**Język programowania** – zbiór zasad określających, kiedy ciąg symboli tworzy [program komputerowy](https://pl.wikipedia.org/wiki/Program_komputerowy) oraz jakie obliczenia opisuje.( *Mordechai Ben-Ari: Understanding Programming Languages. Chichester: John Wiley & Sons, 1996.*)

**[jezyk wysoko i niskopoziomowe]**

[rodzaje jeżyków i ich zastosowania]

Istnieje wiele języków programowania i każdy z nich potrzebuje narzędzia, które przetłumaczy je na polecenia zrozumiałe dla komputera.   
Jednym z takich narzędzi jest kompilator.  
kompilator jest programem, który potrafi przeczytać program sformułowany w jednym języku – języku źródłowym – i przełożyć go na równoważny program w innym języku – języku wynikowym. Ważną rolą kompilatora jest zgłaszanie wykrytych w czasie tłumaczenia dowolnych błędów w programie źródłowym.(książka)   
{rysunek 1.1}

Program wynikowy jest programem wykonywalnym w języku maszynowym, może zostać uruchomiony przez użytkownika w celu przetważania wejścia i wygenerowania wyjścia(książka) { rysunek 1.2}

Innym popularnym narzędziem jest interpreter.

Zamiast tworzenia programu wynikowego jako efektu tłumaczenia na bieżąco wykonuje polecenia przetłumaczone z kodu źródłowego.  
{rysunek 1.3}

Program tworzony przez kompilator jest znacznie szybszy. Z drugiej strony interpretacja może zająć mniej czasu niż kompilacja i uruchomienie oraz udostępnia lepszą diagnostykę błędów. Dlatego interpretacja jest często wykorzystywana w językach skryptowych

Kompilator tłumaczący język wysokiego poziomu na inny język wysokiego poziomu jest nazywany translatorem source-to-source lub transkompilatorem.

Coraz częściej wykorzystywana jest metoda hybrydowa nazywana kompilatorami just-in-tim(JIT, kompilacja na żądanie), gdzie kod źródłowy jest kompilowany do kodu pośredniego zwanym kodem bajtowym(bytecode),który jest interpretowany przez maszynę wirtualną   
{rysunek 1.4}

Do utworzenia programu wykonywalnego może być potrzebne kilka programów   
{rysunek 1.5}

Preprocesor łączy kod żródłowy z kilku plików oraz rozwija skróty nazwane makrami do pełnych wyrażeń języka źródłowego

Kompilator często jako wynik swojej pracy dostarcza program w języku asembler, gdyż jest on łatwiejszy do wykonania i debugowania

Asembler generuje relokowany kod maszynowy jako swoje wyjście(książka)

Linker zwany również konsolidatorem łączy nasz kod maszynowy z kodem maszynowym bibliotek wymaganych do działania programu

Kompilacja odbywa się w 2 częściach: analizy(front-end) i syntezy(back-end)

Część analityczna dzieli kod źródłowy na części składowe stosując strukturę gramatyczną. W tej części szukane są błędy składniowe lub niejednoznaczności semantyczne. Na tym etapie zbierane są również informacje o kodzie źródłowym. Część syntezy na podstawie danych zebranych w fazie analizy tworzy program wynikowy. W każdej z części możemy wyszczególnić kilka faz, w których przekształcana jest jedna reprezentacja kodu źródłowego w kolejną.

{rysunek 1.6}

W praktyce wiele faz może być grupowanych razem i reprezentacje pośrednie między zgrupowanymi fazami nie muszą być jawnie konstruowane (ksiazka)

Analizator leksykalny (lekser) odczytuje strumień znaków budujących program źródłowy i grupuje te znaki w znaczące sekwencje nazywane leksemami. Dla każdego leksemu analizator leksykalny tworzy wyjście w postaci tokenu.(książka)

Analizator składniowy(parser) używa tokenów utworzonych przez analizator leksykalny do zbudowania pośredniej reprezentacji przypominającej drzewo, odwzorowującej gramatyczną strukturę strumienia tokenów.(ksiazka)

Analiza semantyczna sprawdza kod źródłowy pod kątem spójności semantycznej programu z definicją języka. Ponadto gromadzi on informacje o typach i zapisuje je.(książka)

Generowanie kodu pośredniego – tworzone jest jawna niskopoziomowa reprezentacja pośrednia zbliżona do kodu maszynowego. Kod ten powinien być łatwy do utworzenia i przetłumaczenia na kod maszynowy (?książka)

Optymalizacja kodu – Faza niezależnej od architektury maszynowej optymalizacji kodu ma na celu ulepszenie kodu pośredniego dzięki czemu lepszy będzie również kod wynikowy (książka)

Generowanie kodu – instrukcje pośrednie są tłumaczone na sekwencje instrukcji maszynowych wykonujących to samo zadanie. Krytycznym aspektem generowania kodu jest rozważne przypisanie rejestrów do przechowywanych zmiennych.

{rysunek 1.7}