Andrzej Miszczyk

Wprowadzenie do programowania

1. Podstawowe pojęcia

Oprogramowanie

Pojęcie **oprogramowanie** często utożsamiane jest z programami komputerowymi. Należy jednak pamiętać, że w skład oprogramowania oprócz programów wchodzi związana z nimi dokumentacja oraz dane konfiguracyjne niezbędne do zgodnej z założeniami pracy programu.

Niniejszy rozdział ma na celu wprowadzić czytelnika w podstawowe zagadnienia związane z **implementacją oprogramowania**, czyli procesem zamiany opracowanego algorytmu działania na **program wynikowy (wykonywalny)**. Program ten może być uruchomiony na komputerze w celu wykonania postawionego przed nim zadania, zawiera zapisane binarnie instrukcje języka maszynowego komputera.

Programowanie i języki programowania

W ramach implementacji oprogramowania pojawia się **programowanie**, jako czynność wykonywana przez człowieka (programistę). Programowanie może być realizowane przy pomocy określonych metod i języków programowania.

Język programowania pozwala na dogodny dla człowieka zapis realizowanego zadania i sposobu jego wykonania przez komputer. Na język programowania składa się składnia i semantyka, czyli zestaw precyzyjnych reguł tworzenia instrukcji, definiujących podstawowe w danym języku operacje. Instrukcje są budowane z określonego zestawu słów kluczowych (np. if, const, while, repeat), znaków operatorów arytmetycznych i logicznych, stałych liczbowych oraz symbolicznych nazw zmiennych.

Zapisany w postaci jednego lub wielu plików (najczęściej tekstowych) ciąg instrukcji tworzący program, nazywamy programem źródłowym lub kodem źródłowym programu (ang. source program, source code).

Języki programowania dzieli się na dwie zasadnicze grupy:

- Języki niskiego poziomu,
- Języki wysokiego poziomu.

Język niskiego poziomu

Język niskiego poziomu zwany także **językiem asemblera**, jest ściśle związany z typem jednostki centralnej (procesora) i jego architekturą. Język ten powstał na bazie języka maszynowego poprzez zastąpienie kodów maszynowych pisanych binarnie lub szesnastkowo ich mnemonicznymi odpowiednikami. Dzięki zapisowi rozkazów procesora w postaci mnemonicznej można pisać programy zrozumiałe dla człowieka, a dodatkowo bardzo wydajne. Programowanie w języku asemblera jest bardzo pracochłonne i wymaga dobrej znajomości architektury konkretnego systemu mikroprocesorowego.

Obecnie praktycznie nie używa się języka asemblera do pisania całych programów dla komputerów osobistych. Zazwyczaj stosuje się go do pisania fragmentów wymagających wysokiej wydajności lub mających inne szczególne wymagania. Dla małych systemów mikroprocesorowych, zbudowanych przykładowo w oparciu o mikrokontrolery (mikrokomputery jednoukładowe), język asemblera może być jedynym dostępnym językiem programowania.

Poniżej przedstawiono przykład kodu źródłowego w języku asemblera dla procesorów o architekturze Intel 80x86 napisanego w składni Intel.

```
move acc, 10          #kopiuje 10 do akumulatora
add acc, 5            #dodaje 5 do akumulatora
move EF21A29C, acc    #kopiuje zawartość akumulatora do komórki pamięci
```

Język wysokiego poziomu

Język wysokiego poziomu jest bardziej przejrzysty dla człowieka w porównaniu z językiem asemblera, ponieważ instrukcje zapisywane są w postaci zbliżonej do języka naturalnego. Instrukcje budowane są z zestawu słów kluczowych danego języka, znaków operatorów różnego typu (arytmetycznych, logicznych, warunkowych, przypisania, itd.) oraz znaków specjalnych. Kod źródłowy programu zapisuje się zazwyczaj w postaci pliku tekstowego, z którego po poddaniu procesowi translacji powstaje program wykonywalny. Jednej instrukcji języka wysokiego poziomu odpowiada zazwyczaj seria instrukcji w języku niskiego poziomu.

2. Translatory, konsolidatory i maszyny wirtualne

Translator

Program wynikowy zrozumiały dla komputera, powstaje w procesie tłumaczenia kodu źródłowego, przez program zwany **translatorem**. Translator tworzony jest dla konkretnego języka programowania. Można wyróżnić trzy podstawowe rodzaje translatorów: interpretatory (interpretery), kompilatory i asemblery.

Interpretator

Interpretator (ang. interpreter) tłumaczy kolejne instrukcje kodu źródłowego i natychmiast je wykonuje.

Kompilator

Kompilator (ang. compiler) tłumaczy cały kod źródłowy programu. Powstały po kompilacji program wynikowy może być zapisany w pamięci masowej (np. na twardym dysku) wraz z innymi programami i wielokrotnie uruchamiany. Po modyfikacji kodu źródłowego proces kompilacji musi być powtórzony, aby zaktualizować program wynikowy. Programy kompilowane wykonywane są znacznie szybciej niż interpretowane.

Asembler

Asembler (ang. assembler) to rodzaj kompilatora, który tłumaczy kod źródłowy napisany w języku asemblera dla danej architektury systemu mikroprocesorowego.

Konsolidator

Konsolidator (program łączący, ang. linker) jest programem umożliwiającym utworzenie programu wykonywalnego z programów wynikowych otrzymanych w rezultacie niezależnej kompilacji wielu plików kodu źródłowego, także napisanych w różnych językach programowania.

Zazwyczaj konsolidator uruchamiany jest po bezbłędnej kompilacji, dołączając do programu skompilowaną treść funkcji bibliotecznych. Zaawansowane środowiska programowania pozwalają na wybranie opcji automatycznej kompilacji i konsolidacji ze swojego menu poleceń. Natomiast w przypadku uruchamiania kompilatora z wiersza poleceń systemu operacyjnego można taką opcję ustawić w plikach konfiguracyjnych kompilatora (często jest ona ustawiona domyślnie) lub wywołać konsolidator ręcznie. Można również utworzyć skrypt systemowy wywołujący kolejno kompilator i linker.

Maszyna wirtualna

Maszyna wirtualna (ang. virtual machine) to w ogólnym rozumieniu program służący do wykonywania innych programów. Maszyny wirtualne wykorzystując konkretny system komputerowy (np. komputer typu PC z systemem MS Windows), emulują pracę wirtualnego komputera, na którym wykonywane są programy.

Dużą zaletą tej metody jest możliwość tworzenia oprogramowania, które bez żadnych zmian może być uruchamiane na różnych platformach sprzętowych. Konieczne jest jedynie dostarczenie dla danej platformy maszyny wirtualnej zgodnej z oprogramowaniem. Wadą tej metody jest ograniczenie mocy obliczeniowej systemu wynikające z pracy samej maszyny wirtualnej.

Do maszyn wirtualnych zaliczamy np.: interpretery kodu bajtowego (ang. bytecode), kompilatory JIT oraz emulatory rzeczywistego sprzętu mikroprocesorowego.

JIT (Just In Time)

JIT jest metodą translacji polegającą na kompilacji do kodu maszynowego bezpośrednio przed wykonaniem danego fragmentu kodu programu, jest ona wykorzystywana m.in. przez maszyny wirtualne języka Java.

Kod źródłowy programu w tej metodzie kompilowany jest do postaci tzw. kodu bajtowego i w takiej formie jest rozprowadzany do użytkowników. Następnie po uruchomieniu na komputerze użytkownika maszyna wirtualna decyduje, czy dany fragment programu ma być kompilowany albo interpretowany. Zwykle kompilowane są fragmenty wielokrotnie wykonywane podczas pracy programu.

JIT pozwala na zwiększenie szybkości działania programu w porównaniu z klasyczną interpretacją.

3. Techniki programowania

Dobre oprogramowanie powinno spełniać postawione przed nim wymagania funkcjonalne, być efektywne i niezawodne.

Istotnym elementem tworzenia dobrych programów jest stosowanie odpowiednich technik programowania, które zmniejszają możliwość powstawania trudnych do wykrycia błędów oraz zapewniają lepszą czytelność kodu i łatwą jego ewolucję (rozbudowę programu, spełnienie nowych wymagań funkcjonalnych).

Wraz z rozwojem inżynierii oprogramowania powstało wiele **paradygmatów programowania**, takich jak:

- programowanie ekstremalne,
- programowanie funkcyjne,
- programowanie intencyjne,
- programowanie obiektowe,
- programowanie proceduralne,
- programowanie strukturalne,
- programowanie modularne;
- programowanie wieloparadygmatowe,
- programowanie zdarzeniowe,
- programowanie agentowe,
- programowanie logiczne,

Wśród nich najbardziej popularne to: programowanie proceduralne, programowanie strukturalne, programowanie modularne oraz programowanie obiektowe.

Programowanie proceduralne

Programowanie proceduralne polega na dzieleniu kodu na procedury (funkcje), czyli fragmenty wykonujące ściśle określone zadania.

Pisząc program realizujący dane zagadnienie, należ zamienić je na zbiór zadań, z których każde realizowane jest przez oddzielną procedurę. Na początku należy zdecydować, jakie procedury będą potrzebne. Następnie napisać je stosując możliwie najlepsze algorytmy i określić optymalny algorytm wywołań procedur.
Wykonanie takiego programu polega właśnie na określonym porządku wywołań różnych procedur, które zrealizują założone wymagania użytkownika.

Procedury w miarę możliwości nie powinny korzystać ze zmiennych globalnych, lecz pobierać i przekazywać wszystkie dane (lub wskaźniki do nich) jako parametry wywołania. Instrukcje skoku typu „goto” (skok do miejsca w programie oznaczonego etykietą) mogą być używane wewnątrz danej procedury, ale nigdy w celu wskoczenia wewnątrz innej procedury.

Programowanie strukturalne

Programowanie strukturalne stanowi rozszerzenie koncepcji programowania proceduralnego. Oznacza pisanie kodu programu bez użycia instrukcji skoku typu „goto”, zaleca wykorzystanie pętli i instrukcji warunkowej jako jedynych struktur sterujących.

Programowanie modularne

Programowanie modularne nazywa się także programowaniem z ukrywaniem danych, jest wynikiem ewolucji podejścia strukturalnego w kierunku poprawienia organizacji danych.

Zbiór powiązanych ze sobą procedur oraz danych, na których te procedury działają nazywamy modułem. Programista decyduje, jakie moduły będą mu potrzebne w programie.

Dane i procedury, które wymagane są jedynie do realizacji zadań wewnątrz modułu są ukrywane. Każdy moduł mus posiadać interfejs, czyli niezbędny zestaw dostępnych na zewnątrz danych i procedur, przez który można z niego korzystać.

Programowanie obiektowe

Programowanie obiektowe (ang. object-oriented programming) to metodologia tworzenia programów komputerowych, która definiuje programy za pomocą obiektów. Obecnie jest to najsilniej rozwijająca się koncepcja w programowaniu.

Podstawowe pojęcia w programowaniu obiektowym to: klasa i obiekt. Klasa jest typem danych zdefiniowanym przez programistę, który może zawierać różnego rodzaju dane nazywane składowymi klasy. Składowymi klasy oprócz typowych danych (liczby, tablice, łańcuchy tekstowe,...) mogą być funkcje (procedury), które nazywane są w terminologii obiektowej metodami.

Obiektem jest utworzony egzemplarz klasy (zmienna obiektowa), z którym można się komunikować przez wywoływanie dostępnych z zewnątrz metod zawartych w obiekcie. Pod wpływem wywoływanych metod obiekt zmienia swój stan.

Programowanie obiektowe ułatwia pisanie, konserwację i wielokrotne użycie kodu źródłowego. Języki programowania zorientowane obiektowo wykorzystują zwykle takie techniki jak: dziedziczenie i polimorfizm.

Język Ada

Ada to strukturalny, kompilowany język programowania utworzony w latach siedemdziesiątych XX wieku. Od tego okresu powstały dwa standardy języka: Ada83 i Ada95. Nowsza wersja (Ada95) wprowadza między innymi wsparcie dla programowania obiektowego. Programy napisane w Adzie można kompilować na przykład przy pomocy kompilatora GNAT. Poniżej umieszczono przykład kodu źródłowego w języku Ada.

```
with Ada.Text_IO, Ada.Integer_Text_IO;
use Ada.Text_IO, Ada.Integer_Text_IO;
procedure OneInt is
   Index : INTEGER;        -- A simple Integer type
begin
   Index := 23;
   Put("The value of Index is");
   Put(Index);             -- The default field width is 11 columns
   New_Line;
   Index := Index + 12;
   Put("The value of Index is");
   Put(Index, 8);
   New_Line;
end OneInt;
```

Język C

C jest językiem programowania stworzonym na początku lat siedemdziesiątych przez Dennisa Ritchie. Głównym zastosowaniem języka było programowanie systemów operacyjnych i tak w 1973 roku w języku C udało się zaimplementować jądro systemu operacyjnego Unix.

Język C po roku 1980 stał się jednym z najbardziej popularnych języków programowania, nie tylko w dziedzinie systemów operacyjnych, ale także programów użytkowych. Praktycznie większość obecnie wykorzystywanych komputerów o różnych platformach sprzętowych posiada kompilatory pozwalające wykorzystywać język C.

Standard języka C został zapisany w 1989 roku normie ANSI (ANSI X3.159-1989 "Programming Language C") oraz rok później w normie ISO (ISO 9899:1990). Język C jest ciągle rozwijany. W 1999 roku opublikowano normę ISO 9899:1999.

Na bazie języka C powstał język C++, który wprowadza możliwość programowania obiektowego.

Poniżej umieszczono przykład prostego programu w języku C.

Program kula.c

Program monituje o wpisanie wartości promienia kuli, po czym oblicza jej objętość i wypisuje wynik na ekranie.

```
/* Program obliczający objętość kuli */
#include <stdio.h>

int main(void)
{
  float pi = 3.1415926;
  float promien, objetosc;
  printf(" Podaj promień kuli: ");
  scanf("%f", &promien);
  objetosc = (4/3)*pi*promien*promien*promien;
  printf("\n Objętosc kuli wynosi: %f \n", objetosc);
  return 0;
}
```

Język C++

Język C++ jest obiektowo zorientowanym językiem programowania, pozwalającym także na pisanie programów proceduralnych. Został stworzony przez Bjarne Stroustrupa na podstawie języka C w latach osiemdziesiątych XX wieku. Standard języka C++ opublikowany został w 1998 roku (ISO/IEC 14882-1998 "Information Technology - Programming Languages - C++").

Język C++ jest ciągle udoskonalany, przy czym zachowuje bardzo dobrą zgodność ze starszym kodem źródłowym. Kompilatory C++ zwykle bez problemu kompilują kod źródłowy w języku C, często można nawet wymusić tryb zgodności z tym językiem.

C++ dzięki rozbudowanym możliwością programowania obiektowego, bogatym bibliotekom funkcji standardowych oraz wysokiej efektywności programów wynikowych jest obecnie jednym z najpopularniejszych języków wysokiego poziomu.

Poniżej umieszczono prosty przykład programu w języku C++.

Program kula.cpp

Program monituje o wpisanie wartości promienia kuli, po czym oblicza jej objętość i wypisuje wynik na ekranie. Napisano go z wykorzystaniem techniki obiektowej. Zdefiniowano klasę kula, w której znajduje się metoda obliczająca objętość kuli. Metoda ta została następnie wywołana na utworzonym w trakcie działania programu obiekcie kula1 (kula1.objetosc()).

```
#include <iostream>
using namespace std;

class kula  // klasa reprezentująca kulę
    {
    private:
      float r; //promień kuli
    public:
      kula(float promien) { r = promien; }
      void Ustaw_promien(float promien) { r = promien; }
      float promien(void) { return r; }
      float objetosc(void) { return (4/3)*3.1415926*r*r*r; }
    };
```

```
int main(int argc, char *argv[])
{
  float promien;
  cout << " Podaj promień kuli: ";
  cin >> promien;

  kula kula1(promien);  //utworzenie obiektu kula1 z klasy kula

  cout << "\n Objętosc kuli wynosi: " << kula1.objetosc() << "\n";
  return 0;
}
```

Popularne środowiska programistyczne języka C++

- Borland C++ Builder –Windows
- Dev-C++ -Windows
- Microsoft Visual C++ -Windows

Język Fortran

Fortran (FORmula TRANslator) powstał w latach 50 i nadal jest używany. Głównym zastosowaniem tego języka są obliczenia naukowe i analiza numeryczna. Dużą zaletą Fortranu jest szybkość obliczeń oraz wysoka jakość kodu wynikowego generowanego przez jego kompilatory. Istnieje duża ilość bibliotek napisanych w Fortranie pozwalających rozwiązać większość zadań numerycznych.

Wraz z rozwojem języka powstało kilka wersji numerowanych według roku standaryzacji: Fortran 66, Fortran 77, Fortran 90 i Fortran 95. Od wersji 90 w języku dostosowano składnię do współczesnych języków oprogramowania, jednak z uwagi na przyzwyczajenia starszych programistów nadal popularny jest standard 77. Fortran pozwala pisać programy proceduralne z możliwością stosowania struktur w nowszych wersjach języka (Fortran 90/95). Najnowsze kompilatory Fortranu mają możliwość wizualizacji graficznej obliczeń.

Wybrane kompilatory i zintegrowane środowiska języka Fortran 90/95

- G95 Project - Linux
- Intel Fortran Compiler - Linux, Windows
- G95 Project - Linux
- Absoft Pro Fortran - Linux, Windows, Mac OS, PowerPC
- Compaq Visual Fortran - Windows, Linux/Unix/OpenVMS Alpha, OpenVMS VAX
- Lahey/Fujitsu Fortran - Linux, SPARC Solaris
- NAGWare f95 - Linux, Mac OS, SPARC Solaris, IRIX, SunOS, OSF/1, PA-RISC
- NA Software FortranPlus - Linux, Windows
- Portland Group PGHPF Workstation - Linux, Windows
- Salford FTN95 - Windows

Język Pascal i Object Pascal

Pascal to jeden z popularniejszych języków programowania o szerokich zastosowaniach. Został opracowany przez Niklausa Wirtha w 1971 roku, jako język do nauki programowania strukturalnego.

Obecnie Pascal został rozbudowany o mechanizmy wspierające programowanie obiektowe, w wyniku czego powstał Object Pascal.

Jedne z lepszych środowisk programistycznych języka Object Pascal stworzyła firma Borland, należy do nich: Turbo Pascal (MS-DOS), Delphi (Windows) i Kylix (Linux, Solaris). Na uwagę też zasługuje należący do wolnego oprogramowania Free Pascal.

Poniżej przedstawiono prosty program w języku Pascal.

Program kolo.pas
Program oblicza pole koła o promieniu r wpisanym przez użytkownika.

```
program Pole_kola;
{ Program oblicza pole koła o promieniu wprowadzonym z klawiatury }

var          { deklaracja zmiennych }
  r : real;  { promień koła – zmienna rzeczywista }

begin
  writeln('Program oblicza pole koła');
  write('Podaj promień koła: ');
  readln (r);     { odczytaj zmienną z klawiatury }
  write('Pole = ', Pi*r*r); { oblicz i wypisz pole koła }
  readln;
end.
```

Język Java

Java jest współbieżnym, zorientowanym obiektowo, językiem programowania stworzonym przez firmę Sun Microsystems. Do najważniejszych cech języka Java należy niezależność od systemu operacyjnego i platformy sprzętowej, dzięki stosowaniu maszyny wirtualnej.

Kod źródłowy jest kompilowany do postaci pośredniej zwanej kodem bajtowym (ang. bytecode), który jest wykonywany (interpretowany) przez maszynę wirtualną, zaimplementowaną w konkretnym systemie opracyjnym. Obecnie większość systemów operacyjnych posiada dedykowane im maszyny wirtualne języka Java. Ponieważ jednak taki system jest wolniejszy niż tradycyjna kompilacja, maszynę wirtualną często uzupełnia się o technikę JIT (Just In Time).

Programy w Javie mogą być uruchamiane także przez przeglądarki internetowe w postaci umieszczonych na stronach WWW apletów Javy. Java jest często mylona ze skryptowym językiem JavaScript, z którym ma niewiele wspólnego.

Podstawowe narzędzia do tworzenia programów (Java SDK, NetBeans IDE), maszyny wirtualne oraz dokumentację, można znaleźć w oficjalnym serwisie języka Java (http://java.sun.com).

Język Smalltalk

Smalltalk jest czysto obiektowym językiem programowania, w którym każda wartość, każda struktura danych jest obiektem i nie istnieje sztuczne oddzielenie wartości i obiektów. Smalltalk jest językiem reflektywnym, co oznacza, że język Smalltalk jest zaimplementowany w Smalltalku. Obiekty, które ten język definiują, same są opisane w tym języku. Smalltalk jest także biblioteką klas i zintegrowanym środowiskiem programistycznym w jednym.

Prostota, możliwość szybkich dynamicznych zmian, elastyczna natura języka powoduje, że Smalltalk jest doskonałym narzędziem do szybkiego tworzenia złożonych aplikacji o krótkim cyklu rozwoju, w których zachodzi konieczność dopasowywania się do nowych wymagań. Dla języka Smalltalk powstała i została rozwinięta koncepcja programowania ekstremalnego (XP -ang. eXtreme Programming).

Kod źródłowy programu w Smalltalku kompilowany jest do postaci pośredniej i uruchamiany jest na maszynie wirtualnej, dzięki czemu aplikacje mogą być stosunkowo łatwo przenoszone na różne platformy sprzętowe.

Popularne środowiska programistyczne języka Smalltalk

- Squeak –Windows, MacOS
- VisualAge for Smalltalk - Windows, OS/390, Solaris, Unix, OS/2, Linux
- Visual Works -Windows, Linux, Unix, MacOS
- ObjectStudio -Windows
- Dolphin Smalltalk -Windows

5. Proces tworzenia programu na przykładzie języka C

Język C jest popularnym językiem programowania o szerokich zastosowaniach, pozwalającym stosować techniki programowania proceduralnego, strukturalnego oraz modularnego. Nie jest skomplikowany, a przy tym bardzo elastyczny.

Światowe standardy C (ISO i ANSI) są powszechnie stosowane przez twórców kompilatorów, toteż kod źródłowy programów w C może być łatwo przenoszony między różnymi komputerami. Programy napisane w C można kompilować także przy pomocy kompilatorów języka C++. Wynika to z faktu, że C++ powstał na bazie C i zachował z nim zgodność (poza małymi wyjątkami).

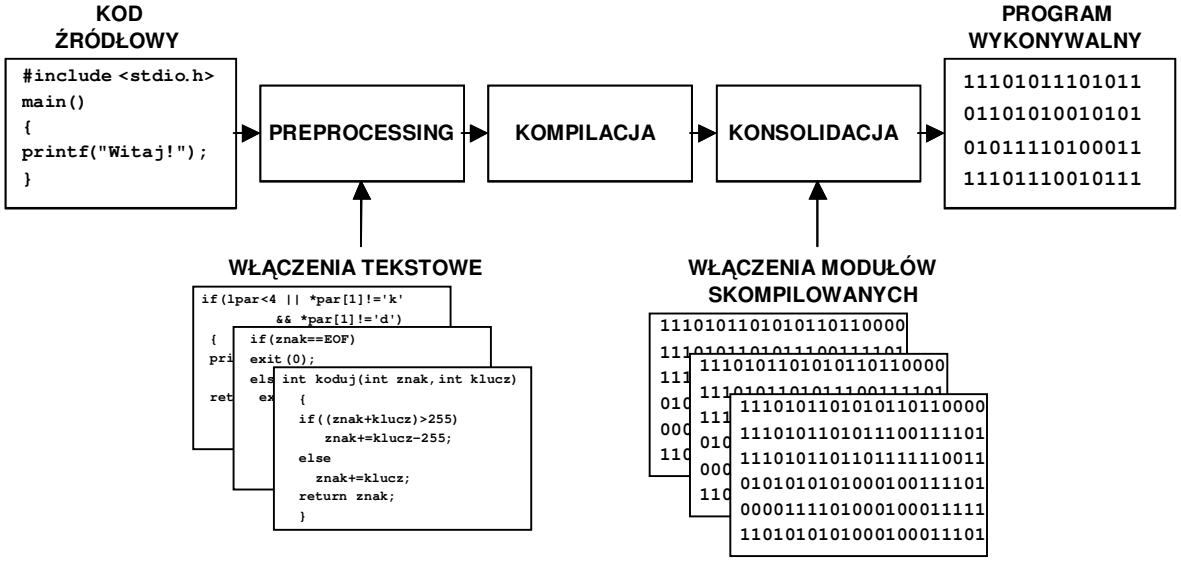
Większość kompilatorów C++ posiada możliwość przełączenia w tryb standardowego C.

Proces tworzenia programu

Proces tworzenia programu w języku C można podzielić na następujące etapy:

1. Napisanie kodu źródłowego w dowolnym edytorze tekstu i zapisanie go w postaci niesformatowanego pliku tekstowego o nazwie zakończonej „**.c**”, na przykład **pierwszy.c**.

2. Kompilacja i konsolidacja pliku źródłowego przy pomocy narzędzi programistycznych języka C, dostępnych w systemie.



Rysunek Proces tworzenia programu.

W fazie wstępnej kompilacji kodu źródłowego (ang. preprocessing), pracuje preprocesor języka wykonując: makrorozwinięcia, włączenia do programu innych plik źródłowych (tekstowych) oraz kompilację warunkową. Preprocesor interpretuje umieszczone w kodzie źródłowym dyrektywy poprzedzone znakiem #.

Po zakończeniu pracy preprocesora uruchamiana jest właściwa kompilacja, której efektem jest powstanie programu w kodzie pośrednim (ang. object code), zwykle w pliku o nazwie kończącej się „**.obj**” lub „**.o**”.

Następnie w fazie konsolidacji do programu skompilowanego dołączane są potrzebne moduły (skompilowane wcześniej), dołączane w ten sposób są na przykład moduły bibliotek standardowych. Wynikiem konsolidacji jest program wykonywalny, który można uruchomić z poziomu systemu operacyjnego komputera.

Podczas procesu tworzenia programu wykonywalnego mogą wystąpić różnego rodzaju błędy (ang. errors). W czasie kompilacji mogą wystąpić błędy syntaktyczne, związane z nieprzestrzeganiem reguł języka (ang. compiler errors). Natomiast przy konsolidacji błędy (ang. linker errors) mogą być związane z niekompletnością dołączanych modułów lub wręcz ich brakiem. Błędy mogą także objawić się dopiero po uruchomieniu programu, tzw. błędy wykonania (ang. runtime errors). Powstają one w wyniku niewłaściwej implementacji algorytmów lub problemów z dostępem do zasobów komputera. Do błędów tego typu należy na przykład: próba dzielenia przez zero, brak pamięci dla zmiennych dynamicznych, niedostępność dysku, itp.

Podczas kompilacji i konsolidacji informacje o błędach wyświetlane są na ekranie lub zapisywane są do plików tekstowych. Powstaje lista zawierająca dla każdego błędu jego krótki opis oraz orientacyjną lokalizację w kodzie źródłowym (numer wiersza).

Oprócz informacji o błędach kompilator może wypisywać ostrzeżenia (ang. warnings), wskazując miejsca w kodzie programu, które mogą być przyczyną błędów wykonania. Przykładowo, można spotkać następujące ostrzeżenia:

- użyto zmiennej bez przypisanej początkowej wartości,
- przypisana wartość zmiennej nigdy nie jest wykorzystywana w programie,
- użyto nie zalecanej (przestarzałej) funkcji bibliotecznej,
- dzielenie przez zero.