Podstawy Programowania

dr inż. Tomasz Marciniak

Organizacja wykładu

- Wykłady – egzamin,
- Forma pisemna,
- Terminy:
  - „0” – ostatni zjazd 17 stycznia 2017,
  - „1” – 07 luty 2017 godz.9:00 s.B1.AN
  - „2” – 15 luty 2017 godz.9:00 s.B1.AN

Organizacja wykładu

- Składnia języka C, zmienne, operatory,
- Słowa kluczowe, instrukcje,
- Funkcje/procedury,
- Struktury, wskaźniki, tablice,
- Operacje na plikach,
- Wzorce w C++, STL.

Literatura

- B. Kernighan, D. Ritchie, Jezyk ANSI C, WNT, (1987), 1994, 2007
- Stroustrup Bjarne: *Język C++, WNT 2002*
- B. W. Kernighan, R. Pike, Lekcja programowania, WNT, 2002
- T. Cormen, C. Leiserson, R. Rivest, Wprowadzenie do algorytmów, WNT 1998, 2007
- N. Wirth, Algorytmy + struktury danych = programy, WNT, 1980, 2004

Po co pisać programy?

- Komputer bez oprogramowania to złom.

Co sprzyja rozwojowi oprogramowania:

- Różne systemy operacyjne- różne potrzeby.
- Duży wybór programów narzędziowych.
- Różne typy danych –różne wymagania.
- Własne potrzeby:
  - Np. baza danych z odpowiednim interfejsem
  - Obsługa nietypowych zdarzeń pochodzących z urządzenia zewnętrznego (np. RS-232)

Zapamiętaj !

- Najlepiej działają programy niepotrzebne
  - Związane jest to z tym, że nikt ich nie testuje, a programista wie gdzie i jakie dane można wprowadzić. Świadomie nie będzie wprowadzał błędnych danych, chyba, że właśnie sprawdza zabezpieczenie programu ☺.
- Żeby nauczyć się programować, trzeba pisać programy ☺

Przykładowy cykl programowania

1. Opracowanie projektu: specyfikacja wymagań, danych wejściowych i wyjściowych;
2. Opracowanie algorytmów: schemat blokowy, opis słowny;
3. Napisanie programu → implementacja algorytmów;
4. Testowanie programu i usuwanie błędów;
5. Przekazanie użytkownikowi;
6. Pielęgnacja programu.

Cykl programowania

- Kroki powinno się wykonywać po kolei;
- Nie zawsze się to udaje:
  - w trakcie opracowywania algorytmów okazuje się, że źle określono np. dane wejściowe → należy wrócić do kroku 1
  - w trakcie pisania programu okazuje się, że nie można zrealizować algorytmu (lub wykonuje się zbyt długo) → powrót do kroku 2;
  - testy programu wykazały błędne dane wyjściowe → powrót do kroku 3.

Cykl programowania

- o pielęgnacji programu należy myśleć na początku tego procesu – niechlujnie napisanego programu nie da się pielęgnować;
- rozpoczynanie programowania od kroku 3 jest bardzo, ale to bardzo złym pomysłem.

Algorytm

- **Algorytm**, dokładny przepis podający sposób rozwiązania określonego zadania w skończonej liczbie kroków; zbiór poleceń odnoszących się do pewnych obiektów, ze wskazaniem porządku, w jakim mają być realizowane. Nabrał znaczenia z rozwojem informatyki, gdzie opisuje logiczny ciąg operacji, które ma wykonać program.
- Algorytm zapisany przy pomocy języka programowania jest programem.

Program to nie algorytm

Algorytm cd.

- Wyróżnia się algorytmy **numeryczne**, operujące na liczbach (np. algorytm Euklidesa), i **nienumeryczne**, operujące na obiektach nieliczbowych (np. sortowanie dokumentów). Istnieje również podział algorytmów na **sekwencyjne** (kolejność czynności jest określona jednoznacznie) i **niesekwencyjne** (równoległe, współbieżne - następstwo między pewnymi operacjami nie jest określone).
- Algorytmy charakteryzują się możliwością **zapisywania** ich w różnych językach i przez skończoną liczbę symboli, bez odwoływania się do analogii, a także faktyczną wykonalnością i możliwością wielokrotnej realizacji.
- Termin **algorytm** wywodzi się od zlatynizowanej formy (Algorismus, Algorithmus) nazwiska matematyka arabskiego z IX w.,

  Al-Chuwarizmiego.

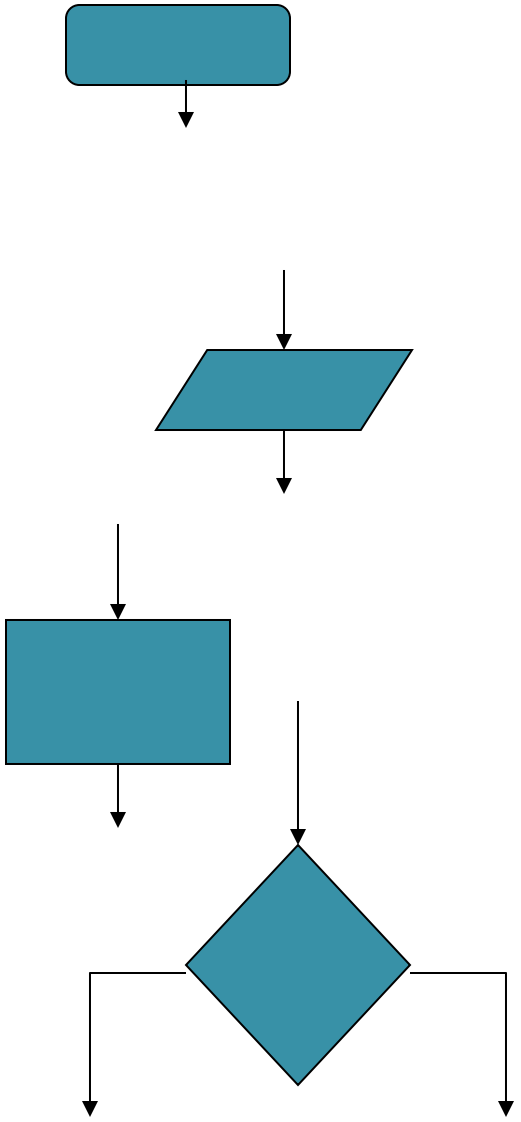
Schemat blokowy

a) Blok graniczny - oznacza on początek, koniec, przerwanie lub wstrzymanie wykonywania działania, np. blok startu czy końca programu.

b) Blok wejścia-wyjścia - przedstawia czynność wprowadzania danych do programu i przyporządkowania ich zmiennym dla późniejszego wykorzystania, jak i wyprowadzenia wyników obliczeń, np. czytaj a, pisz b+10.

c) Blok obliczeniowy - oznacza wykonanie operacji, w efekcie której zmienią się wartości, postać lub miejsce zapisu danych, np. a= a + 1.

d) Blok decyzyjny - przedstawia wybór jednego z dwóch wariantów wykonywania programu na podstawie sprawdzenia warunku wpisanego w ów blok, np. a = b.



Wybór odpowiedniego języka

- Przy wyjeździe za granicę w każdym kraju używany jest język ojczysty. Niektóre języki są do siebie zbliżone jednak różnią się między sobą. Tak samo jest z językami programowania. Pomimo pewnej zbliżonej semantyki, sposób wykonywania poleceń i ich składnia różnią się między różnymi językami.
- Do różnych zadań wybiera się różne języki programowania.

Języki programowania niskiego i wysokiego poziomu

- Nie chodzi oczywiście o wzrost programisty ale sposób porozumiewania się z komputerem. Programista musi pisać w języku zrozumiałym przez komputer.

- W językach *niskiego poziomu* jedna instrukcja napisana przez programistę najczęściej odpowiada jednemu rozkazowi wykonywanemu przez procesor.
- Języki *wysokiego poziomu* są raczej zbliżone do języka ludzkiego (angielskiego) i jedna instrukcja może odpowiadać kilkunastu rozkazom procesora.

Pascal

- Pascal był jednym z najpopularniejszych języków programowania w latach 80. Twórcą jest (1971) Niklaus Wirth. W zamierzeniu autora Pascal miał służyć przede wszystkim do nauki programowania strukturalnego, czyli określania dużych elementów poprzez elementy niższego rzędu, aż do końcowych elementów programowych realizujących określone funkcje. Wraz z rozwojem techniki mikrokomputerów pojawiły się liczne wersje języka Pascal.

ANSI C/C++

- Istnieje wielu producentów oprogramowania jednak standard jest jeden – jest nim ANSI (*American National Standards Institute*) C. Wraz z rozwojem języka, pojawiły się nowe standardy:
  - Standard ANSI C++ powstał w 1997 roku;
  - Standard ISO (*International Organization for Standardization*) C++ powstał 1998 roku;
- Twórcą języka C++ jest Bjarne Stroustrup.
- Nazwa C++ została użyta pierwszy raz w grudniu 1983 roku, a zasugerował ją Rick Mascitti.

Pascal czy C, a może Java

- Wybór między tymi konkretnymi językami programowania powinien należeć raczej do samego programisty. Pierwszy z nich jest dobry do nauki podstaw programowania, drugi, zwłaszcza c++, jest przeznaczony dla osób dobrze zorientowanych i przygotowanych do pisania programów.

Język programowania

- Definicja: **jest to zbiór zasad składni, instrukcji, dzięki którym powstaje kod źródłowy programu.** Procesor jest w stanie wykonywać program w kodzie maszynowym. Jednakże tworzenie programów w tym języku jest praktycznie niemożliwe. Dlatego programista używa języka zrozumiałego dla człowieka, który następnie jest kompilowany bądź interpretowany do postaci maszynowej.

Kod źródłowy

- **Kod źródłowy**, program komputerowy napisany w **języku programowania**. Jest to postać programu, która jest zrozumiała dla programisty (bez konieczności jego uruchamiania). Kod źródłowy jest przekształcany na kod maszynowy w procesie translacji programu.

Kod maszynowy

- **Kod maszynowy**, język rozumiany przez procesor. Program w kodzie maszynowym składa się z ciągu wartości binarnych, które oznaczają zarówno instrukcje jak i dane. Program, który jest napisany w pewnym języku programowania, musi zostać skompilowany, aby mógł być wykonywany przez komputer. **Postać kodu maszynowego zależy od architektury procesora**, na który dany program jest przeznaczony. Dlatego program musi zostać skompilowany na konkretnej **maszynie**, ewentualnie na systemie kompatybilnym z **systemem docelowym**.

Rodzaje języków programowania

Istnieje wiele rodzajów języków programowania.

- strukturalne i obiektowe,
- kryterium ze względu na zastosowanie języków (innych używa się do tworzenia programów multimedialnych, a innych do obliczeń numerycznych czy np. aplikacji sieciowych).

Niektóre z języków są bardziej uniwersalne niż inne. Do najpopularniejszych obecnie języków programowania należą C/C++, Java, czy Delphi.

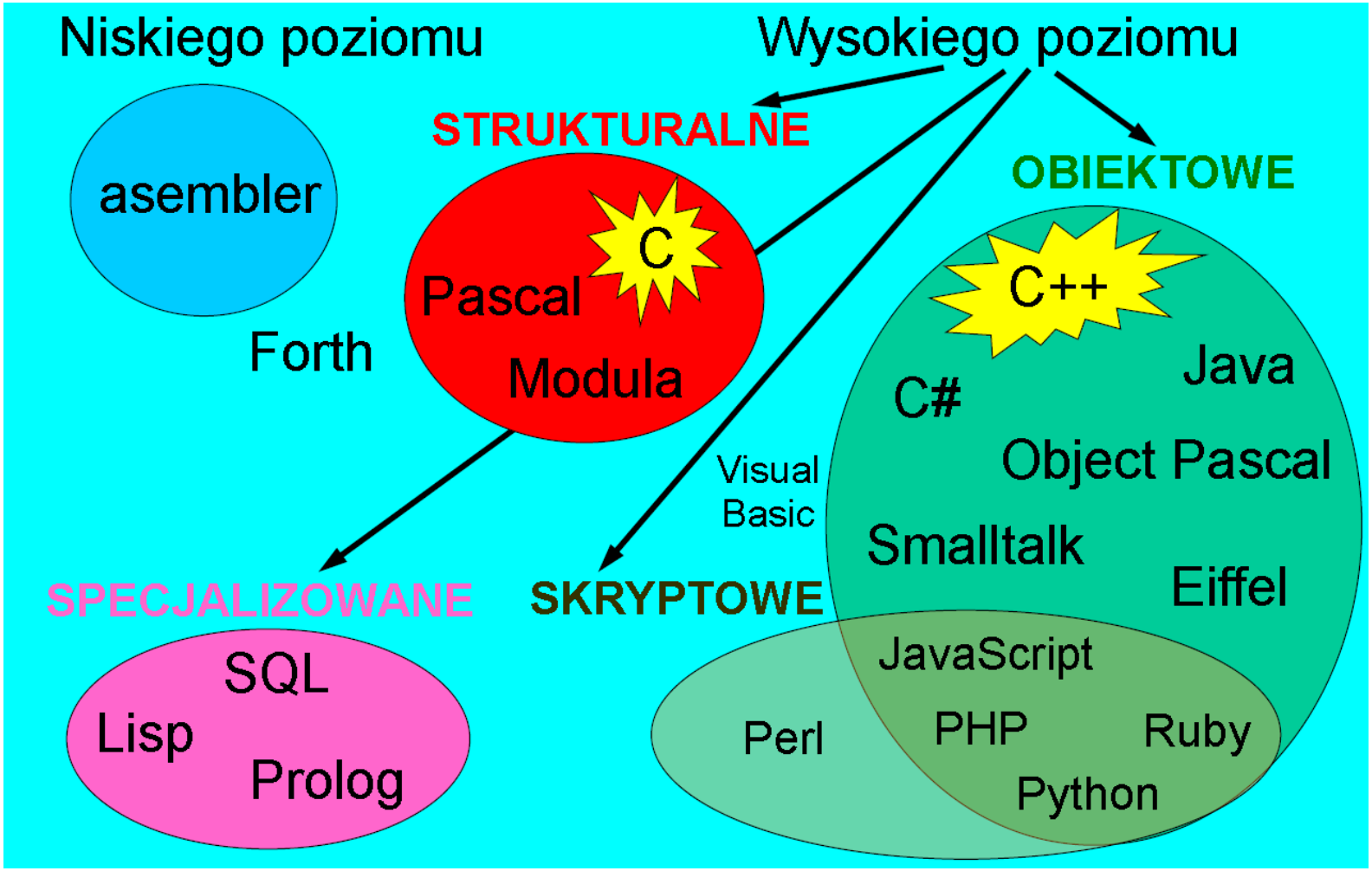
Programowanie strukturalne

- **Programowanie strukturalne,** rodzaj programowania w którym program podzielony jest na niewielkie moduły - procedury, bądź funkcje. Programowanie strukturalne ułatwia projektowanie, testowanie a także utrzymanie kodu programu. Do najpopularniejszych języków strukturalnych zalicza się Pascal, C, Modula-2.

Programowanie obiektowe

- **Programowanie obiektowe**, rodzaj programowania, w którym dane i wykonywane na nich operacje są połączone. Ten formalny zabieg umożliwia szybsze pisanie większych programów, przez "składanie ich" ze wzajemnie powiązanych obiektów, które odpowiadają za daną funkcję programu (np. przygotowanie danych, wykonanie obliczeń, zaprezentowanie wyników).
- Do najpopularniejszych języków programowania obiektowego należą C++, Java, Effel.

Języki programowania



Język obiektowy

- **Język obiektowy** (angielskie *object-oriented language*), język który umożliwia realizowanie paradygmatu obiektowego, tzn. programowanie z użyciem **hierarchii klas** (zwłaszcza klas abstrakcyjnych). Popularnymi językami obiektowymi są: C++, Java, Smalltalk, obiektowy Pascal, Beta, Theta, CLOS, Eiffel, Ada98, Python i in.
- W języku angielskim rozróżnia się też słabsze pojęcie: ***object-based language***, tj. język umożliwiający stosowanie obiektów, w którym użytkownik może definiować i stosować własne typy danych. Od lat osiemdziesiątych języki obiektowe uważa się za szczytowe osiągnięcie inżynierii oprogramowania.

Języki wysokiego poziomu

- **Język wysokiego poziomu** (ang. *high-level language*), język programowania, zazwyczaj o budowie blokowej, spełniający wymagania programowania strukturalnego, programowania z użyciem obiektów lub nawet programowania obiektowego.
- Typowymi i najczęściej używanymi językami wysokiego poziomu są języki: C i C++, Smalltalk, Java, Pascal i Lisp, lecz również języki specjalizowane, np. język SQL służący do formułowania zapytań do baz danych lub edukacyjny język Logo.
- Istnieje wiele modyfikacji języków starszej generacji, np. Fortranu, Basica lub Cobolu, które po unowocześnieniu spełniają w zupełności kryteria języków wysokiego poziomu. Cechą znamienną dla nich jest możliwość bezpieczniejszego programowania, tj. programowania mniej podatnego na błędy, i wynikająca z tego **możliwość opracowywania większych programów** (powyżej 10 000 wierszy kodu).

Typowe narzędzia programisty

- Translatory/interpretery
- Kompilatory
- Linkery

Interpreter, interpretator (ang. *interpreter*)

- Interpreter analizuje instrukcje programu źródłowego a przeanalizowane fragmenty (zazwyczaj jedna instrukcja) są wykonywane. Wykonanie powtórnie tego samego fragmentu wymaga powtórnej analizy.
- Ponieważ interpreter nie tworzy przekładu w kodzie maszynowym, lecz wykonuje instrukcje, tłumacząc je na bieżąco za każdym razem, wykonanie programu znacznie się wydłuża.
- Interpreter nie zmusza z kolei do oczekiwania na wykonanie kompilacji po każdej zmianie programu. O błędach programista dowiaduje się dopiero w chwili wykonywania błędnej instrukcji. Typowymi interpreterami są systemy programowania Basic, Java, Perl;

Kompilator

- **Kompilator**, to program tłumaczący program w języku wysokiego poziomu, który tworzy programy wynikowe, uruchamialne dopiero po zakończeniu tłumaczenia (w odróżnieniu od interpretatora).
- Niektóre kompilatory tłumaczą kod źródłowy na kod język asemblera a dopiero później odbywa się tłumaczenie przez asembler na kod wynikowy.

Kompilacja

- **Kompilacja** (ang. *compiling*), nie całkiem odpowiedni, lecz powszechnie przyjęty termin określający proces tłumaczenia programu z postaci źródłowej na kod wynikowy. Obejmując fazy **analizy leksykalnej, składniowej i generowania kodu**, kompilacja **nie musi przekształcać** programu źródłowego w wykonywalny kod maszynowy komputera. Produktem kompilacji jest najczęściej **kod z adresami względnymi**, obliczonymi z osobna dla każdego skompilowanego modułu. Podczas **łączenia modułów** (które bardziej zasługiwałoby na miano "kompilacji", lecz nie jest to przyjęte) następuje ostateczne przeliczenie adresów względem wspólnej bazy i doprowadzenie przetłumaczonego programu do postaci ładowalnej.

Program binarny

- **Program binarny** (ang. *binary program*), program **przetłumaczony na język maszynowy**, gotowy do ładowania i wykonania.

Deasembler

- **Deasembler, dezasembler** (ang. *disassembler*), program narzędziowy o działaniu odwrotnym do asemblera, przetwarzający dwójkową reprezentację programu w kodzie maszynowym na rozkazy języka asemblera. Deasembler zazwyczaj wchodzi w skład pakietu uruchomieniowego.
- **Deasemblacja,** w informatyce, utworzenie postaci kodu programu zapisanej w języku asemblera, z wersji binarnej (wykonywalnej) tego programu. Deasemblacja jest jedną z funkcji często oferowanych przez programy uruchomieniowe.

Dekompilacja

- **Dekompilacja** (ang. *decompilation, deassembling*), odtworzenie postaci źródłowej programu na podstawie jego kodu wynikowego. Dekompilacja nie odtworzy, co oczywiste, komentarzy i układu tekstu źródłowego programu. Stosowanie dekompilacji może rodzić problemy prawne.

Program wynikowy

1. **Program wynikowy** (angielskie *object program, object module, object code*),
   - **efekt pracy kompilatora, półprodukt** tłumaczenia w postaci nadającej się do dalszego tłumaczenia przez asembler lub do konsolidacji. W systemie MS-DOS pliki programu wynikowego oznacza się konwencjonalnie godłem **obj**;
   - program przetłumaczony, gotowy do wykonania.

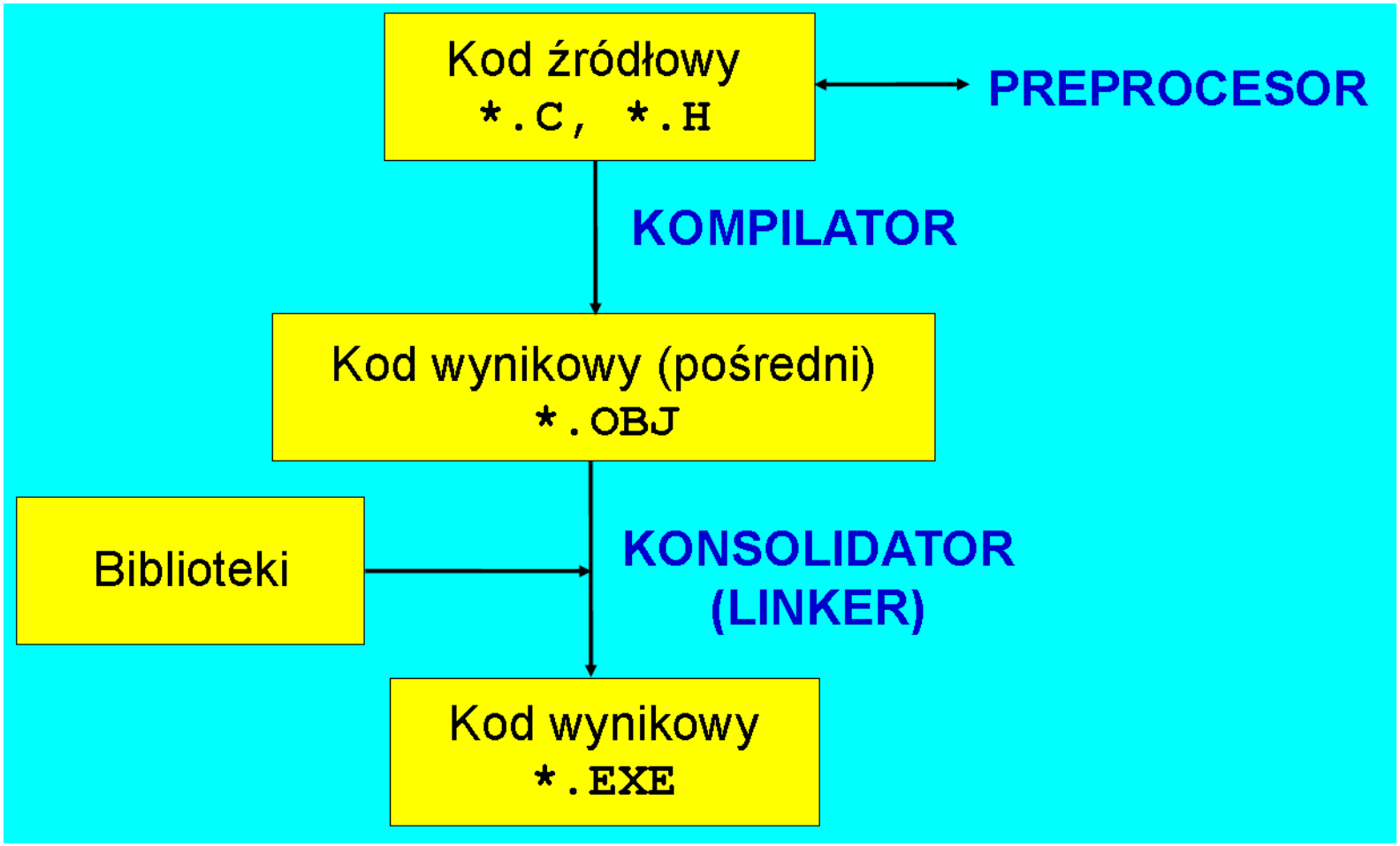
Konsolidacja

- **Łączenie modułów, konsolidacja** (ang. *linking*),
  1) w procesie tłumaczenia przebieg polegający na wykonaniu programu łączącego (łączenie modułów statyczne), prowadzący do utworzenia programu ładowalnego;
  2) dołączanie modułów bibliotecznych do wykonywanego programu (łączenie modułów dynamiczne).

Preprocesor (ang. *preprocessor*)

1) w systemach programowania C i C++ program wykonujący pierwszy przebieg analizowania pliku z kodem źródłowym. Preprocesor rozpoznaje i interpretuje zamieszczone w tekście programu dyrektywy, takie jak **#include, #define, #ifdef, #endif,** umożliwiające zagnieżdżanie plików na wejściu dla kompilatora, definiowanie napisów (m. in. stałych), kompilację warunkową itp. Tekst wytworzony przez preprocesor jest podawany na wejściu kompilatora;
2) program dokonujący wstępnych przekształceń w danych poddawanych następnie dalszej obróbce.

Kompilacja kodu źródłowego w C



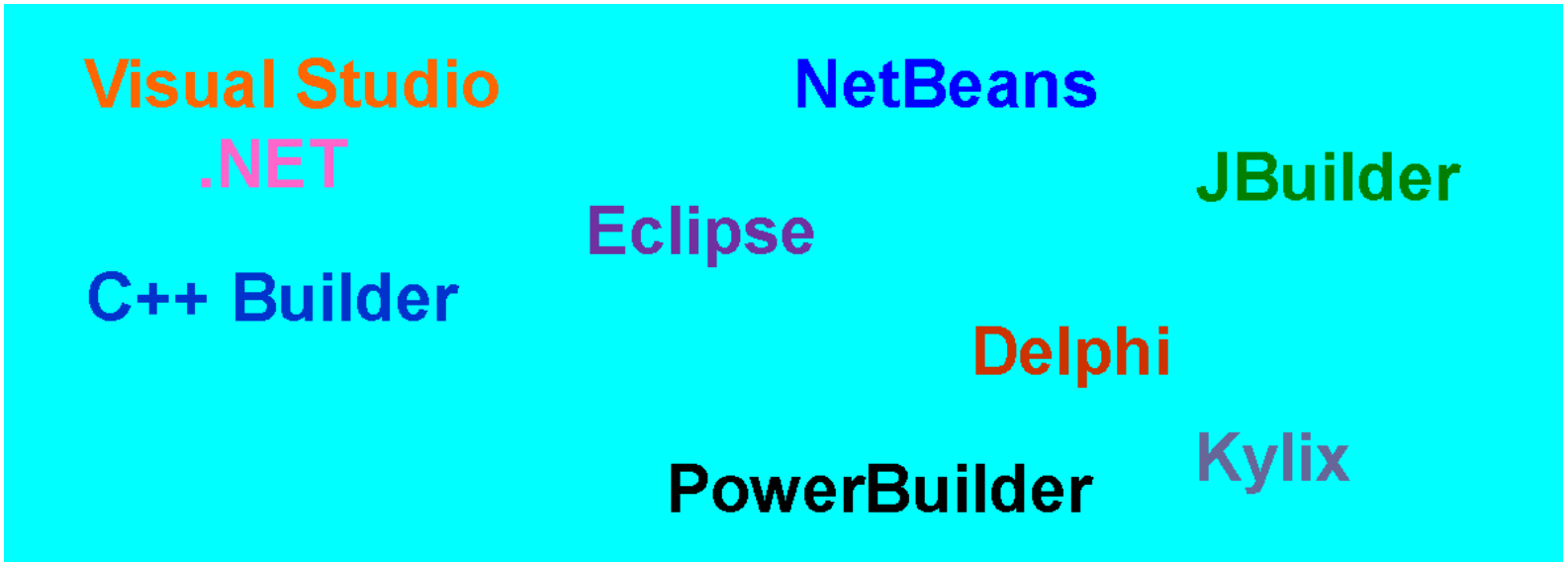
Oprogramowanie

- **System programowania** (angielskie *programming system*), oprogramowanie, w którego skład wchodzi kompilator lub interpretator języka programowania, system wykonawczy oraz biblioteki; **warsztat pracy programisty.**

IDE

1. (*Integrated Develompent Environment*), zespolone środowisko rozwojowe; połączenie w jedną funkcjonalną całość programów używanych do wytwarzania oprogramowania: edytora, kompilatora, konsolidatora, zazwyczaj realizowane w formie środowiska okienkowego;
2. (*Integrated Drive Electronics*), jeden z wielu standardów interfejsu dysku twardego, zaproponowany w 1983 r. przez firmę Compaq, mieszczący większość układów sterownika dysku w obudowie napędu dyskowego.

IDE


Visual Studio
.NET
NetBeans
JBuilder
Eclipse
C++ Builder
Delphi
Kylix
PowerBuilder

RAD

- **Programowanie RAD, Rapid Application Development**, termin komercyjny ("błyskawiczne opracowywanie aplikacji") określający możliwości pakietu oprogramowania działającego w graficznym środowisku okienkowym, służącego do zestawiania aplikacji, w szczególności – ich interfejsów, z modułów wybieranych z bibliotek udostępnianych w postaci drzew tematycznych.

RAD cd.

- W zależności od wybranego elementu pakiet RAD wytwarza **fragmenty kodu** (np. w języku C++) wraz z odpowiednimi wywołaniami. Szczegóły odbiegające od standardu uzupełnia ręcznie programista. Aplikacje wytworzone w środowiskach RAD są z reguły **słabo przenośne** i zajmują dużo pamięci, jednak możliwość automatycznego oprogramowania ich najżmudniejszych elementów (interfejs graficzny) oraz ułatwiony kontakt ze standardowymi bazami danych powoduje, że programowanie RAD staje się coraz popularniejsze.

API

- **API, Aplication Programming Interface**, polecenia dzięki którym użytkownik ma dostęp do funkcji systemu operacyjnego, takich jak na przykład tworzenie interfejsu graficznego. Za pomocą API programista może również otrzymywać informacje o systemie, a także kontaktować się z urządzeniami peryferyjnymi.

Typy danych

- język C/C++ składa się z wyrażeń operujących na zmiennych,
- zmienna –
  - obiekt będący instancją danego typu, realizacją praktyczną – miejscem w pamięci komputera, którego zawartość interpretuje się wg typu zmiennej,
- typ –
  - formalny opis operacji i rodzaju informacji,
- w C/C++ mamy następujące typy:
  - proste
  - złożone

Typy danych

Typy proste:

- zmiennoprzecinkowe
- całkowitoliczbowe
- specjalne

Typy zmiennoprzecinkowe

- float – zmiennoprzecinkowe, wartości pojedynczej precyzji, pamięć interpretowana zgodnie ze standardem IEEE-754
- double – j. w. z tym, że podwójnej precyzji.

Typy danych

Typ całkowitoliczbowy

- bool – typ całkowity, wartości logiczne: *true*, *false*
- char – typ znakowy, wartości całkowite interpretowane jako kod ASCII znaku
- int – typ całkowity, wartości całkowite z przedziału wynikającego z ilości zajmowanego miejsca w pamięci
- enum – typ wyliczeniowy, wartości całkowite interpretowane jako literały występujące w definicji typu

Typy danych

Typy specjalne:

- **void** – typ pusty, bez wartości; nie można zadeklarować obiektu tego typu – służy wyłącznie do zaznaczenia, że funkcja nie zwraca żadnej wartości lub też nie pobiera żadnego argumentu (opcjonalne)
- **wskaźnik** (*) – ”szwajcarski scyzoryk” języka C/C++, wartości całkowite interpretowane jako wskazanie (adres) do miejsca w pamięci zajmowanego przez zmienną danego typu
- **referencja** (&) - rodzaj wskaźnika, któremu wartość można nadać tylko raz; w założeniu miał być bezpiecznym wskaźnikiem

Typy danych

Typy złożone

- **struct** – struktura, spójny ciąg danych (pól, atrybutów)
- **union** – unia spójny, zbiór danych (pól, atrybutów) zajmujących *ten sam obszar pamięci*
- **class** – klasa, spójny zbiór danych (pól, atrybutów) opisujących obiekt w klasie problemu wraz z funkcjami na nich operującymi; interpretacja obszaru pamięci niejednoznaczna
- **tablica** – jednolity, ciągły obszar pamięci zajmowany przez n elementów dowolnego, ale tego samego typu

Literały (*literal constants*)

- Są to pojawiające się w kodzie **dane zapisane w sposób dosłowny**. Nie mogą się one zmieniać w czasie wykonywania programu. Nie mylić ze stałymi. Np.: W wyrażeniu:
  int a;
  a = 2 * 4;
  Pojawiają się dwa literały 2 oraz 4. Nie mają one swojego miejsca w pamięci w czasie uruchomionego programu (nie adresuje się ich).

- **30** to 30 zapisane dziesiętnie,
  036 30 w systemie ósemkowym,
  0x1e 30 w zapisie szesnastkowym,
  3E2 to 300 zapisane wykładniczo ($m * 10^n$ zapiszemy: mEn),
  'a' to literał znakowy, tak zapisujemy jeden znak, np.: char zn = 'a'; Do zmiennej znakowej zapisujemy literał znakowy: a.
  "C++" to ciąg znaków. Ciągi znaków zawsze zakończone są znakiem **null** (o kodzie 0). Np.: char ciag[4] = "C++";

Stałe (*constants*)

- Są to wyrażenia, które przyjmują konkretną, zdefiniowaną wartość. Wyrażenia te muszą należeć do jednego z predefiniowanych typów. Stałe nie mogą zmienić swoich wartości w czasie działania programu, są tylko do odczytu (*read-only*). Np.:
  const double pi = 3.141592;
- Stałe muszą mieć nadaną wartość przy deklaracji.

Komentarze

- Komentarze to elementy programu przydatne programistom, nie mające wpływu na sam program. Komentarze kilku liniowe rozpoczynamy znakami /* i kończymy */. Komentarz rozpoczęty znakami // kończy się końcem linii. Np.:

```
int a=0; // to jest komentarz
int b=1; /* to też jest komentarz
ale kończy się dopiero tutaj */
```

void

- Jest to pusty typ danych. Używa się go do deklarowania funkcji, która nie pobiera i/lub nie zwraca żadnych danych oraz do deklaracji ogólnych wskaźników (nie wiemy, jaki typ danych jest zapisany w pamięci, w miejscu, do którego ten wskaźnik prowadzi).

char

- Jest to 8 bitowy typ danych. Służy przede wszystkim do przechowywania **znaków**, choć można też go stosować do przechowywania wartości liczbowych. Stosowany w postaci:
  char lub **signed char** przechowuje wartości całkowite z zakresu: **<-128; 127>**
  unsigned char przechowuje wartości całkowite z zakresu: **<0; 255>**

int

- Jest to typ danych służący do przechowywania liczb całkowitych. Może być stosowany z modyfikatorami: **signed** (domyślnie), **unsigned**, **short** i **long**. Ilość pamięci przeznaczona na ten typ zależy od tego, ilu bitowy jest system operacyjny. Np.: MS-DOS 16-bitowy, MS-Windows 9x 32-bitowy, Linux RedHat 32-bitowy.

int

- reprezentacja binarna U2, np. w rejestrze procesora, w pamięci, jest taka sama:

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1

- Dla liczby bez znaku

$2^{15} + 2^{10} + 2^{0} = 32768 + 1024 + 1 = 33793$

- Dla liczby ze znakiem

$-2^{15} + 2^{10} + 2^{0} = -32768 + 1024 + 1 = -31743$

Int – Uwaga!

- ta sama zawartość pamięci, rejestru ma dwie różne interpretacje
- dwa rożne typy: signed ≠ unsigned
- specjalne znaczenie wartości ”-1”, dla typu ze znakiem oznacza, że wszystkie bity mają wartość 1
- dla 16 bitow => -1 = 65535 = 0xFFFF
- typ *char domyślnie jest typem ze znakiem* (- 128 .. 127)
- kody ASCII są liczbami bez znaku (0 .. 255)

Int – Uwaga!

- porównywanie liczby ze znakiem i bez znaku daje wynik *nieokreślony!*,
- mieszanie typów *signed i unsigned wymaga* większej staranności i uwagi
- przekroczenie zakresu typu w wyniku dodawania (dla kodu U2)
  - dla zmiennej 16 bitowej bez znaku:
    - 32767 + 1 = 32768
    - 65535 + 1 = 0
  - dla zmiennej 16 bitowej ze znakiem:
    - 32767 + 1 = -32768
    - 65535 + 1 = 0 (-1 + 1 = 0)

Int –przekroczenia zakresu

- przekroczenie zakresu w wyniku mnożenia
  - dla liczby 16-bitowej bez znaku:
    - 256 * 256 = 0
  - dla liczby 16-bitowej ze znakiem:
    - 128 * 256 = -32768;
- w ogólnym przypadku mnożenie dwóch liczb N - bitowych daje w wyniku liczbę 2N – bitową
- przy małych liczbach problem nie jest dostrzegany
- przy skomplikowanych wyrażeniach taki błąd jest trudny do znalezienia

float

- Jest to 32-bitowy typ danych służący do przechowywania liczb zmiennoprzecinkowych (niekoniecznie całkowitych) **float** przechowuje wartości z zakresu: **$<3.4*10^{-38}; 3.4*10^{38}>$**

double

- Jest to 64 lub 80-bitowy typ danych służący do przechowywania liczb zmiennoprzecinkowych (niekoniecznie całkowitych)
  double przechowuje wartości z zakresu:
  $<1.7*10^{-308}; 1.7*10^{308}>$
  long double przechowuje wartości z zakresu:
  $<3.4*10^{-4932}; 3.4*10^{4932}>$

Prosty program

```
int main() {
   int a; //zmienna a
   int b;
   int c;
   a = 2;
   b = 7;
   c = a * b;
   return 0;
}
```

Zmienne

- deklaracja zmiennej poprzedzona jest jej typem np. *int a;*
- deklaracja kończy się średnikiem
- dla typów całkowitych typ zmiennej można poprzedzić kwalifikatorem znaku lub długości np.:
  - signed int a;
  - unsigned short a;
  - unsigned a;

Zmienne

- zmienna typu całkowitego *int jest domyślnie* definiowana jako zmienna ze znakiem
- deklaracja zmiennej typu *int może odbywać się* również za pomocą samych kwalifikatorów
- Jeżeli deklaruje się kilka zmiennych tego samego typu można to zrobić jednym wyrażeniem, np.:
  - unsigned int a, b, c, d;
- deklaracja zmiennej może, a czasami nawet powinna, wiązać się z nadaniem jest tzw. wartości początkowej np. int a=2;

Zmienne

- zmienne globalne
  - deklarowane poza funkcją *main*
  - domyślna wartość początkowa – 0
  - mogą być wykorzystywane przez dowolną funkcję
  - deklaracja może być łączona z kwalifikatorem *extern*
  - globalna zmienna zewnętrzna musi być zadeklarowana w innym pliku
  - deklaracja może być łączona z kwalifikatorem *static*
  - globalna zmienna statyczna jest widoczna tylko w obrębie danego pliku

Zmienne

- zmienne lokalne
  - deklarowane zazwyczaj na początku funkcji
  - można je deklarować w ciele funkcji (deklaracja mieszana)
  - deklaracja zmiennej musi poprzedzać jej użycie
  - mogą być wykorzystywane tylko przez daną funkcję
  - argumenty funkcji traktowane są jak zmienne lokalne
  - zmienne lokalne mogą być deklarowane jako statyczne
  - zmienne statyczne zachowują swoją wartość pomiędzy kolejnymi wywołaniami funkcji

Prosty program

```
int a;
int main() {
   int b;
   int c;
   a = 2;
   b = 7;
   c = a * b;
   return 0;
}
```


zmienna globalna

zmienna lokalna

Operatory arytmetyczne

- przypisania: a = b
- dodawania: a + b
- odejmowania: a – b
- mnożenie: a * b
- dzielenie: a / b
- modulo (reszta z dzielenia): a % b
- post i preinkrementacji: a++ i ++a
- post i predekrementacji: a-- i --a

Operatory porównania

- mniejsze: a < b
- większe: a > b
- mniejsze lub równe: a <= b
- większe lub równe: a >= b
- różne: a != b
- równe: a == b

Operatory logiczne i bitowe

- logiczne
  - logiczna negacja: !a
  - logiczne i: a && b
  - logiczne lub: a || b
- bitowe:
  - bitowa negacja: ~a
  - bitowe i: a & b
  - bitowe lub: a | b
  - bitowe rozłączne lub (xor): a ^ b
  - przesunięcie bitowe w lewo: a << b
  - przesunięcie bitowe w prawo: a >> b

Operatory skrócone

- dodawanie: a += b
  - a = a + b
- odejmowanie: a -= b
  - a = a - b
- mnożenie: a *= b
  - a = a * b
- dzielenie: a /= b
  - a = a / b
- reszta z dzielenia: a %= b
  - a = a % b

Operatory skrócone

- bitowe i: a &= b
  - a = a & b
- bitowe lub: a |= b
  - a = a | b
- bitowe rozłączne lub: a ^= b
  - a = a ^ b
- bitowe przesunięcie w lewo: a <<= b
  - a = a << b
- bitowe przesunięcie w prawo: a >>= b
  - a = a >> b

Wyrażenia

- wyrażenia są połączeniem operatorów i zmiennych
- kolejność działań wynika z priorytetów operatorów
  - a = a + b * c & d
- wynik pojedynczego wyrażenia może być wykorzystywany w innym wyrażeniu (łańcuch)
  - d = a = (b += 2) – (c -= 3);
- wyrażenia mogą posiadać tzw. skutek uboczny
  - a = b + c++;

Wyrażenia

- wyrażenia mogą mieć wartość zależną od implementacji
  - a = a + b * c / d
- wartość wyrażeń może być nieokreślona
  - c = ++a + a--;
  - int func(int a) { return 2 * a; }
  - c = (a /= 3) + func(a);
- wynik dzielenia przez 0 jest zależny od implementacji
- operatora przypisania nie należy stosować w wyrażeniach warunkowych (if, while, switch)