

2. ETAPY PROCESU OPRACOWANIA PROGRAMU

Na rysunku 2.1 pokazano graf procesu opracowania programu w języku Turbo Pascal. Pierwszy etap to *opracowanie koncepcyjne*. Przede wszystkim należy dobrze sformułować zadanie, to znaczy dokładnie określić przeznaczenie programu i sposób posługiwania się nim przez przyszłego użytkownika. Następnie trzeba wybrać jeden ze znanych algorytmów wykonania podobnych zadań, lub opracować własny algorytm. Z wyborem algorytmu wiąże się ściśle wybór struktur pamięciowych, w których przechowywane będą dane wejściowe, wyniki pośrednie i rezultaty obliczeń. Wybór algorytmu i struktur danych w zasadniczy sposób wpłynie na cechy decydujące o jakości programu: szybkość obliczeń i zajętość pamięci, natomiast właściwe założenia, dotyczące sposobów komunikacji programu z użytkownikiem, sprawiają, że program będzie „przyjazny”, a więc wygodny i łatwy w eksploatacji.

Trzy następne etapy procesu przygotowania programu są realizowane za pomocą tzw. *środowiska programowego* (ang. *Integrated Development Environment*) o nazwie *Turbo Pascal 7.0*. Jest to złożone narzędzie, wspomagające opracowanie programu w fazach jego edycji, kompilacji i wykonania.

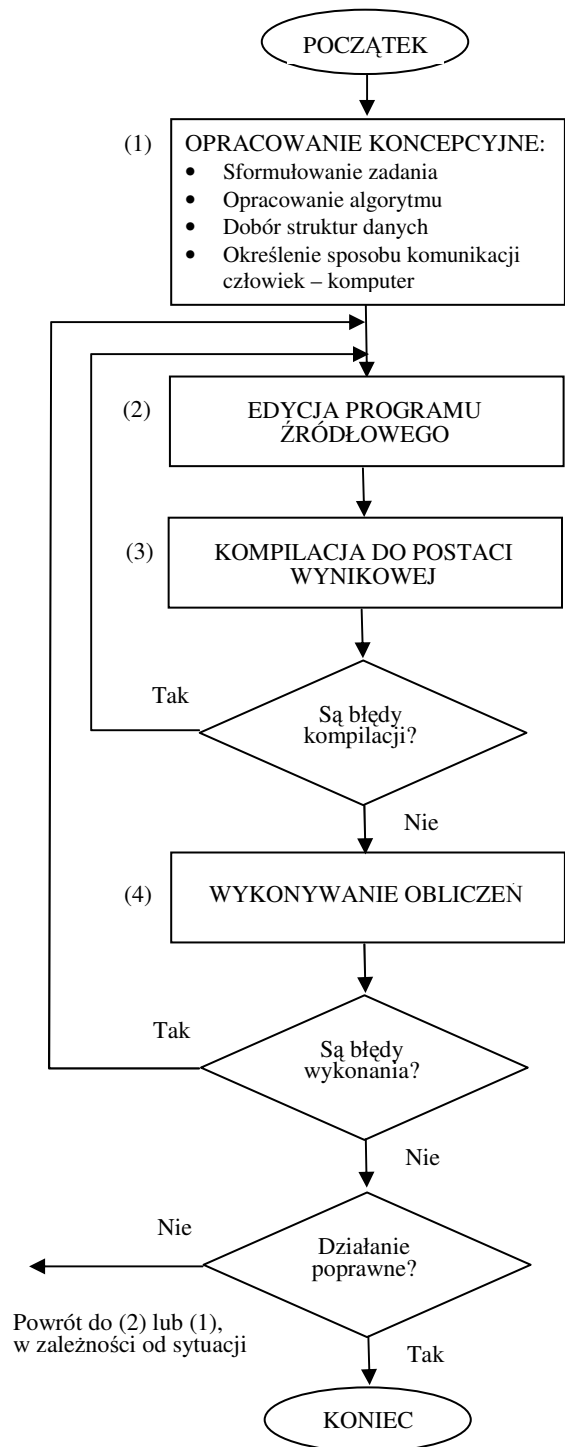
Edycja to pisanie źródłowego tekstu programu. Program, w tym pomocny, to *edytor*. Edytor daje szereg możliwości – ułatwia szybkie przechodzenie kursora no początek lub koniec linii i tekstu, utrzymuje pozycję wcięć wierszy tekstu, wyróżnia pogrubioną czcionką słowa kluczowe, umożliwia kopiowanie, przenoszenie i usuwanie fragmentów tekstu itp. Można pracować nad plikiem już istniejącym (opcja *File/Open*) lub zacząć pisać program od początku (opcja *File/New*). Możliwa jest jednoczesna praca nad kilkoma programami, prezentowanymi w różnych oknach (opcja *Window/Cascade* lub *Window/Tile*). Edytor automatycznie nadaje nowemu programowi nazwę *Noname00.pas*, którą oczywiście można zmienić przy jego zapamiętaniu, stosując polecenie *File/Save as*. Rozszerzenie „.pas” w nazwie pliku informuje o tym, że plik jest programem źródłowym w Pascalu.

Po napisaniu źródłowego tekstu programu następuje jego *kompilacja*, to jest tłumaczenie na język wewnętrzny, czyli przekształcenie do postaci wynikowej. Na ogół, (zwłaszcza dotyczy to niedoświadczonych programistów), kompilacja nie od razu przebiega pomyślnie, ponieważ w tekście programu trafiają się błędy składniowe, będące wynikiem pomyłek lub niedokładnej znajomości języka. Środowisko programowe pomaga w identyfikacji i określeniu błędu. Proces kompilacji ulega przerwaniu i na ekranie pojawia się komunikat w języku angielskim o rodzaju *błąd kompilacji*. Ponadto, położenie kursora wskazuje wiersz, w którym zidentyfikowano błąd. Poniżej pokazano kilka przykładowych komunikatów:

```
Error 3: Unknown identifier.      {Nieznana nazwa.}
Error 26: Type mismatch.         {Niewłaściwy typ danej.}
Error 85: „;” expected.         {Brakuje średnika.}
Error 113: Error in statement.   {Błąd w zapisie instrukcji.}
```

W przypadku wystąpienia błędu kompilacji, następuje powrót do fazy edycji w celu usunięcia błędu. Proces usuwania kolejnych błędów powtarzamy cyklicznie, aż do pojawienia się komunikatu „*Compile successful*” (Pomyślna kompilacja). Od tej chwili w pamięci operacyjnej znajduje się gotowy do wykonania program wynikowy, zapisany jako plik o tej samej nazwie, co plik źródłowy, lecz z rozszerzeniem „.exe”.

Wykonanie skompilowanego programu jest inicjowane poleceniem *Run/Run*. W toku wykonywania programu może wystąpić *błąd wykonania*. Ma to miejsce w wyniku niewłaściwej operacji, niewłaściwego formatu danych, nielegalnej operacji na pliku dyskowym, lub innych okoliczności, uniemożliwiających kontynuowanie obliczeń. Po wystąpieniu błędu wykonania program automatycznie ulega przerwaniu, a na ekranie pojawia się komunikat o rodzaju błędu. Poniżej pokazano trzy przykłady takich komunikatów:



Rys 2.1. Etapy opracowania programu w Turbo Pascalu

Error 104: File not open for input. {Nie otwarto pliku do zapisu.}
Error 106: Invalid numeric format. {Niewłaściwy zapis liczby.}
Error 200: Division by zero. {Próba dzielenia przez zero.}

Jeżeli przyczyną błędu wykonania był niewłaściwy zapis danej, lub nielegalna operacja plikowa, to należy powrócić do etapu (2) i tak zmodyfikować program źródłowy, by kontrolował takie sytuacje. Jeżeli błąd był skutkiem próby wykonania zakazanej operacji arytmetycznej, to może okazać się konieczna modyfikacja algorytmu tj. powrót do etapu (1).

Zakończenie pracy programu bez wystąpienia błędów wykonania nie daje pewności, że jest on poprawny. Trzeba jeszcze ocenić, czy wyniki obliczeń są prawidłowe. Proces sprawdzania wyników działania nazywa się *testowaniem* programu.

W przypadku bardzo prostych programów obliczeniowych porównuje się wyniki uzyskane dla wybranych wartości danych z wynikami uzyskanymi w wyniku obliczeń ręcznych. Trzeba również zbadać zachowanie się programu w przypadku wprowadzania danych o niewłaściwych wartościach. Program powinien sygnalizować takie sytuacje – w przeciwnym przypadku mogą wystąpić błędy wykonania, lub wyniki obliczeń mogą okazać się niepoprawne.

W przypadku złożonych programów o znacznej liczbie danych o ciągłych zakresach wartości, testowanie jest złożonym procesem, który zmniejsza prawdopodobieństwo pozostawienia błędów w programie, ale nie daje pewności ich całkowitej eliminacji. Proces testowania obejmuje:

- przygotowanie zestawów danych,
- uruchomienie programu dla różnych kombinacji danych wejściowych i różnych funkcji programu,
- analizę otrzymanych wyników w celu wykrycia błędnych rezultatów,
- analizę możliwych przyczyn wystąpienia błędów.

Stosowane są dwa podejścia do testowania programu:

- Testowanie *funkcjonalne* polega na sprawdzeniu poprawności wykonania poszczególnych opcji (funkcji) programu dla odpowiednio dobranych zestawów danych wejściowych.
- Testowanie *strukturalne* polega na takim uruchamianiu programu, by każda jego instrukcja została przynajmniej raz wykonana. Kolejne testy muszą zapewnić przejście przez wszystkie możliwe ścieżki grafu programu, wynikające z istniejących w nim rozgałęzień.

Czas przeznaczony na testowanie programu jest ograniczony, a liczba możliwych wartości danych uniemożliwia kompletne przetestowanie programu. Dlatego w praktyce zdarza się, że niektóre błędy wykrywane są w profesjonalnych programach dopiero po ich oddaniu do eksploatacji.