje się znak ‘*’. Tym sposobem przy otwarciu pliku arkuszem kalkulacyjnym można zaznaczyć opcję znaku odgraniczającego, co wygodnie rozłoży kolejne wartości do odpowiednich pól. Pozostały format pliku działa tak samo jak w *textInsert2()*. Po wykonaniu pełnego tekstu do zmiennej zostaje on wpisany do pliku tekstowego i wyświetlony jest komunikat o zakończenia eksportu.

Listing 6.15. Funkcja *textInsert1()*

```
1   void textInsert1(string t,int i)
2   {
3         if(i==-1)
4         {
5               for(int j=0;j<iTmax;j++)
6               {
7                     if(dT[j,6]==1)
8                     {
9                           textResults.Text+=t;
10                    }
11              }
12        }
13        else
14        {
15              if(dT[i,6]==1)
16              {
17                    textResults.Text+=t;
18              }
19        }
20  }
```

Listing 6.16. Funkcja *textInsert2()*

```
1   void textInsert2(string s,int j)
2   {
3         textResults.Text+="|"+s.PadRight(31,' ')+"|";
4         for(int i=0;i<iT;i++)
5         {
6               if(j!=6)
7               {
8                     if(dT[i,7]==1)
9                     {
10  textResults.Text+=Math.Round(dT[i,j],2).ToString().PadLeft(4,'
    ').PadRight(6,' ');
11                    }
12              }
13              else
14              {
15                    string groupText="";
16                    switch(dT[i,6].ToString().Substring(0,1))
17                    {
18                          case "1":
19                                groupText="Yy";
20                                break;
21                          case "2":
22                                groupText="Yd";
23                                break;
24                          case "3":
25                                groupText="Dy";
26                                break;
27                          case "4":
28                                groupText="Dd";
29                                break;
30                          case "5":
```

```
31                                groupText="Yz";
32                                break;
33                           case "6":
34                                groupText="Dz";
35                                break;
36                      }
37    textResults.Text+=(groupText+dT[i,6].ToString().Substring(1).Tr
      imStart('0')).PadLeft(4,' ').PadRight(6,' ');
38                }
39           }
40           textResults.Text+="|"+nl;
41      }
```

Listing 6.17. Funkcja *textInsert3()*

```
 1    void textInsert3(int i)
 2    {
 3          sumaSzn=0;
 4          textResults.Text+="|";
 5          activeOrder=getOrder(i);
 6          for(int j=0;j<iT;j++)
 7          {
 8                if(activeOrder[j])
 9                {
10                      sumaSzn+=dT[j,0];
11          textResults.Text+=(j+1).ToString().PadRight(2,' ');
12                }
13                else
14                {
15                      textResults.Text+="  ";
16                }
17          }
18          for(int j=iT;j<iTmax;j++)
19          {
20                textResults.Text+="  ";
21          }
22          textResults.Text+=sumaSzn.ToString().PadLeft(21,' ')+" ";
23    textResults.Text+=Math.Round(dS[i],2).ToString().PadLeft(16,'
      ')+"|";
24          if(i==idSmin)
25          {
26                textResults.Text+=" <- Min";
27          }
28          if(i==idSmax)
29          {
30                textResults.Text+=" <- Max";
31          }
32          textResults.Text+=nl;
33    }
```

Listing 6.18. Funkcja *textInsert4()*

```
1   void textInsert4(string s,int j)
2   {
3         resultexport.Append(nl+s+"*");
4         for(int i=0;i<iT;i++)
5         {
6               if(j!=6)
7               {
8                     if(dT[i,7]==1)
9                     {
10                    resultexport.Append(Math.Round(dT[i,j],2)+"*");
11                    }
12              }
13              else
14              {
15                    string groupText="";
16                    switch(dT[i,6].ToString().Substring(0,1))
17                    {
18                          case "1":
19                                groupText="Yy";
20                                break;
21                          case "2":
22                                groupText="Yd";
23                                break;
24                          case "3":
25                                groupText="Dy";
26                                break;
27                          case "4":
28                                groupText="Dd";
29                                break;
30                          case "5":
31                                groupText="Yz";
32                                break;
33                          case "6":
34                                groupText="Dz";
35                                break;
36                    }
37    resultexport.Append(groupText+dT[i,6].ToString().Substring(1)+"
      *");
38              }
39        }
40  }
```

Cztery użyte funkcje można na szybko opisać razem. *textInsert1()* działa na zasadzie wstawiania tekstu za pomocą pętli *for*, albo tylko poprzez indeks transformatora. Z drugiej strony mamy *textInsert2()*, gdzie poprzednia funkcja została zastosowana w celu utworzenia wiersza składającego się z nazwy danej oraz wartości owej danej dla wiersza. Rozwinięte jest to o zamianę liczby grupy z powrotem na tekst grupy. Jeśli pierwsza wartość to 1 należy wpisać grupę Yy, jeśli 2 to Yd itd. Funkcja *textInsert3()* jest najbardziej rozbudowana. Ona sprawdza kolejne kombinacje i wypisuje zawarte w niej transformatory. Jeśli dany transformator jest nieaktywny to w tekście wpisuje się przerwę zamiast numeru. Jako moc kombinacji określa się sumę mocy znamionowych aktywnych transformatorów, a ta jest przedstawiana razem ze stratami mocy. Wartość mocy znamionowej i mocy obciążeniowej

wpisuje się do każdego wiersza. Dodatkowo, jeśli aktualny wiersz odpowiada najmniejszym lub największym stratom, to obok tabeli wpisuje się odpowiedni tekst indykujący. Funkcja *textInsert4()* zamienia wartość odpowiadającą grupie transformatora na faktyczny tekst tej grupy. Co prawda ona jest używana przy eksporcie, ale jej działanie jest zbliżone do pozostałych funkcji *textInsert*. Dodaje się znaki tekstowe dla sformatowania pliku, a potem tak jak w *textInsert2()* wpisuje się odpowiedni tekst grupy.

Listing 6.19. Wydarzenie *ButtonCopyClick()*

```
1    void ButtonCopyClick(object sender, EventArgs e)
2    {
3         Clipboard.SetText(textResults.Text);
4    MessageBox.Show("Skopiowano do schowka","Kopiowanie",MessageBoxButtons.OK,MessageBoxIcon.Information);
5    }
```

Listing 6.20. Wydarzenie *ButtonSaveClick()*

```
1    void ButtonSaveClick(object sender, EventArgs e)
2    {
3         saveDialog.ShowDialog();
4         saveDialog.FileName="Wyniki_"+count+".txt";
5    }
```

Listing 6.21. Wydarzenie *ButtonExportClick()*

```
1    void ButtonExportClick(object sender, EventArgs e)
2    {
3         exportDialog.FileName="Wyniki_Eksport.txt";
4         exportDialog.ShowDialog();
5    }
```

Listing 6.22. Wydarzenie *SaveDialogFileOk()*

```
1    void SaveDialogFileOk(object sender, System.ComponentModel.CancelEventArgs e)
2    {
3         File.WriteAllText(saveDialog.FileName,textResults.Text);
4    }
```

Kopiowanie, zapisywanie i eksportowanie uzyskanych wyników są względnie proste. Funkcja *Clipboard.SetText()* pozwala programistycznie ustawić aktualny tekst w schowku, a za pomocą obiektu *saveDialog* łatwo można łatwo poprosić użytkownika o wybranie lokalizacji i nazwy pliku do zapisu. Tak samo działa obiekt *exportDialog*. W innym przypadku zapisanie pliku byłoby trudne, ale dla samego tekstu wystarczy funkcja *File.WriteAllText()*.

Listing 6.23. Wydarzenie *ButtonPrintClick()*

```
1   void ButtonPrintClick(object sender, EventArgs e)
2   {
3        File.WriteAllText(tmpPath,textResults.Text);
4        if(printDialog.ShowDialog()==DialogResult.OK)
5        {
6             try
7             {
8                 reader=new StreamReader(tmpPath);
9                 try
10                {
11                    PrintDocument pd=new PrintDocument();
12  PrintPage+=new PrintPageEventHandler(this.pd_PrintPage);
13                    pd.Print();
14                }
15                finally
16                {
17                    reader.Close();
18                }
19            }
20            catch (Exception ex)
21            {
22                 MessageBox.Show(ex.Message);
23            }
24       }
25       File.Delete(tmpPath);
26  }
```

Z drukowaniem jednak nie jest tak prosto. Drukowanie rozpoczyna się od zapisu wyników w pliku tymczasowym na pulpicie komputera. Następnie wewnątrz warunku otwiera się dialog drukowania. Jeśli użytkownik zgodzi się na drukowanie, program próbuje zacząć drukowanie. Zmiennej *reader* przypisuje się plik tymczasowy. Do listy stron do drukowania dodaje się nową stronę za pomocą wydarzenia *pd_PrintPage*, po czym następuje drukowanie. Na koniec kończy się działanie zmiennej *reader* i usuwa plik tymczasowy. Jeśli nastąpi jakiś błąd, wyświetlona zostanie informacja użytkownikowi.

Listing 6.24. Wydarzenie *pd_printPage()*

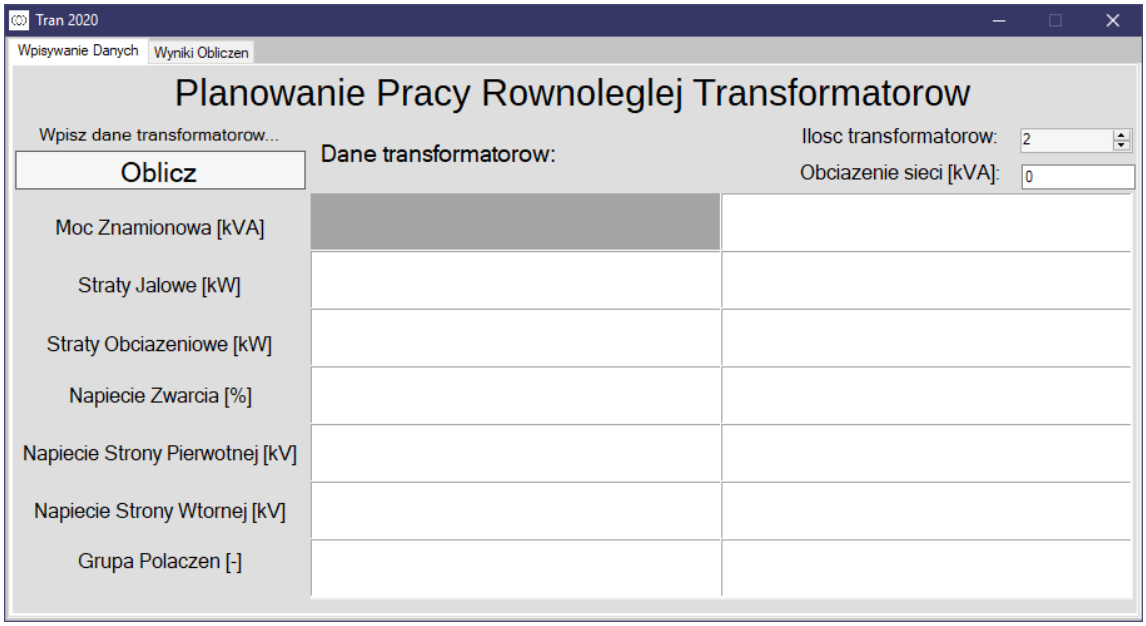
```
1   void pd_PrintPage(object sender, PrintPageEventArgs e)
2   {
3        float linesPerPage=0;
4        float y=0;
5        int index=0;
6        float leftMargin=e.MarginBounds.Left;
7        float topMargin=e.MarginBounds.Top;
8        string line=null;
9       Font font=new Font("Courier New", 11);
10  linesPerPage=e.MarginBounds.Height/font.GetHeight(e.Graphics);
11      while  (index<linesPerPage  &&((line=reader.ReadLine())  !=
    null))
12      {
13            y=topMargin+(index*font.GetHeight(e.Graphics));
14  e.Graphics.DrawString(line,font,Brushes.Black,leftMargin,y,new
    StringFormat());
```

```
15                  index++;
16          }
17          if (line!=null)
18          {
19                  e.HasMorePages=true;
20          }
21          else
22          {
23                  e.HasMorePages=false;
24          }
25      }
```

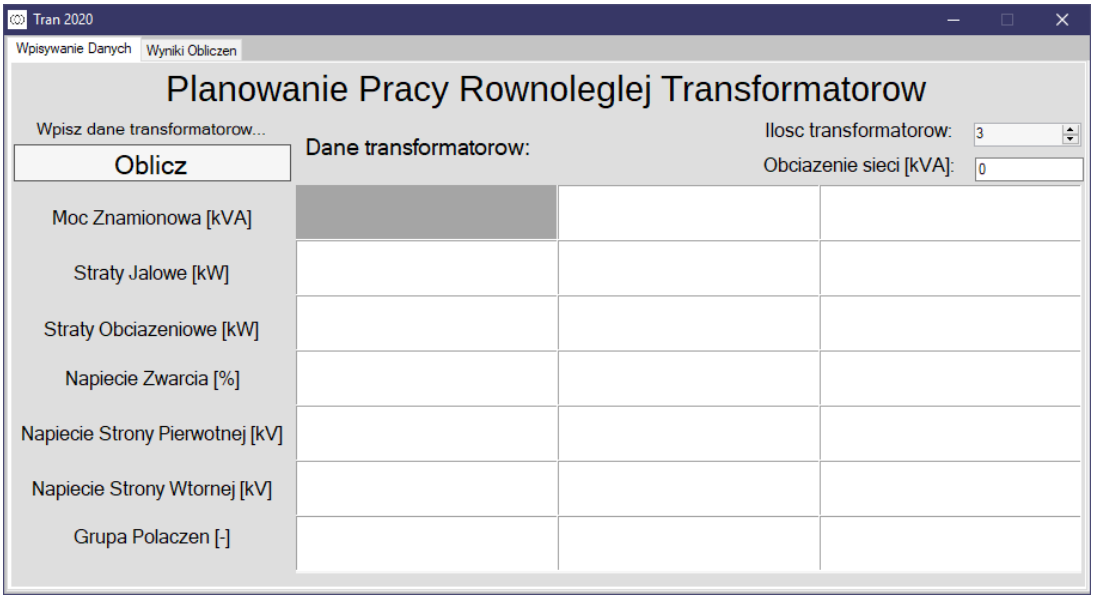
Obsługa drukowania jest skomplikowanym problemem [10]. W przypadku plików tekstowych to jest najłatwiejsze podejście. Tutaj potrzebne jest zdefiniowanie kilku zmiennych dodatkowych. *linesPerPage* to ilość linii tekstowych dostępnych na danej stronie. (jest to zależne od wybranej czcionki oraz jej wielkości), *y* oznacza aktualną współrzędną Y, na której ma zostać wydrukowany tekst, a *index* to aktualny numer drukowanej linii. Dla ułatwienia zostały dodatkowo zdefiniowane wartości marginesu lewego i górnego. W tym przypadku zastosuje się czcionkę *Courier New* o wielkości 11. Ilość linii na stronę można obliczyć jako wysokość strony drukowanej podzielona na wysokość czcionki. Następnie, wewnątrz pętli *while* następuje wydrukowanie kolejnych linii. Pierwszym warunkiem pętli jest sprawdzenie, czy aktualny indeks jest mniejszy od ustalonej ilości linii na stonę. Drugi warunek jednocześnie sprawdza, czy aktualna linia nie jest pusta oraz wykonuje operację odczytania tejże linii. Współrzędna Y tekstu można wyznaczyć jako współrzędną górnego marginesu i iloczyn indeksu z wysokością czcionki. Samo ustawienie strony wykonuje się za pomocą funkcji *Graphics.DrawString()*, po czym zwiększa się *index* i przechodzi do następnej linii. Gdy zostaną wydrukowane wszystkie linie danej strony, sprawdza się czy pozostał jeszcze tekst do drukowania. Jeśli tak, wysyła się informację do drukarki, że pozostały jeszcze strony (*e.HasMorePages*). Inaczej kończy się drukowanie i użytkownik może wrócić do ustawienia nowych transformatorów.

7 Przykład działania stworzonego programu Tran 2020

Stworzony program posiada pełny interfejs graficzny, dzięki czemu po uruchomieniu użytkownik ma dostęp do całości wymaganych danych. Poniżej tekstu tytułowego „Planowanie Pracy Równoległej Transformatorów" widoczny jest przycisk obliczania, macierz danych transformatorów, element definiujący ilość transformatorów oraz pole do wpisania obciążenia sieci. Dla porównania z oryginalnym programem możemy wykonać to samo zadanie obliczeniowe, co wcześniej. Domyślnie na początku wybrane są dwa transformatory, ale naciskanie strzałek w górę i w dół może tą ilość zmienić. Ustawmy trzy transformatory, tak jak w rozdziale 4.2.

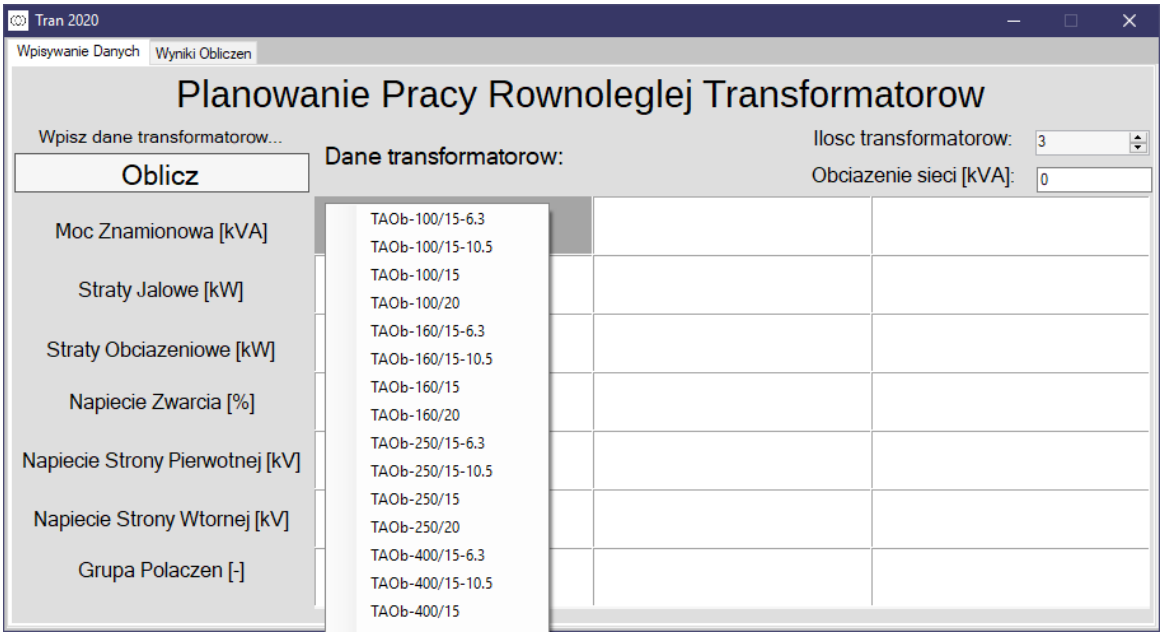


Rys. 7.1. Ekran programu po uruchomieniu [opracowanie własne]

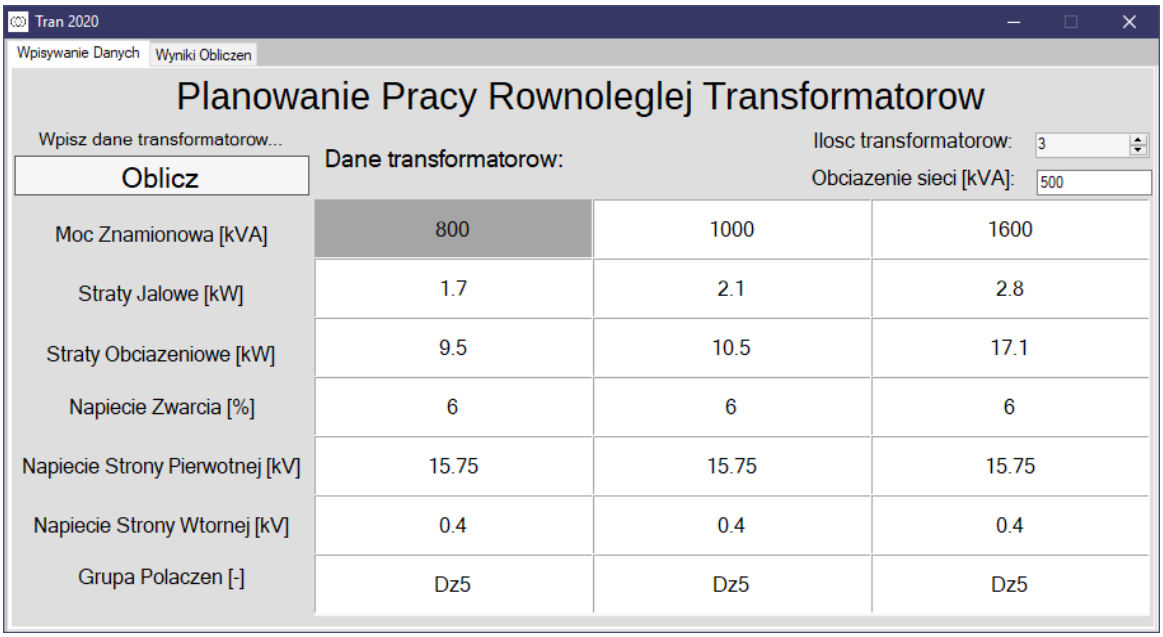


Rys. 7.2. Ekran programu po zmianie ilości transformatorów [opracowanie własne]

Wybrana ilość może zostać zmieniona w każdym momencie jednym przyciskiem. Wpisanie danych można wykonać na dwa sposoby. Tak samo jak w poprzednim programie można wpisywać wartości ręcznie. Tutaj nie ma ograniczenia, co do kolejności, można dowolnie przeskakiwać pomiędzy polami macierzy. Z drugiej strony jednak można stosować bibliotekę transformatorów. Kliknięcie prawym przyciskiem myszki otwiera pasek narzędzi z listą transformatorów, których dane są już w bibliotece. Kliknięcie na nazwę transformatora wpisze jego dane do wybranej kolumny.

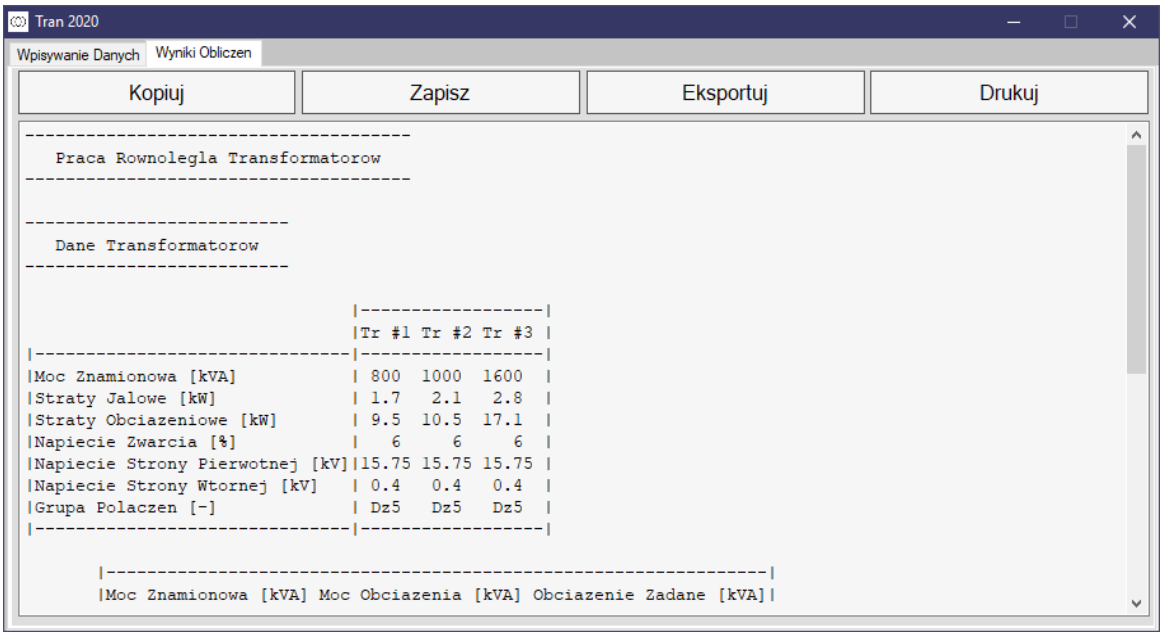


Rys. 7.3. Ekran programu z fragmentem paska narzędzi [opracowanie własne]



Rys. 7.4. Ekran programu po wstawieniu danych transformatorów i obciążenia sieci [opracowanie własne]

Po wpisaniu danych oraz odpowiedniego obciążenia sieci można nacisnąć przycisk Oblicz. Wyniki obliczeń są dostępne na zakładce Wyniki Obliczeń, gdzie dostępne są kolejne opcje zarządzania danymi.



Rys. 7.5. Ekran programu po przejściu do zakładki Wyniki Obliczeń [opracowanie własne]

Tutaj jest dostępny sformatowany w kodzie tekst wykonanych obliczeń, wraz z opcjami zarządzania. W przeciwieństwie do programu oryginalnego tutaj wszystkie dane są dostępne od razu, nawet wartości obliczonych strat dla kombinacji.

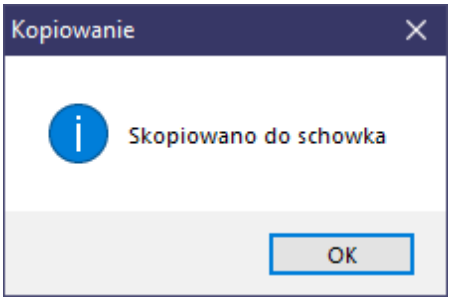
```
       |-------------------------------------------------------------------|
       |Moc Znamionowa [kVA] Moc Obciazenia [kVA] Obciazenie Zadane [kVA]|
|------|-------------------------------------------------------------------|
|Tr #1 |       800               117.65                      500          |
|Tr #2 |      1000               147.06                      500          |
|Tr #3 |      1600               235.29                      500          |
|------|-------------------------------------------------------------------|

------------------------
   Straty Kombinacji
------------------------

|-----------------------------------------------------------|
|   Kombinacja [-]    Moc Kombinacji [kVA] Straty Mocy [kW]|
|-----------------------------------------------------------|
|1                          800               5.41         |
|  2                       1000               4.72         |
|1 2                       1800               5.34         |
|    3                     1600               4.47         | <- Min
|1   3                     2400               5.65         |
|  2 3                     2600               5.92         |
|1 2 3                     3400                7.4         | <- Max
|-----------------------------------------------------------|
```

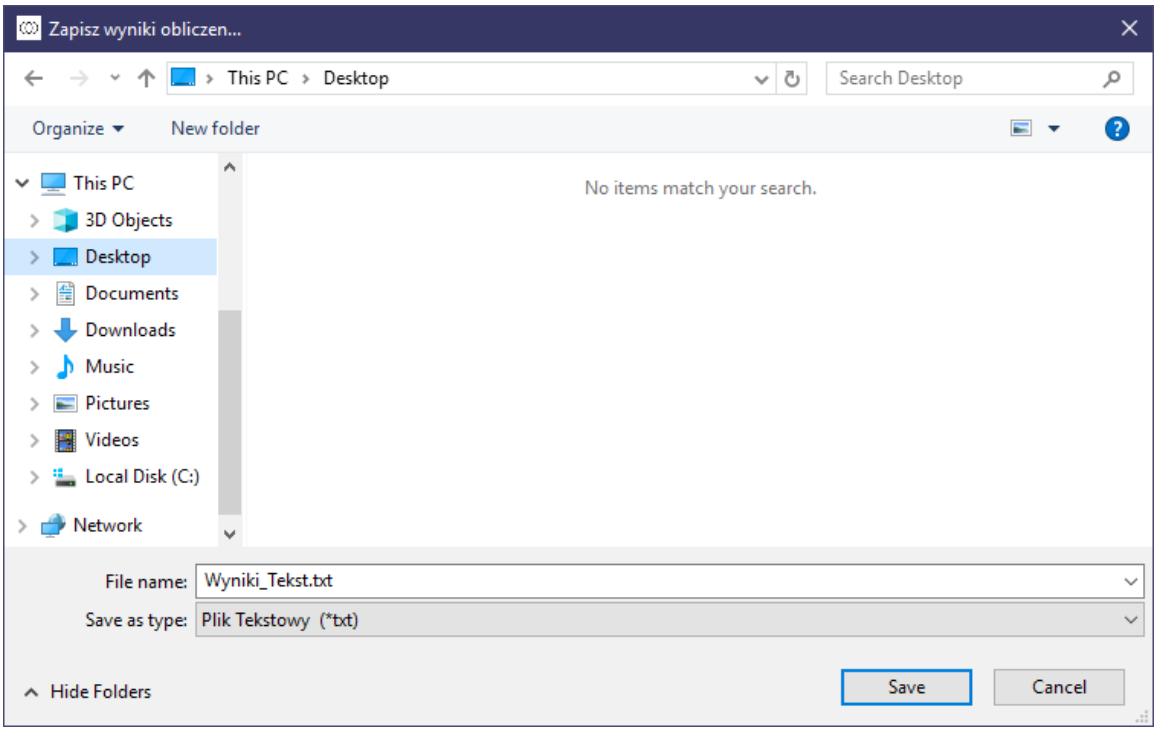
Rys. 7.6. Tabele mocy obciążeń oraz strat kombinacji [opracowanie własne]

Porównanie z oryginalnym programem pokazuje, że wyniki obliczeń są identyczne. Obciążenie sieci rozłożyło się proporcjonalnie na transformatorach, najniższe straty występują przy wykorzystaniu transformatora TAOb-1600/15, a maksymalne przy włączeniu wszystkich trzech. Po wykonaniu obliczeń wyniki można skopiować do schowka (równoznaczne z zaznaczeniem tekstu wyników i skopiowania przez ctrl+c), zapisać do pliku tekstowego, eksportować do pliku w formacie łatwym do importowania w arkuszu kalkulacyjnym lub wydrukować je na papierze. Naciśnięcie przycisku „Kopiuj" skopiuje wyniki obliczeń do schowka i wyświetli komunikat o wykonaniu tej akcji. Jest to przydatne w przypadku dużej ilości transformatorów, gdzie zaznaczanie setek możliwych kombinacji jest niewygodne.



Rys. 7.7. Komunikat o wykonaniu skopiowania [opracowanie własne]

Zapis do pliku jest szybkim sposobem ściągnięcia danych do pliku, zwłaszcza jeśli użytkownik chce szybko wykonać wiele analiz pod rząd. Naciśnięcie przycisku „Zapisz" otworzy dialog zapisu pliku, gdzie użytkownik ma możliwość wyboru gdzie i pod jaką nazwą zapisać wyniki obliczeń.



Rys. 7.8. Dialog zapisu pliku [opracowanie własne]

W celu zautomatyzowania procesu analizy możliwe jest również eksportowanie wyników obliczeń w formacie łatwym do zaimportowania do arkusza kalkulacyjnego. Należy zauważyć, że nie jest to pełny eksport – nadal uzyskuje się plik tekstowy zawierający dane z obliczeń.

Listing 7.1. Zawartość pliku eksportowanych wyników

```
1     Praca Rownolegla Transformatorow
2
3     Dane Transformatorow
4     * Tr #1* Tr #2* Tr #3*
5     Moc Znamionowa [kVA]*800*1000*1600*
6     Straty Jalowe [kW]*1.7*2.1*2.8*
7     Straty Obciazeniowe [kW]*9.5*10.5*17.1*
8     Napiecie Zwarcia [%]*6*6*6*
9     Napiecie Strony Pierwotnej [kV]*15.75*15.75*15.75*
10    Napiecie Strony Wtornej [kV]*0.4*0.4*0.4*
11    Grupa Polaczen [-]*Dz5*Dz5*Dz5*
12
13    *Moc Znamionowa [kVA]*Moc Obciazenia [kVA]*Obciazenie Zadane
      [kVA]
14    Tr #1*800*117.65*500
15    Tr #2*1000*147.06*500
16    Tr #3*1600*235.29*500
17
18
19    Straty Kombinacji
20    Kombinacja [-]*Moc Kombinacji [kVA]*Straty Mocy [kW]
21    1  *800*5.41
22     2 *1000*4.72
23    12 *1800*5.34
24       3*1600*4.47*<-Min
25    1 3*2400*5.65
26     23*2600*5.92
27    123*3400*7.4*<-Max
```

Uzyskany plik tekstowy wygląda nietypowo, gdy jest on otwarty bez kontekstu. Zawiera on wiele znaków ‘*’ w miejscach, gdzie powinny być spacje. Zostało to zrobione z powodu mechaniki importu plików do arkuszy kalkulacyjnych. Aby zachować odpowiednie odstępy pomiędzy kolejnymi polami nie możemy używać spacji, ponieważ to rozdzieliłoby teksty opisowe. Dla przykładu tekst „Moc Znamionowa [kVA]” zostałby podzielony na trzy pola, przez co tabela rozłożyłaby się nierównomiernie z wynikami. Rozwiązaniem jest dodanie nowego znaku odgraniczającego, który zostanie umieszczony dokładnie w miejscach przejścia do kolejnego pola. Tutaj tą rolę spełnia znak ‘*’, nieużywany nigdzie indziej w obliczeniach. Tym sposobem import danych do arkusza obliczeniowego jest tylko kwestią wyboru tego znaku.

	A	B	C	D
1	Praca Rownolegla Transformatorow			
2				
3	Dane Transformatorow			
4		Tr #1	Tr #2	Tr #3
5	Moc Znamionowa [kVA]	800	1000	1600
6	Straty Jalowe [kW]	1.7	2.1	2.8
7	Straty Obciazeniowe [kW]	9.5	10.5	17.1
8	Napiecie Zwarcia [%]	6	6	6
9	Napiecie Strony Pierwotnej [kV]	15.75	15.75	15.75
10	Napiecie Strony Wtornej [kV]	0.4	0.4	0.4
11	Grupa Polaczen [-]	Dz5	Dz5	Dz5
12				
13		Moc Znamionowa [kVA]	Moc Obciazenia [kVA]	Obciazenie Zadane [kVA]
14	Tr #1	800	117.65	500
15	Tr #2	1000	147.06	500
16	Tr #3	1600	235.29	500
17				
18				
19	Straty Kombinacji			
20	Kombinacja [-]	Moc Kombinacji [kVA]	Straty Mocy [kW]	
21	1	800	5.41	
22	2	1000	4.72	
23	12	1800	5.34	
24	3	1600	4.47	<-Min
25	1 3	2400	5.65	
26	23	2600	5.92	
27	123	3400	7.4	<-Max

Rys. 7.9. Wyniki obliczeń zaimportowane do arkusza kalkulacyjnego [opracowanie własne]

Ostatnią opcją jest drukowanie. Działa ono tak samo jak w przypadku oryginalnego programu. Cały tekst wyników obliczeń bierze się, jako bazowy tekst do wydrukowania. Po wykonaniu drukowania powinien on wyglądać tak samo jak w programie.

```
-------------------------------------
   Praca Rownolegla Transformatorow
-------------------------------------

-------------------------
   Dane Transformatorow
-------------------------

                               |------------------|
                               |Tr #1 Tr #2 Tr #3 |
|------------------------------|------------------|
|Moc Znamionowa [kVA]          | 800  1000  1600  |
|Straty Jalowe [kW]            | 1.7   2.1   2.8  |
|Straty Obciazeniowe [kW]      | 9.5  10.5  17.1  |
|Napiecie Zwarcia [%]          |   6     6     6  |
|Napiecie Strony Pierwotnej [kV]|15.75 15.75 15.75 |
|Napiecie Strony Wtornej [kV]  | 0.4   0.4   0.4  |
|Grupa Polaczen [-]            | Dz5   Dz5   Dz5  |
|------------------------------|------------------|

       |-----------------------------------------------------------------|
       |Moc Znamionowa [kVA] Moc Obciazenia [kVA] Obciazenie Zadane [kVA]|
|------|-----------------------------------------------------------------|
|Tr #1 |       800               117.65                      500       |
|Tr #2 |      1000               147.06                      500       |
|Tr #3 |      1600               235.29                      500       |
|------|-----------------------------------------------------------------|

-----------------------
   Straty Kombinacji
-----------------------

|-----------------------------------------------------|
|   Kombinacja [-]    Moc Kombinacji [kVA] Straty Mocy [kW]|
|-----------------------------------------------------|
|1                          800              5.41         |
|  2                       1000              4.72         |
|1 2                       1800              5.34         |
|    3                     1600              4.47         | <- Min
|1   3                     2400              5.65         |
|  2 3                     2600              5.92         |
|1 2 3                     3400               7.4         | <- Max
|-----------------------------------------------------|
```

Rys. 7.10. Skan wydrukowanych wyników obliczeń [opracowanie własne]

Po zarządzeniu wynikami można przejść do obliczenia obciążeń transformatorów oraz strat mocy dla pozostałych wartości obciążenia sieci. Ponieważ te działania wyglądają identycznie do poprzednich, nie będą one ponownie tłumaczone. Wyniki obliczeń przedstawiono w poniższej tabeli.

Tab. 7.1. Zestawienie wyników obliczeń obciążeń transformatorów

Obciążenie Sieci	Obciążenie TAOb–800/15	Obciążenie TAOb–1000/15	Obciążenie TAOb–1600/15
[kVA]	[kVA]	[kVA]	[kVA]
500	117.65	147.06	235.29
850	200	250	400
1100	258.82	323.53	517.65
2700	635.29	794.12	1270.59
3250	764.71	955.88	1529.41
3400	800	1000	1600
3000	705.88	882.35	1411.76
1700	400	500	800
750	176.47	220.59	352.94

Tab. 7.2. Zestawienie wyników obliczeń strat mocy

Obciążenie Sieci	Optymalna Kombinacja	Optymalne Straty	Najgorsza Kombinacja	Najgorsze Straty
[kVA]	[–]	[kW]	[–]	[kW]
500	3	4.47	1 + 2 + 3	7.4
850	3	7.63	2	9.69
1100	2 + 3	9.84	1 + 2	11.27
2700	1 + 2 + 3	30	1 + 2 + 3	30
3250	1 + 2 + 3	40.5	1 + 2 + 3	40.5
3400	1 + 2 + 3	43.7	1 + 2 + 3	43.7
3000	1 + 2 + 3	35.48	1 + 2 + 3	35.48
1700	1 + 2 + 3	15.88	1 + 2	21.64
750	3	6.56	1	10.05

Wyniki obliczeń są identyczne do wyników z programu tran.exe, co potwierdza prawidłowe wykonywanie obliczeń wewnątrz samego programu. Dużą pomocą w wykonaniu tego zadania była biblioteka transformatorów załączona do programu. Jeśli istnieje potrzeba, można do niej wpisać nowe transformatory zgodnie z formatem poprzednich. Przykład pokazano poniżej.

Listing 7.2. Format transformatora w pliku transformatory.txt

```
1     Transformator
2     TAOb-100/15-6.3
3     100
4     0.4
5     1.76
6     4.5
7     6.3
8     0.4
9     Yz5
```

Format transformatora w bibliotece jest prosty. Każdy transformator wymaga użycia słowa kluczowego „Transformator” w jednej linii tekstu na końcu pliku. To słowo jest wyszukiwane przez program przy inicjalizacji programu, bez niego transformator nie zostanie wpisany. Kolejne linie po słowie kluczowym to kolejne dane tego transformatora: nazwa, moc znamionowa, jałowe straty mocy, obciążeniowe straty mocy, napięcie zwarcia, napięcie strony pierwotnej, napięcie strony wtórnej oraz grupa połączeń transformatora. Po wpisaniu danych należy zapisać plik oraz zrestartować program (jeśli był otwarty). Przy kolejnym otwarciu dodany transformator powinien się znaleźć na końcu listy z paska narzędzi.

8 Wnioski

Wykonany program **Tran 202**0 jest w pełni udaną aktualizacją programu **tran.exe**. Stworzony program, jako program oknowy nie jest ograniczony przez naturę konsoli. Użytkownik ma możliwość wpisywania danych w dowolnej kolejności. Dodatkowo zaimplementowana biblioteka transformatorów pozwala na wpisanie danych popularnych transformatorów za pomocą jednego kliknięcia myszki. Jeśli potrzeba, użytkownik może też dodać brakujące transformatory do biblioteki ręcznie. Program oblicza wszystkie potrzebne wartości naraz. W przeciwieństwie do programu oryginalnego wartości strat danych są dostępne od razu po wykonaniu obliczeń mocy obciążeniowych transformatorów. Wynikami obliczeń można zarządzać na kilka sposobów. Za pomocą jednego przycisku da się skopiować całą zawartość zakładki z wynikami do schowka. Użytkownik potrafi również automatycznie zapisać wyniki do pliku tekstowego. Kolejną opcja jest eksport, dzięki któremu importowanie danych do arkusza kalkulacyjnego jest proste. Na koniec istnieje możliwość wydrukowania wyników przy użyciu nowych drukarek.

Aktualizacja programu umożliwia wykonywanie obliczeń warunków pracy równoległej znacznie szybciej i wygodniej niż w programie bazowym. Szybka zmiana danych pozwala na zrobienie znacznie większej ilości obliczeń w danym czasie. Poza tym, wyniki obliczeń strat mocy nie wymagają drukowania do analizy, co oszczędza ilość papieru. W programie oryginalnym sprawdzenie tych danych wymaga marnowania materiału dla samego sprawdzenia, czy wybrane połączenie transformatorów jest efektywne. Opcja eksportu daje możliwość szybkiego wprowadzania danych do arkuszy operacyjnych, co da się zautomatyzować po stronie tychże arkuszy (np. operacje macro w Excel). Wykonany program Tran 2020 jest pełną, udaną aktualizacją programu tran.exe na nowe komputery.

9 Literatura

[1] Kaszowska B., Kucharska B., „*Laboratorium sieci i systemów elektroenergetycznych*”, Politechnika Opolska, Skrypt nr 190, Opole 1996.

[2] Demenko A., „*Praca Równoległa Transformatorów*” [online]. [dostęp: 17 marca 2003]. Dostępny w Internecie: <zme.iee.put.poznan.pl/wp-content/uploads/2017/11/lab_bme/rown.htm>

[3] Zientek E., „*Loading Considerations when Paralleling Transformers*”, Square D Engineering Services, Nr. 1, 2011, pp. 3-9.

[4] Plamitzer A., „Maszyny Elektryczne”, WNT 1972.

[5] Sołbut A., „*Maszyny Elektryczne 1*”, Oficyna Wydawnicza Politechniki Białostockiej, Nr. 1, 2017, p. 44.

[6] IBM Corporation, „*Operating System Technical Reference Volume 1*” [online]. [dostęp: 01 września 1987]. Dostępny w Internecie: <http://bitsavers.informatik.uni-stuttgart.de/pdf/ibm/pc/os2/84X1434_OS2_Technical_Reference_Volume_1_Sep87.pdf>

[7] Mandell S., „*TURBO Pascal Programming Today*”, West Publishing Company, 1979

[8] Feuer A., Gehani N., „*A Comparison of the Programming Languages C and Pascal*”, Bell Laboratories, 1982

[9] Wille C., „*SharpDevelop Repository*” [online]. [dostęp: 12 październiknika 2019]. Dostępny w Internecie: <https://github.com/icsharpcode/SharpDevelop>

[10] Gold M., „*Printing in C#*” [online]. [dostęp: 28 maja 2019]. Dostępny w Internecie: <https://www.c-sharpcorner.com/article/printing-in-C-Sharp>