Programowanie obiektowe

Paweł Mikołajczak

1. Wprowadzenie

Web site: informatyka.umcs.lublin.pl

Wykład 1

- ***Wstęp***
- ***Rys historyczny.***
- ***Nowe elementy C++11***
- ***Paradygmaty programowania.***
- ***Kod źródłowy i kompilacja***
- ***Obiekty i klasy***
- ***Hermetyzacja danych***
- ***Dziedziczenie***
- ***Polimorfizm***
- ***Podsumowanie terminologii***

Rodzina języków C

Należy przypomnieć, że język C jest integralnie związany z systemem Unix. Język C++ został zaprojektowany jako obiektowa wersja języka C.

W języku C++ powstało i powstaje wiele znaczącego oprogramowania: systemy operacyjne, programy narzędziowe, programy użytkowe, programy do obsługi sieci komputerowych. Obecnie od programisty wymaga się znajomości przynajmniej dwóch języków programowania i znajomości minimum dwóch systemów operacyjnych, język C/C++ i Unix (Linux) a także Windows są bezwzględnie wymagane.

Systemy operacyjne na urządzenia mobilne (Android, IOS) są równie ważne. Obecnie (rok 2018) ważnym jest język C++ , integralnie związany z programowaniem obiektowym.

Język C++ jest ***nadzbiorem*** języka C. Aby przyspieszyć projektowanie i implementowanie programów, zostały opracowane specjalne systemy wspomagające prace programistów.

Rodzina języków C

Doskonałymi narzędziami do tworzenia aplikacji są pakiety takie jak np. Qt czy Visual C++ .

Te produkty należą do systemów szybkiego projektowania aplikacji (ang. RAD - Rapid Application Development).

Korzystając z tych pakietów możemy efektywnie konstruować 32-bitowe programy pracujące w systemie Windows i innych.

Wydaje się rozsądne, aby dobrze wykształcony informatyk opanował następujące narzędzia programistyczne:

język C
język Objective-C lub/i Swift
język C++
język Java
język C#
JavaScript & jQuery

Zalecana literatura

Skrypty akademickie

1. P. Mikołajczak, Język C - podstawy programowania, Skrypt Akademicki, UMCS, Lublin 2011, skrypt dostępny na stronach UMCS
2. P. Mikołajczak, Język C++ - podstawy programowania, Skrypt Akademicki, UMCS, Lublin 2011 , skrypt dostępny na stronach UMCS
3. P. Mikołajczak, Programowanie generyczne w Qt, Skrypt Akademicki, UMCS, Lublin 2011 , skrypt dostępny na stronach UMCS
4. P. Mikołajczak, Wprowadzenie do STL, Skrypt Akademicki, UMCS, Lublin 2011, skrypt dostępny na stronach UMCS
5. P. Mikołajczak, M. Ważny, Metody numeryczne w C++, Skrypt Akademicki, UMCS, Lublin 2011 , skrypt dostępny na stronach UMCS

Kultowe podręczniki

1. B.Stroustrup, Język C++, WNT, Warszawa, 1994, 700 stron
2. B.Stroustrup, Programowanie, Teoria i praktyka z wykorzystaniem C++, Helion, Gliwice, 2010, 1106 stron
3. B.Stroustrup, Język C++, Helion, Gliwice, 2013, 1292 stron
4. N. Josuttis, C++, Biblioteka standardowa, Helion, Gliwice, 1999, 726 stron
5. N. Josuttis, C++, Biblioteka standardowa, Helion, Gliwice, 2012, 1120 stron
6. S.Prata, Język C++ - szkoła programowania, Helion, Gliwice, 2006,1303 stron
7. S.Prata, Język C++ - szkoła programowania, Helion, Gliwice, 2012,1194 stron

Zalecana literatura

Polecane podręczniki

1. H.Deitel, P.Deitel, C++ - programowanie, RM, Warszawa, 1998, 1082 strony
2. B.Eckel, Thinking in C++, Helion, Tom 1,Gliwice, 2002, 642 strony
3. B.Eckel, Thinking in C++, Helion, Tom 2,Gliwice, 2004, 688 stron
4. A.Koenig, B. Moo, C++ - potęga języka, Helion, Gliwice, 2004, 428 stron
5. A.Koenig, B. Moo, C++ Koncepcje i techniki programowania, Helion, Gliwice, 2004, 385 stron
6. J.Coplien, C++- styl i techniki zaawansowanego programowania, Helion, Gliwice, 2004, 478 stron

Dodatkowe podręczniki

1. T. Hansen, C++ - zadania i odpowiedzi, WNT, Warszawa, 1990, 617 stron
2. D.Stephens, C.Diggins, J.Turkanis, J.Cogswell, C++ - receptury, Helion, 2006, 556 stron
3. R.Neapolitan, K.Naimipou, Podstawy algorytmów z przykładami w C++, Helion, Gliwice, 2004, 648 stron
4. A.Drozdek, C++ - algorytmy i struktury danych, Helion, Gliwice, 2004, 612 stron
5. B.Stroustrup, Język C++, WNT, Warszawa, 1994, 700 stron
6. V.Shtern, C++ - inżynieria programowania, Helion, Gliwice, 2004, 1083 strony
7. N. Solter, S.Kleper, C++ - zaawansowane programowania, Helion, Gliwice, 2006, 907 stron

Rys historyczny – język C

Historia powstawania języka C a następnie C++ jest długa, rozwojem tych języków zajmowało się wielu wspaniałych fachowców. Język C powstał dzięki przejęciu podstawowych zasad z dwóch innych języków: BCPL i języka B. BCPL opracował w 1967 roku Martin Richards. Ken Thompson tworzył język B w oparciu o BCPL. W 1970 roku w Bell Laboratories na podstawie języka B opracowano system operacyjny UNIX.

Twórcą języka C jest Dennis M. Ritchie, który także pracował w Bell Laboratories. W 1972 roku język C był implementowany na komputerze DEC PDP-11. Jedną z najważniejszych cech języka C jest to, że programy pisane w tym języku na konkretnym typie komputera, można bez większych kłopotów przenosić na inne typy komputerów. Język C był intensywnie rozwijany w latach siedemdziesiątych. Za pierwszy standard przyjęto opis języka zamieszczony w dodatku pt. "C Reference Manual" podręcznika „The C Programming Language”. Publikacja ukazała się w 1978 roku. Opublikowany podręcznik definiował standard języka C, reguły opisane w tym podręczniku nazywamy standardem K&R języka C.

Rys historyczny – język C

W 1983 roku Bell Laboratories wydało dokument pt. "The C Programming Language - Reference Manual", którego autorem był D. M. Ritchie.

W 1988 Kernighan i Ritchie opublikowali drugie wydanie "The C Programming Language".

Na podstawie tej publikacji, w 1989 roku Amerykański Narodowy Instytut Normalizacji ustalił standard zwany standardem ANSI języka C (zwany także standardem C90).

Standard ANSI C99 został opublikowany w 1999 roku.

Najnowsza wersja języka C (stan na rok 2018) zawiera zmiany wprowadzone do języka C++, jest to standard C++11.

Dalsze zmiany (C++14 i C++17) są wprowadzane!

Rys historyczny – język C++

Za twórcę języka C++ uważany jest Bjarne Stroustrup, który w latach osiemdziesiątych pracując w Bell Laboratories tworzył i rozwijał ten język.

Bjarne Stroustrup wspólnie z Margaret Ellis opublikował podręcznik „The Annotated C++ Reference Manual”.

W 1994 roku Amerykański Narodowy Instytut Normalizacji opublikował zarys standardu C++.

Nowym językiem, silnie rozwijanym od 1995 roku jest język programowania Java. Język Java opracowany w firmie Sun Microsystem, jest oparty na języku C i C++. Java zawiera biblioteki klas ze składnikami oprogramowania do tworzenia multimediów, sieci, wielowątkowości, grafiki, dostępu do baz danych i wiele innych.

Rys historyczny – język C++

Istnieje wiele kompilatorów języka C++. W 1997 roku ukazało się trzecie wydanie książki Bjarne'a Stroustrupa **„Język programowania C++”**. Oczywiście język C++ jest nadal rozwijany, ale uznano, że trzecie wydanie podręcznika B.Stroustrupa wyczerpująco ujmuje nowe elementy języka i *de facto* może być traktowane jako standard. Ostatecznie standard języka C++ został przyjęty przez komitet ANSI/ISO w 1998 roku, została wtedy ustalona jednolita specyfikacja języka. Standard ten o nazwie C++98 wprowadzał do języka RTTI (mechanizm identyfikacji czasu wykonania), szablony a także wprowadził wspaniałą bibliotekę programistyczną (opartą na szablonach) zwaną STL (Standard Template Library).

W 2002 roku przyjęto kolejną poprawioną wersję tego standardu. Dokument opisujący ten standard jest dostępny za pośrednictwem sieci Internet pod numerem 14882 na stronie http://www.ansi.org.

Rys historyczny – język C++

W 2003 roku wydana została nowa specyfikacja języka, usuwała ona pewne nieścisłości poprzedniej specyfikacji. Ta wersja języka oznaczana jest jako C++03.

Prace nad rozwojem języka trwają cały czas.

W roku 2011 komitet ISO zaakceptował nowy standard języka C++ , znany jako ***język C++11,*** formalnie standard ten jest oznaczany jako ISO/IEC 14882:2011.

Ten standard zawiera istotne nowości. Informacje na temat standardu zawarte są w dokumencie N3242.

(P.Becker, ed. Working draft, Standard for programming Language C++ (C++11),

http:/www.open-std/jtc1/sc22/wg/21/docs/papers/2011/n3242.pdf

W standardach języka C++ musi być spełniona zasada zgodności wstecznej. Oznacza to, że wszystko co da się skompilować kompilatorem standardu C++03 powinno być skompilowane językiem C++11. Są jednak wyjątki ponieważ w C++11 wprowadzono istotne zmiany. Można wykryć wersje języka przy pomocy makrodefinicji __cplusplus

```
#define __cplusplus  201103L    // dla kompilatora C++11
#define __cplusplus  199711L    // dla kompilatora C++98 i C++03
```

Nowe elementy języka C++11

Surprisingly, C++11 feels like a new language: The pieces just fit together better than they use to and I find a higher-level style of programming more natural than before and as efficient as ever.

Bjarne Stroustrup.

Nowe elementy języka C++11

Według Nicolai Josuttisa nowości w standardzie języka C++11 to:

- Istotne pomniejsze porządki składniowe (***nullptr i nullptr_t***)
- Automatyczna dedukcja typu ze słowem ***auto***
- Jednolita składnia inicjalizacji i listy inicjalizacyjnej
- Pętle zakresowe (**for** (*element : kolekcja*))
- Semantyka przeniesienia i referencje do r-wartości
- Nowe literały napisowe (np. **R"(\\n)"**)
- Słowo ***noexcept*** (funkcja nie może rzucać wyjątków)
- Słowo ***constexpr*** (wymuszanie obliczenia wyrażenia w czasie wykonania)
- Nowe elementy szablonów (np. zmienna lista argumentów)
- Lambdy (pozwalają definiować funkcjonalność wprost w miejscu użycia) (symbol [ ])
- Słowo ***decltype*** (kompilator określa typ wyrażenia)
- Nowa składnia deklaracji funkcji
- Klasy wyliczeniowe (wyliczenia ograniczone)
- Nowe typy podstawowe (*long long, unsigned long long, nullptr_t, char16_7*)

Nowe elementy języka C++11

Mamy: 41 zmian w języku
27 zmian w STL

Nowe narzędzia językowe:

- Jednolita i ogólna inicjalizacja przy użyciu list {}
- Dedukcja typów z inicjatora: auto
- Zapobieganie zawężaniu
- Ogólne i gwarantowane wyrażenia stałe: constexpr
- Zakresowa instrukcja for
- Słowo kluczowe oznaczające wskaźnik pusty: nullptr
- Ściśle typowane wyliczenia wyznaczające zakres: enum class
- Asercja czasu kompilacji: static_assert
- Językowe odwzorowanie list {} w std::initializes_list
- Referencje prawostronne
- Zagnieżdżone argumenty szablonów zakończone >> (bez odstępu)
- Lambdy

Nowe elementy języka C++11

- Szablony zmienne
- Aliasy typów i szablonów
- Znaki Unicode
- Typ całkowitoliczbowy long long
- Kontrola wyrównania: alignas i alignof
- Możliwość używania typu wyrażenia jako typu w deklaracji: decltype
- Surowe literały łańcuchowe
- Ogólne obiekty POD
- Ogólne unie
- Klasy lokalne jako argumenty szablonowe
- Przyrostkowa składnia typu zwrotnego
- Składnia definiowania atrybutów i dwa atrybuty: carries_dependency i noreturn
- Wyłączanie propagacji wyjątków: noexcept
- Sprawdzanie możliwości wystąpienia wyjątku: operator noexcept
- Składniki C99

Nowe elementy języka C++11

- _func_
- Przestrzenie nazw inline
- Delegacja konstruktorów
- Wewnątrz klasowe inicjatory składowych
- Kontrolowanie ustawień domyślnych: default i delete
- Operator jawnej konwersji
- Definiowanie literałów przez użytkownika
- Kontrola nad konkretyzacją szablonów: szablon ex tern
- Domyślne argumenty szablonowe dla szablonów funkcji
- Dziedziczenie konstruktorów
- Kontrolowanie przesyłania: override i final
- Uproszczona zasada SFINAE
- Model pamięci
- Pamięć lokalna wątków: thread_local

Nowe elementy języka C++11

Nowe składniki biblioteki standardowej (STL)

- Konstruktory initializer_list dla kontenerów
- Semantyka przenoszenia dla kontenerów
- Lista jednokierunkowa: forward_list
- Kontenery mieszające:
  - unordered_map
  - unoredered_multimap
  - unordered_set
  - unordered_multiset
- Wskaźniki do zarządzania zasobami:
  - unique_ptr
  - shared_ptr
  - weak_ptr
- Narzędzia do współbieżności:
  - thread
  - muteksy
  - blokady
  - zmienne warunkowe

Nowe elementy języka C++11

- Wysokopoziomowe narzędzia do współbieżności:
  - packaged_thread
  - future
  - promise
  - asnc
- Krotki
- Wyrażenia regularne: regex
- Liczby losowe:
  - uniform_int_distribution
  - normal_distribution
  - random_engine, itp.
- Nazwy typów całkowitoliczbowych:
  - int16_t
  - uint32_t
  - int_fast64_t
- Ciągły kontener sekwencyjny o stałym rozmiarze: array
- Kopiowanie i powtórne zgłaszanie wyjątków

Nowe elementy języka C++11

- Raportowanie błędów przy użyciu kodów błędów: system_error
- Operacje emplace() dla kontenerów
- Szerokie zastosowanie funkcji constexpr
- Systematyczne stosowanie funkcji noexcept
- Udoskonalone adaptacje funkcji: function i bind()
- Konwersja typu string na typy wartościowe
- Alokatory zakresu
- Cechy typów np. is_integral i is_base_of
- Narzędzia do pracy z czasem: duration i time_point
- Arytmetyka liczb wymiernych w czasie kompilacji: ratio
- Porzucenie procesu: buick_exit
- Dodatkowe algorytmy:
  - move()
  - copy_if()
  - is_sorted()
- API usuwania nieużytków
- Pomoc do niskopoziomowego programowania współbieżnego: atomic

Paradygmaty programowania.

Mamy wiele metod programowania w języku C++, wydaje się jednak, że decydujące znaczenie mają cztery paradygmaty programowania:

- Programowanie proceduralne
- Programowanie strukturalne
- Programowanie zorientowane obiektowo
- Programowanie uogólnione (ang. generic programming)

Powstanie i rozwój języków i stylów programowania ma swój początek w pracach matematyków z pierwszej połowy XX wieku. Wprowadzono wtedy pojęcie funkcji obliczalnej, oznaczające funkcję, której wartość dla dowolnych wskazanych argumentów można obliczyć w sposób efektywny (w skończonym czasie).

Program w ***języku proceduralnym*** jest zestawem instrukcji definiujących algorytm działania. Do grupy języków proceduralnych zaliczają się: assemblery oraz języki wysokiego poziomu, takie jak Fortran, Basic, Pascal, język C.

Każdy algorytm składa się z ***opisu podmiotów*** oraz ***opisu czynności***, które mają być na tych podmiotach wykonane. W zapisanym w języku proceduralnym programie mamy : ***dane*** (struktury danych) i ***instrukcje***.

Paradygmaty programowania.

Modyfikując styl programowania proceduralnego wypracowano koncepcję tzw. ***programowania strukturalnego***. Zakłada ona grupowanie fragmentów kodu w podprogramy (procedury i funkcje) i definiowanie zasad komunikowania się tych podprogramów między sobą. Istotne zalety programowania strukturalnego polegały na poprawie czytelności programu i możliwości tworzenia i użytkowania bibliotek podprogramów. Styl programowania strukturalnego doskonale nadawał się do pisania małych i średnio dużych programów przenośnych, nie sprawdzał się (był kosztowny i generował dużo błędów) przy realizacji dużych i bardzo zaawansowanych programów. Coraz większe wymagania stawiane programistom spowodowały powstanie nowego stylu programowania – ***programowania obiektowego***. Na jego gruncie wyrosła szeroka klasa języków obiektowych, specjalizowanych, (***Smalltalk***), jak i ogólnego przeznaczenia, np. C++, Objective-C, Object Pascal, Java, C#.

W programowaniu proceduralnym i strukturalnym program jest traktowany jako seria procedur (funkcji) działających na danych. Dane są całkowicie odseparowane od procedur, programista musi pamiętać, które funkcje były wywołane i jakie dane zostały zmienione.

Paradygmaty programowania.

Przy realizacji dużych programów, dominuje programowanie obiektowe. Programowanie ***zorientowane obiektowo*** dostarcza technik zarządzania złożonymi elementami, umożliwia ponowne wykorzystanie komponentów i łączy w logiczną całość dane oraz manipulujące nimi funkcje. Zadaniem programowania zorientowanego obiektowo jest modelowanie ***„obiektów”*** a nie ***danych***.

Język C++ został stworzony dla programowania zorientowanego obiektowo. Zasadniczymi elementami stylu programowania obiektowego są:

- klasy (abstrakcyjne typy danych)
- hermetyzacja danych (ukrywanie danych)
- dziedziczenie
- polimorfizm

Paradygmaty programowania.

Bruce Eckel w podręczniku „Thinking in C++” przytoczył pięć podstawowych cech języka Smalltalk (był to jeden z pierwszych języków obiektowych).

1. Wszystko jest obiektem. Obiekt jest szczególnym rodzajem zmiennej, przechowuje ona dane, ale może także wykonać operacje na sobie samej
2. Program jest grupą obiektów, przekazujących sobie wzajemnie informacje o tym, co należy zrobić za pomocą komunikatów. Komunikat może być traktowany jako żądanie wywołania funkcji należącej do tego obiektu.
3. Każdy obiekt posiada własną pamięć, złożoną z innych obiektów. Oznacza to, że nowy rodzaj obiektu jest tworzony przez utworzenie pakietu złożonego z już istniejących obiektów.
4. Każdy obiekt posiada ***typ***. Mówimy, że każdy obiekt jest ***egzemplarzem*** jakiejś ***klasy***. ***Klasa jest synonimem słowa typ.***
5. Wszystkie obiekty określonego typu mogą odbierać te same komunikaty,

Paradygmaty programowania.

Najnowszym paradygmatem programowania jest ***programowanie uogólnione ,*** zwane także ***programowaniem ogólnym*** albo też ***programowaniem generycznym*** (ang. generic programming).

Według Bjarne Stroustrupa:

„Programowanie ogólne – ten styl programowania skupia się na projektowaniu, implementacji i wykorzystaniu hierarchii klas. Oprócz definiowania siatek klas język C++ udostępnia wiele narzędzi do nawigacji po tych siatkach oraz do upraszczania definicji klas tworzonych na bazie istniejących klas. Hierarchie klas umożliwiają polimorfizm w czasie działania programu oraz hermetyzację”.

Według Stephena Prata , to kolejny paradygmat programowania:

„jego cechą wspólną z programowaniem obiektowym jest dążenie do uproszczenia wielokrotnego wykorzystania kodu oraz technika wydzielania abstrakcyjnych pojęć ogólnych. O ile jednak w przypadku obiektowości największy nacisk kładziony jest na dane, o tyle w programowaniu uogólnionym akcentuje się uniezależnienie operacji od typów danych. ….Pojęcie *uogólnione* odnosi się do tworzenia kodu niezależnego od konkretnych typów.”

Paradygmaty programowania.

Według Bjarne Stroustrupa klasyfikacja paradygmatów programowania ma postać:

- Programowanie proceduralne – jest to styl programowania skoncentrowany na przetwarzaniu i projektowaniu odpowiednich struktur danych(język C) .
- Abstrakcja danych – jest to styl programowania skoncentrowany na projektowaniu interfejsów, ukrywaniu szczegółów implementacji w ogólności oraz reprezentacji w szczególności. Język C++ obsługuje zarówno klasy konkretne jak i klasy abstrakcyjne.
- Programowanie obiektowe – jest to styl programowanie skupiony na projektowaniu implementacji i wykorzystaniu hierarchii klas. Język obiektowy (C++) udostępnia wiele narzędzi do obsługi klas oraz do upraszczania definicji klas tworzonych na bazie istniejących klas. Hierarchia klas umożliwia polimorfizm w czasie działania programu.
- Programowanie ogólne – jest to styl programowania w centrum uwagi stawiający projektowanie, implementację oraz używanie ogólnych algorytmów. Słowo ***ogólny*** oznacza algorytm zaprojektowany w taki sposób, że może on działać na różnych typach. Ważnym elementem języka C++ w tym programowaniu są szablony, które pozwalają na polimorfizm parametryczny w czasie kompilacji

Kod źródłowy.

W naszym kursie podstawowym zasad programowania obiektowego koncentrujemy się na plikach z kodem źródłowym. Plik źródłowy może być utworzony na kilka sposobów. Wydaje się, że obecnie (rok 2017) najpopularniejszą metodą tworzenia plików źródłowych jest korzystanie ze ***zintegrowanego środowiska programistycznego*** (IDE, ang. Integrated Development Environment). W wielu środowiskach mamy całą obsługę – od napisania kodu do wykonania programu.

Takimi środowiskami z implementacją języka C++ są:

- Qt
- Microsoft Visual C++
- Embarcadero C++ Builder (dawniej Borland Builder ver.6)
- Apple Xcode
- Open Watcom C++
- Digital Mars C++
- Freescale Code Warrior
- Inne……

Kod źródłowy.

W wielu implementacjach wykonywana jest tylko kompilacja i konsolidacja, użytkownik musi wydawać polecenia (komendy) z poziomu wiersza poleceń (konsoli). Najbardziej popularne pakiety to:

- GNU C++ w systemach UNIX
- GNU C++ w systemach LINUX
- IBM XL C/C++ dla systemu AIX
- Darmowe wersje kompilatora Builder ver.5.5 (obecnie Embarcadero)
- Digital Mars

W systemach UNIX mamy do dyspozycji edytory takie jak vi, ed, ex, emax.

W systemach Windows możemy wykorzystać edytory edlin , edit i wiele innych.

W systemach Windows możemy także wykorzystać dowolny procesor tekstu zapisujacy plik jako standardowy plik ASCII (np. Notatnik) do napisania kodu źródłowego. Należy pamiętać, żeby w systemach Windows ***nie korzystać*** z edytora typu WORD (bo daje format procesora tekstu)

Nazewnictwo plików.

Bardzo ważne jest w konkretnej implementacji kompilatora C++ nazewnictwo plików.

Formalnie mamy zapis:

nazwa_pliku.rozszerzenie

Nazwa pliku jest dowolna natomiast rozszerzenie zależy od implementacji, możemy napisać np. prog_01.cpp

Implementacja	rozszerzenie
UNIX	C, cc, cxx, c
GNU C++	C, cc, cxx, cpp, c++
Digital Mars	cpp, cxx
Borland C++	cpp
Watcom C++	cpp
Mocrosoft Visual C++	cpp, cxx, cc
Freestyle Code Warrior	cpp, cp, cc, cxx, c++

Kompilacja w systemie UNIX (1)

System UNIX jest interesujący, ale…..

W systemie UNIX polecenie CC wykorzystywane jest do kompilacji programu źródłowego w języku C++. Pamiętajmy, że to polecenie musi być zapisane dużymi literami (polecenie cc powoduje kompiluje programy w języku C). Praktycznie kompilowanie programu o nazwie afera01.C polega na wydaniu polecenia:

CC afera01.C

Kompilator wygeneruje kod z plikiem wynikowym, nada mu rozszerzenie o.

W podanym przykładzie mamy:

afera01.o

W kolejnym kroku, kompilator automatycznie przekaże plik afera01.o konsolidatorowi i utworzy plik wykonywalny (nazwa domyślna) a.out

Program uruchamiamy pisząc polecenie:

a.out

Kompilacja w systemie UNIX (2)

System UNIX jest interesujący, ale…..

Musimy pamiętać o programach wieloplikowych. W takiej sytuacji kompilujemy program wypisując nazwy wszystkich plików wchodzących w skład projektu. Jeżeli mamy przykładowo program zbudowany z dwóch plików afera01 i afera02 to w wierszu poleceń piszemy:

```
CC afera01.C afera02.C
```

Pamiętamy, że gdy projekt składa się z wielu plików, kompilator nie usuwa pliku z kodem pośrednim. Jeżeli zatem zmienimy plik o nazwie afera01 to ponowna kompilacja polega na napisaniu polecenia:

```
CC afera01.C afera02.o
```

W ten sposób kompilowany jest ponownie plik afera01 i konsolidowany z plikiem (wcześniej skompilowanym) afera02.

Często musimy jawnie dołączyć bibliotekę . Przykładowo dołączenie biblioteki matematycznej polega na wydaniu polecenia:

```
CC afera03.C -lm
```

Kompilacja w systemie LINUX.

Najczęściej używanym kompilatorem języka C++ w systemie LINUX jest g++. Kompilator ten jest tworzony przez Free Software Foundation.

Aby utworzyć plik wykonywalny dla pliku afera01 i otrzymać plik wykonywalny a.out piszemy:

```
g++ afera01.cxx
```

Gdy zachodzi potrzeba dołączenia biblioteki C++ to należy użyć polecenia:

```
g++ afera01.cxx –lg++
```

Dla programów wieloplikowych musimy napisać polecenie z nazwami wielu plików:

```
g++ afera01.cxx afera02.cxx
```

Powstanie znowu plik wynikowy a.out.

Kompilator GNU C++ dostępny jest zarówno w środowiskach Unix jak i Windows.

Kompilacja w systemie Windows (1).

W systemach Windows mamy kilka możliwości kompilowania programów. Dwie popularne techniki:

- praca z wykorzystaniem okna trybu poleceń
- praca z kompilatorami „okienkowymi”

Korzystanie z okna trybu poleceń.

Tryb poleceń jest pozostałością po zapomnianym już systemie MS-DOS.

Możemy do testowania programów pobrać środowiska zawierające kompilator GNU C++, polecamy pakiety Cygwin i MinGW. Te pakiety mają doskonały kompilator g++.

W omawianej technice należy otworzyć okno trybu poleceń. Jeżeli mamy program o nazwie *afera01.cpp* to skompilowanie pliku źródłowego tego programu polega na wydaniu polecenia:

g++ afera01.cpp

W wyniku kompilacji otrzymamy plik wykonywalny o nazwie

a.exe

Kompilacja w systemie Windows (2).

Pakietów typu IDE przeznaczonych dla Windows jest bardzo dużo.

Polecamy następujące pakiety:

- Microsoft Visual C++ 20xx (xx oznacza kolejne edycje: 2010, 2012 itd.)
- Qt Development Software (kolejne wersje, zaczynając od Qt 4.7.0)

Należy dokładnie zapoznać się z technika obsługi tych pakietów. Są to pakiety o kolosalnych możliwościach. Dla naszych celów wystarczające jest korzystanie z edytora i opcji kompilatora, aby można było napisać program , skompilować go i wykonać. Często pakiety te maję opcję „console application”.

Przykładowo w Microsoft Visual C++ 2010 zaznaczamy opcję „Win 32 Console Application”.

W pakiecie Qt w oknie kreatora (wersja: Qt Creator 2.0.1, platforma od wersji Qt 4.7.0, (32 bit) wybieramy opcję „Aplikacja konsolowa Qt”.

Kompilacja w systemie Windows (3).

Ustalił się nieformalny standard obsługi pakietów typu IDE, gdy mamy już napisany plik źródłowy a chcemy go wykonać.

W menu okienkowym najczęściej mamy dostępne następujące opcje:

- Compile – prowadzi to do skompilowania przetwarzanego pliku źródłowego
- Build i Make – oznacza standardowa kompilacje wszystkich plików źródłowych projektu.
- Run i Execute – oznacza uruchomienie programu
- Debug – oznacza tryb diagnostyczny, często mamy możliwość śledzenia programu wykonywanego krok po kroku

W wielu platformach program wykona się szybko, a pakiet powróci do trybu edycji – użytkownik może nie zdążyć przeczytać wyników działania programu. W takim przypadku należy stworzyć procedurę „***przetrzymania ekranu wynikowego***”. Można w programie umieści dodatkowe instrukcji jak np. :

cin.get(); lub getche() ;

Czasem te polecenia trzeba podwoić.

Kompilacja w komputerach Macintosh

Ostatnio w Europie popularne stają się komputery firmy Apple. Komputery tej firmy pracują pod nadzorem systemu operacyjnego o nazwie Mac OS X. Firma Apple dostarcza także bardzo dobre środowisko programistyczne o nazwie Xcode.

Jest to typowe środowisko typu IDE, mamy możliwość instalowania kompilatorów do wyboru – rekomendujemy g++ i clang).

To co czyni komputery Apple atrakcyjnymi (jest to mój personalny punkt widzenia) to fakt, że możemy dzięki komputerom tej firmy projektować doskonałe aplikacje na systemy mobilne.

Firma Apple oferuje system operacyjny iOS wykorzystywany w urządzeniach mobilnych (główny konkurent systemu Android). System iOS 5 nadzoruje pracą wszystkich sprzętów firmy Apple, np. iPhon'ach, czy iPadach.

Tworzenie aplikacji wymaga pakietu iOS 5 SDK , języka Objective-C (lub inny) oraz narzędzi pomocniczych (np. framework Cocoa i Cocoa Touch).

Język Objective-C jest nadzbiorem języka C i jest językiem zorientowanym obiektowo.

Standardowo Xcode i SDK pozwalają na tworzenie aplikacji w C, C++ oraz ich odmianach bazujących na Ojective-C.

Kompilacja w pakiecie Qt

Do pisania programów komputerowych i tworzenia aplikacji w Instytucie Informatyki UMCS aktualnie (rok 2017) mocno rekomendowany jest pakiet Qt. Jest to bardzo zaawansowane IDE zawodowych programistów.

Szczegóły korzystania z Qt do pisania, kompilowania i uruchamianiu programów komputerowych dostępne są w skryptach akademickich:

- K. Kuczynski, P.Mikołajczak, M.Denkowski, R.Stęgierski, K.Dmitruk, M.Panczyk,
  Wstęp do programowania w Qt, Skrypt akademicki, UMCS, Lublin 2012, dostępny na stronach internetowych Instytutu Informatyki UMCS
- P. Mikołajczak,
  Programowanie generyczne w Qt, Skrypt akademicki, UMCS, Lublin 2012, dostępny na stronach internetowych Instytutu Informatyki UMCS

Obiekty i klasy

Kluczowe znaczenie w technologii programowania obiektowego mają ***obiekty*** i ***klasy***. Wydaje się, że człowiek postrzega świat jako zbiór obiektów. Oglądając obraz na ekranie monitora, widzimy np. raczej twarz *aktora* niż zbiór kolorowych pikseli. *Aktor* jest obiektem, który ma odpowiednie cechy i może wykonywać różnorodne akcje. Innym przykładem obiektu jest np. *samochód.* Również *samochód* ma odpowiednie cechy i może wykonywać różne działania.

Obiekty mają ***atrybuty*** takie jak np. wielkość, kolor, kształt. Obiekty mogą działać, np. *aktor* biegnie, *samochód* hamuje, itp. Obserwujemy i rozumiemy świat badając cechy obiektów i ich zachowanie. Różne obiekty mogą mieć podobne atrybuty i podobne działania. Obiekty takie jak *człowiek* czy *samochód* w istocie są bardzo skomplikowanymi i złożonymi obiektami, jest prawie niemożliwe dokładne ich opisanie. Abstrakcja umożliwia nam zredukowanie złożoności problemu.

Obiekty i klasy

Właściwości (cechy) i procesy (działania, akcje) zostają zredukowane do niezbędnych cech i akcji zgodnie z celami, jakie chcemy osiągnąć. Dzięki abstrakcji uzyskujemy możliwość rozsądnego opisania wybranych obiektów i zarządzania złożonymi systemami.

Różne obiekty mogą mieć podobne atrybuty.

Obiekty takie jak *samochód*, *samolot* i *okręt* mają wiele wspólnego (mają wagę, przewożą pasażerów, bilet na dany pojazd ma cenę, itp.). Obiekt nie musi reprezentować realnie istniejącego bytu. Może być całkowicie abstrakcyjny lub reprezentować jakiś proces. Obiekt może reprezentować np. *rozdawanie kart do pokera* lub reprezentować *mecz piłki nożnej*. W tym ostatnim przykładzie atrybutami obiektu mogą być : nazwa drużyn, ilość strzelonych goli, nazwiska trenerów, itp.

Obiekty i klasy

W języku C/C++ istnieje typ danych zwany ***strukturą,*** który umożliwia łączenie różnych typów danych. Struktura stanowi jeden ze sposobów zastosowania abstrakcji w programach. Np. struktura **pracownik** może mieć postać:

```
struct pracownik  {
        char nazwisko[MAXN];
        char imie{MAXI}
        int rok_urodzenia;
};
```

Deklaracja ta opisuje strukturę złożoną z dwóch tablic i jednej zmiennej typu **int.** Nie tworzy ona rzeczywistego obiektu w pamięci, a jedynie określa, z czego składa się taki obiekt. Opcjonalna etykieta (znacznik, ang. ***structure tag***) **pracownik** jest nazwą przyporządkowaną strukturze. Ta nazwa jest wykorzystywana przy deklaracji zmiennej:

```
struct pracownik kierownik;
```

Deklaracja ta stwierdza, że **kierownik** jest zmienną strukturalną o budowie **pracownik**. Nazwy zdefiniowane wewnątrz nawiasów klamrowych definicji struktury są ***składowymi struktury*** (elementami struktury, polami struktury). Definicja struktury pracownik zawiera trzy składowe, dwie typu **char** – **nazwisko** i **imie** oraz jedna typu **int** – **rok_urodzenia**.

Obiekty i klasy

Dostęp do składowych struktur jest możliwy dzięki ***operatorom dostępu*** do składowych. Mamy dwa takie operatory: operator kropki (.) oraz operator strzałki (->). Za pomocą operatora kropki możliwy jest dostęp do składowych struktury realizowany przez nazwę lub referencję do obiektu. Np. aby wydrukować składową **rok_urodzenia** możemy posłużyć się wyrażeniem:

```
cout << kierownik.rok_urodzenia;
```

Składowe tej samej struktury muszą mieć różne nazwy, jednak dwie różne struktury mogą zawierać składowe o tej samej nazwie. Składowe struktury mogą być dowolnego typu. Struktury danych umożliwiają przechowywanie cech (właściwości, atrybutów) obiektów. Z obiektem związane są również najróżniejsze działania. Działania te tworzą ***interfejs*** obiektu. W języku C nie ma możliwości umieszczenia w strukturze atrybutów obiektu i operacji, jakich można na obiekcie wykonać.
W języku C++ wprowadzono nowy ***abstrakcyjny typ danych***, który umożliwia przechowywanie atrybutów i operacji. Tworzenie abstrakcyjnych typów danych odbywa się w języku C++ (tak samo jak w innych językach programowania obiektowego) za pomocą **klas**.

Klasa stanowi implementację abstrakcyjnego typu danych.

Obiekty i klasy

Na każdy projektowany obiekt składają się dane (atrybuty) i dobrze określone operacje (działania). Do danych (gdy są chronione) nie można dotrzeć bezpośrednio, należy do tego celu wywołać odpowiednią **metodę**. Dzięki temu chronimy dane przed niepowołanym dostępem. Komunikacja użytkownika z obiektem (a także komunikacja wybranego obiektu z innym obiektem) zaczyna się od wysłania do niego odpowiedniego żądania (ang. *request*). Po odebraniu żądania (inaczej komunikatu) obiekt reaguje wywołaniem odpowiedniej metody lub wysyła komunikat, że nie może żądania obsłużyć.

Klasa opisuje **obiekt**. Z formalnego punktu widzenia klasa stanowi typ.
W programie możemy tworzyć zmienne typu określonego przez klasę. Zmienne te nazywamy ***instancjami*** (wcieleniami). Instancje stanowią realizację obiektów opisanych przez klasę.

Jak już wiemy, operacje wykonywane na obiektach noszą nazwę metod.
Aby wykonać konkretną operację, obiekt musi otrzymać komunikat, który przetwarzany jest przez odpowiednią metodę. Cechą charakterystyczną programów obiektowych jest przesyłanie komunikatów pomiędzy obiektami.

Obiekty i klasy, terminologia

Ponieważ język C++ jest w istocie językiem hybrydowym, panuje pewne zamieszanie w stosowanej terminologii.

Metody noszą nazwę ***funkcji*** (procedury i podprogramy w języku C++ noszą nazwy funkcji), a ***instancje*** nazywana są ***obiektami konkretnymi***. W specyfikacji języka C++ pojęcie instancji nie jest w ogóle używane.

Dla określania ***instancji*** klasy używa się po prostu terminu ***obiekt***.

Istnieje duża różnorodność, jeżeli chodzi o terminologię stosowaną w opisie elementów klasy. Mamy takie terminy jak: ***elementy, składowe, atrybuty, dane***.

Operacje nazywane są ***metodami, funkcjami składowymi*** lub po prostu ***funkcjami***.

W języku C++ klasa jest ***strukturą,*** której składowe mogą także być funkcjami (metodami).

Deklaracja klasy

Deklaracja klasy precyzuje, jakie dane i funkcje publiczne są z nią związane, czyli do których ma dostęp użytkownik klasy. Deklaracja klasy **punkt** może mieć postać:

```
class punkt
{
   private:
      int x;
      int y;
   public:
      void init (int, int) ;
      void przesun (int, int)
};
```

Do deklarowania klasy służy słowo kluczowe **class**. Po nim podajemy nazwę tworzonej klasy, a następnie w nawiasach klamrowych umieszczamy zmienne wewnętrzne (dane) i metody (funkcje składowe). Deklarację kończy się średnikiem. W tym przykładzie klasa **punkt** zawiera dwie prywatne dane składowe **x** i **y** oraz dwie publiczne funkcje składowe **init()** i **przesun()**. Deklaracja tej klasy nie rezerwuje pamięci na nią. Mówi ona kompilatorowi, co to jest **punkt**, jakie dane zawiera i co może wykonać.

Obiekty

Obiekt nowego typu definiuje się tak, jak każdą inną zmienną, np. typu **int**:

```
int radian;          //definicja radian
punkt srodek;        //definicja punktu
```

W tym przykładzie definiujemy zmienną o nazwie **radian** typu **int** oraz **srodek** , który jest typu **punkt**.

Definicja klasy składa się z definicji wszystkich funkcji składowych. Definiując funkcje składowe podajemy nazwę klasy bazowej przy użyciu operatora zakresu(**::**).

Definicja funkcji składowej **init()** może mieć postać:

```
void punkt :: init (int xp, int yp)
    {
    x = xp;
    y = yp;
     }
```

W tej definicji **x** i **y** reprezentują dane składowe **x** i **y** obiektu klasy **punkt**.

Obiekty

Aby skorzystać z klasy **punkt**, powinniśmy zadeklarować obiekty klasy punkt, np. :

```
punkt p1, p2;
```

Zostały utworzone dwa obiekty klasy **punkt**. Dostęp do publicznej funkcji składowej **init()** uzyskujemy przy pomocy operatora kropki:

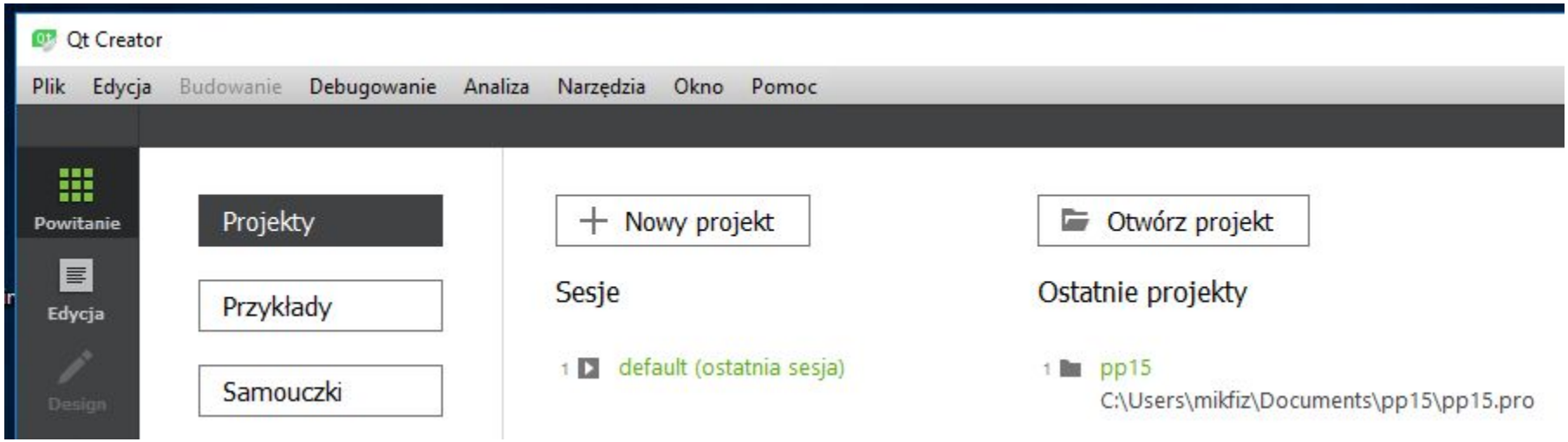
```
p1.init( 10,10 );
```

Została wywołana publiczna funkcja składowa **init()** klasy, do której należy obiekt **p1**, to znaczy do klasy **punkt**.

Powyższe rozważania zilustrujemy realnym programem komputerowym. Rekomendujemy pakiet Qt (aplikacja C++).

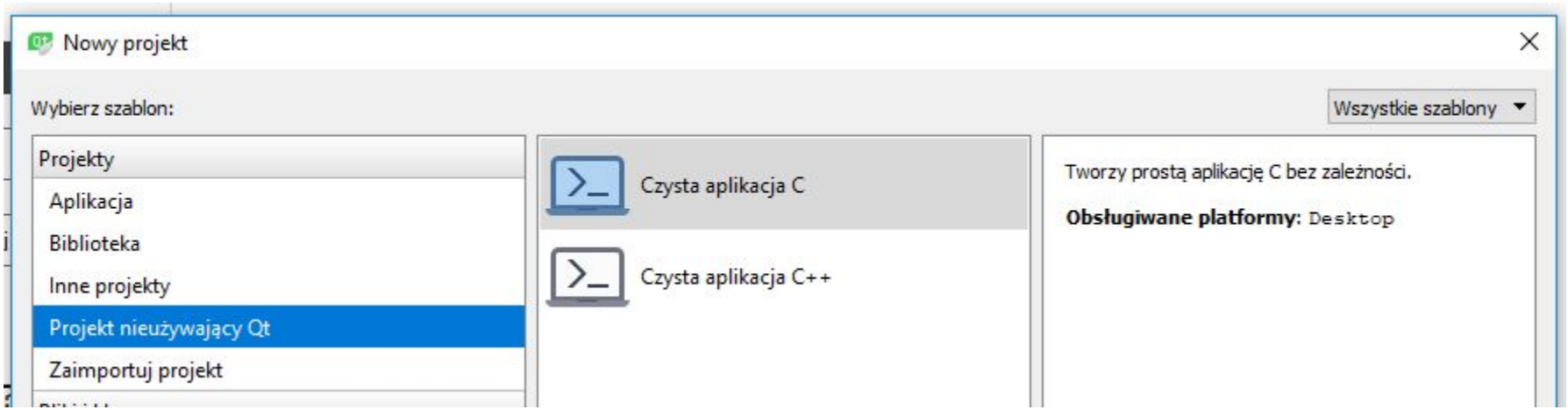
Pakiet Qt

Po prawidłowo zainstalowanym pakiecie Qt, dobrze jest umieścić na ekranie ikonę Kreatora Qt, dzięki któremu mamy szybki dostęp do edytora.

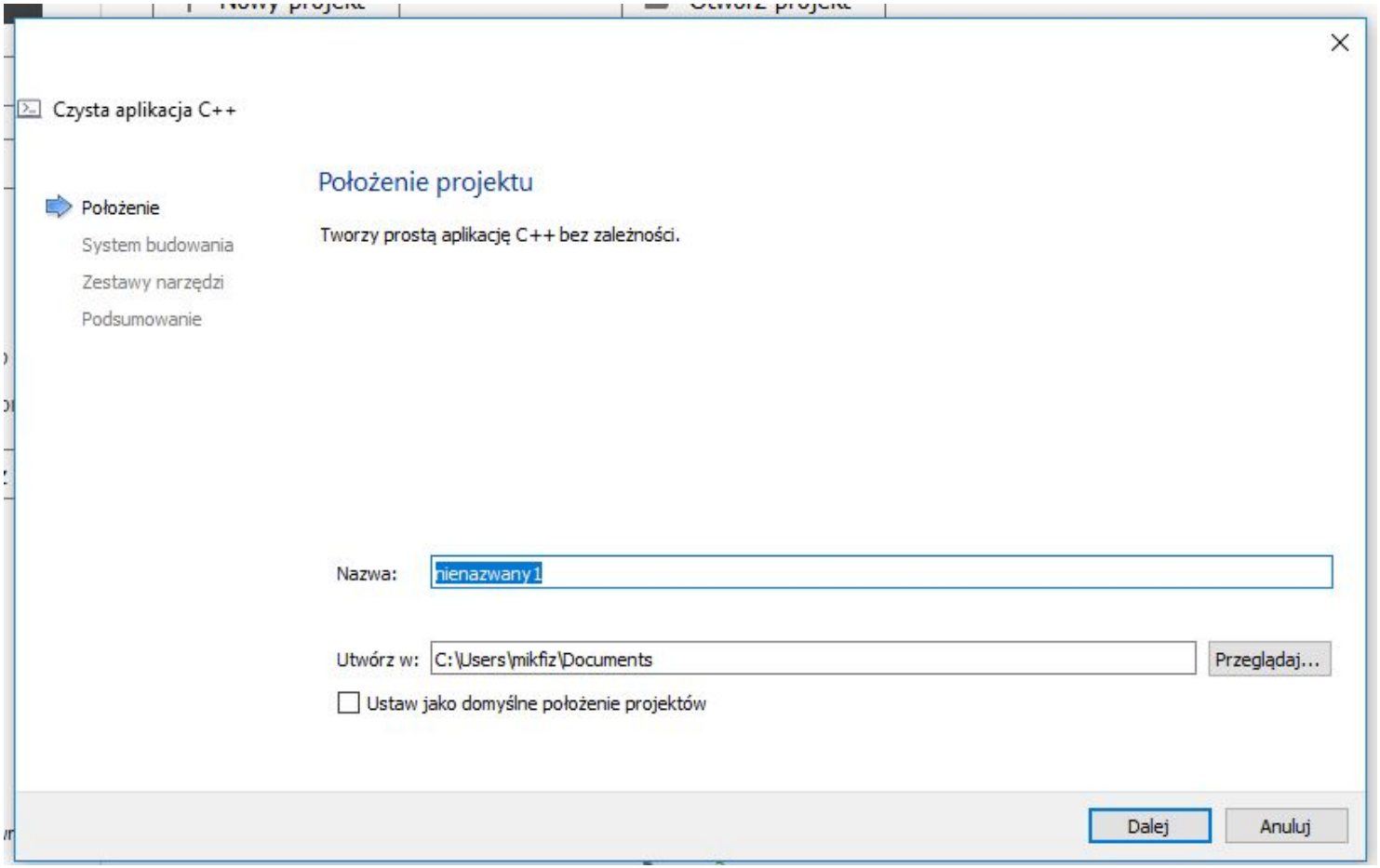


Pakiet Qt

Wybieramy opcję „Nowy projekt”, „Czysta aplikacja C++”

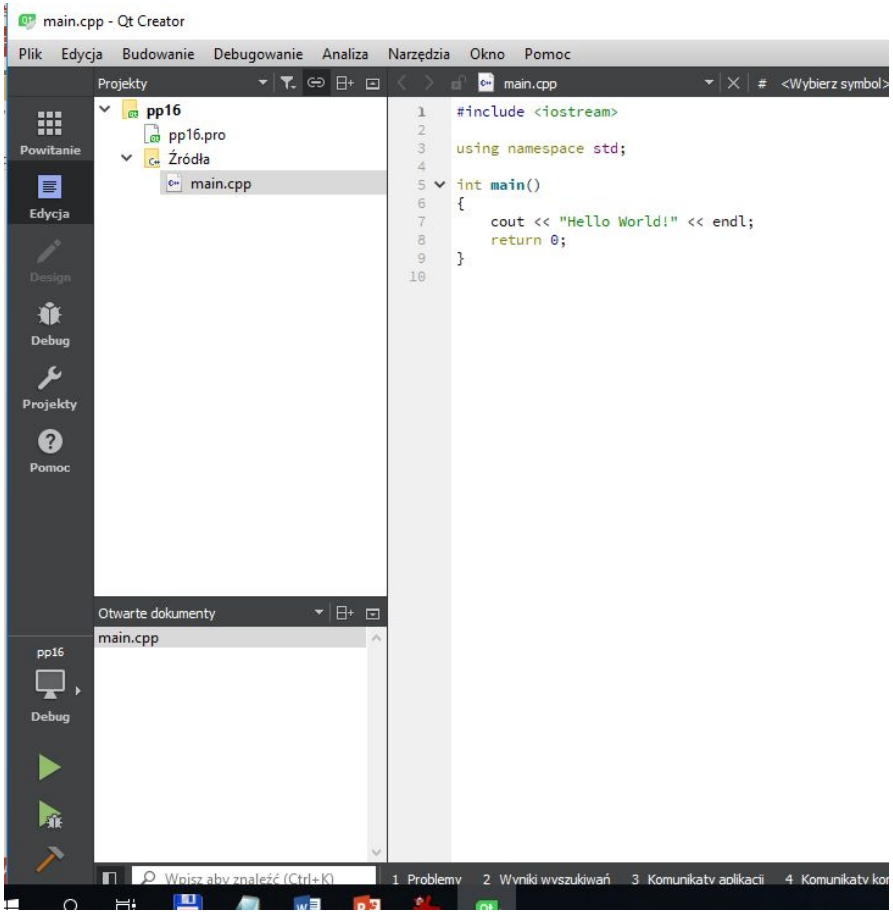


Wypełniamy kolejne formularze



Pakiet Qt

Pokazuje się edytor Qt, piszemy nasz plik źródłowy



Inne kompilatory

Możemy korzystać z innych kompilatorów.

Mam dwie opcje:

- inne IDE (np. Dev C++, Code :: Block)
- Kompilatory online

online C++ compiler

Można korzystać z kompilatorów online.

W sieci mamy ich wiele kompilatorow online.

Przykład:

https://www.jdoodle.com/online-compiler-c++/

Wspaniałą rzeczą w kompilatorze C ++ online jest to, że nie musimy ich pobierać.
Aby z niego skorzystać, potrzebujemy tylko obsługiwanej przeglądarki i aktywnego połączenia internetowego. Zamiast pobierać różne narzędzia programistyczne, środowiska C ++ IDE itd., po prostu odwiedzamy te witryny i używamy kompilatorów online do natychmiastowego uruchamiania naszego kodu.

W serwisie:

(**https://www.techgeekbuzz.com/best-online-cpp-compiler/**

mamy ranking najlepszych kompilatorow (na rok 2019)

online C++ compiler

1.**Jdoodle** provides an interesting and fun-to-use online C++ compiler with an astonishing execution speed. It also supports compiler and interpreter for different programming languages.

2. **HackerEarth** provides an easy to use online C++ compiler with the version of C++ (GCC 5.4.0). The output of the code displays on the same tab after you hit the compile and run button. It provides a multitude of features, including account login, coding color, debugging tools, auto-based, auto-suggestion, and auto-close brackets.

3. **Repl.it** gives the most beautiful and one of the best online compilers and IDEs for C++. It supports the latest version of C++ with some exciting functions. Using the Repl.it online C++ compiler, you can create projects as well as separate module files. It provides features like code coloring, screen customization, debugging, and auto-suggestion.

4. **Ideone** provides an online C++ compiler alongside a debugging tool. It has more than 1.4 million visitors per month. Features like download code, login, and color-coding are available with this online C++ compiler.

5. **Tutorialspoint** is one of the most reputed websites among computer geeks. It provides online compilers for different programming languages, including C++. If you have a stable internet connection then Tutorialspoint online C++ compiler will execute your code in no time.

6. **CodeChef** provides a fast and easy-to-use online C++ compiler. Apart from C++, you can compile your Java code and interpret your Python code too. It also provides many other interesting topics that help students to learn new things in programming.

7. **OnlineGBD** provides an interesting and great user interface for online C++ compilation. The online C++ compiler provides you with many great features, like color coding, auto-suggestion, auto-close brackets, save source code, and debug code.

8. **Rextester** provides a basic online C++compiler. It supports 3 different versions of C++:

- Clang,
- GCC, and
- vc++.

9. **Codepad** comes with the upgraded and latest C++ compiler. It is though a very basic online compiler. Its compiler user interface seems modest at best. Like other online compilers, Codepad provides support for compilers and interpreters for different programming languages. The cool thing about Codepad is that it works great on even mobile devices.

10. **C++ shell**, as suggested by the name, provides an online shell or IDE interface to compile the C++ code. The problem, however, is that this website often seems to hang when we compile or execute the C++ code.

Dev-C++

Dev-C++ – zintegrowane środowisko programistyczne – zintegrowane środowisko programistyczne, obsługujące języki C – zintegrowane środowisko programistyczne, obsługujące języki C i C++ – zintegrowane środowisko programistyczne, obsługujące języki C i C++, na licencji GPL, dla systemów rodziny Windows i Linux.

Jest zintegrowany z MinGWJest zintegrowany z MinGW, czyli z windowsowym portemJest zintegrowany z MinGW, czyli z windowsowym portem kompilatoraJest zintegrowany z MinGW, czyli z windowsowym portem kompilatora GCC.

Program od 2005 roku nie jest rozwijany.

Na bazie Dev-C++ jest rozwijany nowy program – wxDev-C++.

Nieoficjalne wersje

Od czerwca 2011 holenderski programista o pseudonimie Orwell wydawał kolejne wersje Dev-C++ pod nazwą Orwell Dev-C++.

Ostatnią wersję, 5.11, wydał w kwietniu 2015. Od tamtego czasu programista nie daje żadnych znaków aktywności.

Do nauki programowania obiektowego w C++ jest to wystarczające narzędzie IDE. Jego zaletą jest niewielki rozmiar i mało skomplikowane kopiowanie na nasz dysk.

Code::Blocks

- **Code::Blocks** – jest wieloplatformowe – jest wieloplatformowe, zintegrowane środowisko programistyczne – jest wieloplatformowe, zintegrowane środowisko programistyczne (IDE) na licencji GNU – jest wieloplatformowe, zintegrowane środowisko programistyczne (IDE) na licencji GNU, oparte na projekcie Scintilla – jest wieloplatformowe, zintegrowane środowisko programistyczne (IDE) na licencji GNU, oparte na projekcie Scintilla. Wspiera języki C – jest wieloplatformowe, zintegrowane środowisko programistyczne (IDE) na licencji GNU, oparte na projekcie Scintilla. Wspiera języki C, C++ – jest wieloplatformowe, zintegrowane środowisko programistyczne (IDE) na licencji GNU, oparte na projekcie Scintilla. Wspiera języki C, C++ oraz Fortran (od wersji 13.12).
- Program jest napisany w C++ z wykorzystaniem wieloplatformowej bibliotekiProgram jest napisany w C++ z wykorzystaniem wieloplatformowej biblioteki wxWidgetsProgram jest napisany w C++ z wykorzystaniem wieloplatformowej biblioteki wxWidgets. Dzięki temu działa zarówno na systemach operacyjnychProgram jest napisany w C++ z wykorzystaniem wieloplatformowej biblioteki wxWidgets. Dzięki temu działa zarówno na systemach operacyjnych LinuxProgram jest napisany w C++ z wykorzystaniem wieloplatformowej biblioteki wxWidgets. Dzięki temu działa zarówno na systemach operacyjnych Linux i WindowsProgram jest napisany w C++ z wykorzystaniem wieloplatformowej biblioteki wxWidgets. Dzięki temu działa zarówno na systemach operacyjnych Linux i Windows, jak również MacOS XProgram jest napisany w C++ z wykorzystaniem wieloplatformowej biblioteki wxWidgets. Dzięki temu działa zarówno na systemach operacyjnych Linux i Windows, jak również MacOS X. Przechowywany jest na witrynach BerliOSProgram jest napisany w C++ z wykorzystaniem wieloplatformowej biblioteki wxWidgets. Dzięki temu działa zarówno na systemach operacyjnych Linux i

Programowanie obiektowe

Podstawowe elementy programowania obiektowego

hermetyzacja

Typowy program obiektowy (jednoplikowy) zawiera następujące bloki:

- Dyrektywy preprocesora (najczęściej #include< >)
- Określenie przestrzeni nazw i zmienne globalne
- opis klasy
- Definicje metod
- Funkcję main()

Obiekty

```
#include <iostream>                 // cout
using namespace std;                // przestrzeń nazw
class punkt                         // definicja klasy punkt
   { private:
        int x;
        int y;
     public:
        void init (int, int) ;
        void print();
   };
void punkt::init( int xx, int yy)    //definicja metody
    {x =xx;  y = yy;}
void punkt::print()                  //definicja metody
    {cout <<"x = "<<x<<"   y = "<<y<<endl;}
int main()                          //wykonanie, funkcja main()
{ punkt a;
   a.init(2,4);
   a.print();
  return 0;
}
```

Wynik:

x = 2 y = 4

Hermetyzacja danych

Hermetyzacja (ang. ***encapsulation***) oznacza połączenie danych i instrukcji programu w jednostkę programową, jakim jest obiekt. Hermetyzacja obejmuje interfejs i definicję klasy. Podstawową zaletą hermetyzacji jest możliwość zabezpieczenia danych przed równoczesnym dostępem ze strony różnych fragmentów kodu programowego. W tym celu wszystkie dane (**pola** w obiekcie, atrybuty**)** i zapisy instrukcji (funkcje składowe) dzieli się na ogólnodostępne (interfejs obiektowy) i wewnętrzne (implementacja obiektu). Dostęp do pól i metod wewnętrznych jest możliwy tylko za pośrednictwem "łącza obiektowego" - pól i metod ogólnodostępnych. Wybrane pola i metody można więc ukryć przed określonymi obiektami zewnętrznymi. ***Hermetyzacja umożliwia separację interfejsu od implementacji klasy***. W klasycznej strukturze danych w języku C mamy swobodny dostęp do składowych struktury. Oznacza to, że z dowolnego miejsca w programie możemy dokonać zmiany tych danych. *Nie jest to dobra cecha.* Zastosowanie takiego modelu dostępu do danych obiektu grozi wystąpieniem niespójności danych, co prowadzi zwykle do generacji błędów.

W języku C++ rozwiązano problem panowania nad dostępem do danych przy pomocy ***hermetyzacji*** danych. W literaturze przedmiotu spotkamy się z zamiennie stosowanymi terminami takimi jak: ***ukrywanie danych, kapsułkowanie*** czy zgoła całkiem egzotycznym terminem ***enkapsulacja***.

Hermetyzacja danych

W celu uniemożliwienia programistom wykonywania niekontrolowanych operacji na danych obiektu ***silnie ograniczono dostęp*** do jego składowych za pomocą dobrze zdefiniowanego interfejsu. Programista może na obiekcie wykonać tylko te operacja, na które ***pozwolił*** mu projektant klasy. W języku C++ dostęp do składowych klasy kontrolowany jest przez trzy specyfikatory:

private

public

protected

Składowe zadeklarowane po słowie kluczowym **private** nie są dostępne dla programisty aplikacji. Posiada on dostęp do składowych oraz może wywoływać funkcje składowe zadeklarowane po słowie kluczowym **public**. Dostęp do atrybutów obiektu z reguły jest możliwy za pośrednictwem funkcji. Jeżeli definiując klasę pominiemy specyfikator dostępu, to żadna składowa klasy nie będzie dostępna na zewnątrz obiektu, domyślnie przyjmowany jest sposób dostępu określony specyfikatorem **private**. Hermetyzacja ma ogromne znaczenie dla przenośności programów i optymalizowania nakładów potrzebnych na ich modyfikacje.

hermetyzacja

```
#include <iostream>
using namespace std;
const double PI = 3.1415926536;
class kolo
{  public:
      double r;
      void set(double rr) {
              r = rr;
             }
};

int main()
{  kolo k1;
    k1.set(1.0);
    cout <<"promien = " << k1.r << endl;
    cout <<"pole = " << PI*k1.r*k1.r <<endl;
    return 0;
}
```

Obliczamy pole koła o zadanym promieniu
Dana r jest publiczna.

Wynik:

promien = 1
pole = 3.14159

hermetyzacja

```
#include <iostream>
using namespace std;
const double PI = 3.1415926536;
class kolo
{   double r;                          //r jest prywatne!
  public:
    void set(double rr) {
           r = rr;
         }
};
int main()
{  kolo k1;
    k1.set(1.0);
    cout <<"promien = " << k1.r << endl;
    cout <<"pole = " << PI*k1.r*k1.r <<endl;
    return 0;
}
```

Program nie wykonuje się!
Kompilator generuje komunikat:

```
C:\Users\mikfiz\Documents\pp17\main.cpp:5: błąd: 'double kolo::r' is private
 {   double r;
```

hermetyzacja

```
#include <iostream>
using namespace std;
const double PI = 3.1415926536;
class kolo
{   double r;                          //r jest prywatne!
  public:
    void set(double rr) {
           r = rr;
         };
    double get_r(){return r;}        //funkcja dostepu
};

int main()
{  kolo k1;
    k1.set(2.0);
    cout <<"promien = " << k1.get_r() << endl;
    cout <<"pole = " << PI*k1.get_r()*k1.get_r() <<endl;
    return 0;
}
```

Wynik:
promien = 2
pole = 12.5664

Program wykonuje się, ponieważ w definicji klasy umieszczono funkcję dostępu (tzw. akcesor)!

Dziedziczenie

Jedna z najistotniejszych cech programowania zorientowanego obiektowo jest **dziedziczenie** (ang. ***inheritance***).

Mechanizm dziedziczenia służy w językach obiektowych do odwzorowania występujących często w naturze powiązań typu ***generalizacja - specjalizacja.*** Umożliwia programiście definiowanie potomków istniejących obiektów. ***Każdy potomek dziedziczy przy tym pola i metody obiektu bazowego, lecz dodatkowo uzyskuje pewne pola i własności unikatowe, nadające mu nowy charakter.***

Typ takiego obiektu potomnego może stać się z kolei typem bazowym do zdefiniowania kolejnego typu potomnego.

Bjarne Stroustrup w podręczniku „Język C++” rozważając zależności występujące pomiędzy klasą **figura** i klasą **okrąg**, tak zdefiniował paradygmat programowania obiektowego:

zdecyduj, jakie chcesz mieć klasy;

dla każdej klasy dostarcz pełny zbiór operacji;

korzystając z mechanizmu dziedziczenia, jawnie wskaż to, co jest wspólne

Dziedziczenie

W praktyce okazało się, że wiele klas ma zazwyczaj kilka wspólnych cech. Naturalnym jest więc dążenie, aby analizując podobne klasy wyodrębnić wszystkie wspólne cechy i stworzyć uogólnioną klasę, która będzie zwierać tylko te atrybuty i metody, które są wspólne dla rozważanych klas.

W językach obiektowych taką uogólnioną klasę nazywamy ***superklasą*** lub ***klasą rodzicielską***, a każda z analizowanych klas nazywa się ***podklasą*** lub ***klasą potomną***.

W języku C++ wprowadzono koncepcje ***klasy bazowej*** (ang. ***base class***) i ***klasy pochodnej*** (ang. ***derived class***). Klasa bazowa (***klasa podstawowa***) zawiera tylko te elementy składowe, które są wspólne dla wyprowadzonych z niej klas pochodnych. Podczas tworzenia nowej klasy, zamiast pisania całkowicie nowych danych oraz metod, programista może określić, że nowa klas odziedziczy je z pewnej, uprzednio zdefiniowanej klasy podstawowej. Każda taka klasa może w przyszłości stać się klasą podstawową.

W przypadku ***dziedziczenia jednokrotnego*** klasa tworzona jest na podstawie ***jednej*** klasy podstawowej. W sytuacji, gdy nowa klasa tworzona jest w oparciu o ***wiele*** klas podstawowych mówimy o ***dziedziczeniu wielokrotnym***.

Dziedziczenie

W języku C++ projektując nową klasę pochodną mamy następujące możliwości:

- ***w klasie pochodnej można dodawać nowe zmienne i metody***
- ***w klasie pochodnej można redefiniować metody klasy bazowej***

W języku C++ możliwe są trzy rodzaje dziedziczenia: publiczne, chronione oraz prywatne. Klasy pochodne nie mają dostępu do tych składowych klasy podstawowej, które zostały zadeklarowane jako **private**. Oczywiście klasa pochodna może się posługiwać tymi składowymi, które zostały zadeklarowane jako **public** lub **protected**. Jeżeli chcemy, aby jakieś składowe klasy podstawowej były ***niedostępne*** dla klasy pochodnej, to deklarujemy je jako **private.** Z takich prywatnych danych, klasa pochodna może korzystać wyłącznie za pomocą ***funkcji dostępu*** znajdujących się w ***publicznym*** oraz zabezpieczonym interfejsie klasy podstawowej.

Mechanizm dziedziczenia obiektów ma znaczenie dla optymalizowania nakładów pracy potrzebnych na powstanie programu (optymalizacja kodu, możliwość zrównoleglenia prac nad fragmentami kodu programowego) i jego późniejsze modyfikacje (przeprowadzenie zmian w klasie obiektów wymaga przeprogramowania samego obiektu bazowego).

Dziedziczenie

```
#include <iostream>
using namespace std;
class Kot
{ public:
    int wiek;
    int waga;
    string glos="miauczy";
};
class Pies
{ public:
    int wiek;
    int waga;
    string glos="szczeka";
};
int main()
{  Kot Mruczek;
   Pies Burek;
   Mruczek.wiek = 5 ;
   Burek.wiek = 7;
    cout <<"Kot Mruczek ma  " << Mruczek.wiek <<" lat" <<" i "<<Mruczek.glos<< endl;
    cout <<"Pies Burek ma   " << Burek.wiek <<" lat" <<" i "<<Burek.glos<< endl;
    return 0;
}
```

Wynik:
Kot Mruczek ma 5 lat i miauczy
Pies Burek ma   7 lat i szczeka

Dziedziczenie

```
#include <iostream>
using namespace std;
class Animal                //klasa bazowa
{ public:
    int wiek;
    int waga;
    };
class Kot : public Animal   //dziedziczenie, klasa potomna
{ public:
    string glos = "miauczy";
};
class Pies : public Animal  //dziedziczenie, klasa potomna
{ public:
    string glos = "szczeka";
};
int main()
{  Kot Mruczek;
   Pies Burek;
   Mruczek.wiek = 5 ;
   Burek.wiek = 7;
    cout <<"Kot Mruczek ma  " << Mruczek.wiek <<" lat" <<" i "<<Mruczek.glos<< endl;
    cout <<"Pies Burek ma   " << Burek.wiek <<" lat" <<" i "<<Burek.glos<< endl;
    return 0;
}
```

Wynik:

Kot Mruczek ma 5 lat i miauczy
Pies Burek ma 7 lat i szczeka

Polimorfizm

Drugą istotną cechą (po dziedziczeniu) programowania zorientowanego obiektowo jest ***polimorfizm*** (ang. ***polimorphism***). Słowo polimorfizm oznacza dosłownie ***wiele form***.

Polimorfizm, stanowiący uzupełnienie dziedziczenia sprawia, że możliwe jest pisanie kodu, który w przyszłości będzie wykorzystywany w warunkach nie dających się jeszcze przewidzieć. Mechanizm polimorfizmu wykorzystuje się też do realizacji pewnych metod w trybie nakazowym, abstrahującym od szczegółowego typu obiektu. Zachowanie polimorficzne obiektu zależy od jego pozycji w hierarchii dziedziczenia. ***Jeśli dwa lub więcej obiektów maja ten sam interfejs, ale zachowują się w odmienny sposób, są polimorficzne.*** Jest to bardzo istotna cecha języków obiektowych, gdyż pozwala na zróżnicowanie działania tej samej funkcji w zależności od rodzaju obiektu. Zagadnienie polimorfizmu jest trudne pojęciowo, w klasycznych podręcznikach programowania jest omawiane zazwyczaj razem z ***funkcjami wirtualnymi*** (ang. ***virtual function***). Przy zastosowaniu funkcji wirtualnych i polimorfizmu możliwe jest zaprojektowanie aplikacji, która może być w przyszłości prosto rozbudowywana.

Polimorfizm

Jako przykład rozważmy zbiór klas modelujących figury geometryczne takie jak koło, trójkąt, prostokąt, kwadrat, itp. Wszystkie są ***klasami pochodnymi*** klasy bazowej **figura**. Każda z klas pochodnych ma możliwość narysowania siebie, dzięki metodzie **rysuj**. Ponieważ mamy różne figury, funkcja rysuj jest inna w każdej klasie pochodnej. Z drugiej strony, ponieważ mamy klasę bazową **figura**, to dobrze by było, aby niezależnie, jaką figurę rysujemy, powinniśmy mieć możliwość wywołania metody **rysuj** klasy bazowej **figura** i pozwolić programowi ***dynamicznie określić, którą z funkcji rysuj z klas pochodnych ma zastosować.***

Język C++ dostarcza narzędzi umożliwiających stosowanie takiej koncepcji. W tym celu należy zadeklarować metodę **rysuj** w klasie bazowej, jako funkcję wirtualną.

Polimorfizm

Funkcja wirtualna może mieć następującą deklarację:

virtual void rysuj ( ) const ;

i powinna być umieszczona w klasie bazowej **figura**.

Powyższy prototyp deklaruje funkcję **rysuj** jako stałą, nie zawierającą argumentów, nie zwracającą żadnej wartości i wirtualną.

Jeżeli funkcja **rysuj** została zadeklarowana w klasie bazowej jako wirtualna, to gdy następnie zastosujemy ***wskaźnik*** w klasie bazowej lub ***referencję*** do obiektu w ***klasie pochodnej*** i wywołamy funkcję **rysuj** stosując ten wskaźnik , to program powinien wybrać dynamicznie właściwą funkcję **rysuj**.

Polimorfizm umożliwia tworzenie w typach potomnych tzw. **metod wirtualnych**, nazywających się identycznie jak w typach bazowych, lecz ***różniących*** się od swych odpowiedników pod względem znaczeniowym.

Polimorfizm

Zwykłe przeciążanie funkcji

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

void oblicz(int x, int y) {cout<<"pole = "<<x*y<<endl; }
void oblicz(int x, int y, int z){cout<<"objetosc = "<<x*y*z<<endl; }

int main()
{ int x = 1;
  int y = 2;
  int z = 3;
  oblicz(x,y);  //funkcja globalna przeciazona
  oblicz(x,y,z); //funkcja globalna przeciazona

  return 0;
}
```

Wynik:
pole = 2
objetosc = 6

Polimorfizm

```
#include <iostream>
using namespace std;
class figura              //klasa bazowa
{ public:
    double x;
    double y;
};
class trojkat : public figura   //dziedziczenie, klasa potomna
{ public:
    void oblicz_pole() {cout <<"pole trojkata   = "<<0.5*x*y<<endl;
    }
};
class prostokat : public figura  //dziedziczenie, klasa potomna
{ public:
    void oblicz_pole() {cout <<"pole prostokata = "<<x*y<<endl;
    }
};
int main()
{  trojkat tr;
   prostokat pr;
   tr.x = 1.0;
   tr.y = 2.0;
   pr.x = 1.0;
   pr.y = 2.0;
   tr.oblicz_pole();  //nadpisanie metody
   pr.oblicz_pole();  //nadpisanie metody
  return 0;
}
```

Nadpisanie metody

Wynik:
pole trojkata   = 1
pole prostokata = 2

Polimorfizm

```
#include <iostream>
using namespace std;
class Animal              //klasa bazowa
{ public:
    virtual void typ() = 0;  //metoda virtualna
    virtual void opis() = 0; //metoda virtualna
};
class Kot : public Animal   //dziedziczenie, klasa potomna
{public:
     void typ(){cout <<"kot ";};
     void opis() {cout <<"miauczy"<<endl;};
};
class Pies : public Animal  //dziedziczenie, klasa potomna
{ public:
     void typ(){cout <<"pies ";};
     void opis() {cout <<"warczy"<<endl;};
};

int main()
{  Kot Mruczek;
   Pies Burek;
   Mruczek.typ();
   Mruczek.opis();
   Burek.typ();
   Burek.opis();
 return 0;
}
```

Metody czysto wirtualne

Wynik:

kot miauczy
pies warczy

Podsumowanie terminologii

W powyższych rozważaniach wprowadziliśmy dużo nowych terminów związanych z programowaniem zorientowanym obiektowo. Wydaje się celowe sporządzenie krótkiego spisu nowych terminów z krótkimi objaśnieniami.

Atrybuty.

Są to dane klasy, inaczej składowe klasy, które opisują bieżący stan obiektu. Atrybuty powinny być ukryte przed użytkownikami obiektu, a dostęp do nich powinien być określony i zdefiniowany w interfejsie.

Obiekt.

Obiekt jest bytem istniejącym i może być opisany. Obiekt charakteryzują atrybuty i operacje. Obiekt jest instancją klasy (egzemplarzem klasy).

Klasa.

Klasa jest formalnie w języku C++ typem. Określa ona cechy i zachowanie obiektu. Klasa jest jedynie opisem obiektu. Należy traktować klasę jako szablon do tworzenia obiektów.

Zadanie 1

1. Utworzyć klasę Kot z ***publicznymi danymi***:
   1. imie
   2. waga
   3. wiek

   Dane wprowadzić z klawiatury. Przetestować klasę w funkcji main().

Test może mieć postać:

```
Podaj imie kota: Mruczek
Podaj wage kota: 6
Podaj wiek kota: 8
nasz kot nazywa sie Mruczek ma 8 lat  i wazy 6 kg
```

Zadanie 2

1. Utworzyć klasę Kot z ***prywatnymi danymi***:
    1. imie
    2. waga

    Przetestować klasę w funkcji main().

Zadanie 3

1. Utworzyć klasę Kot z ***prywatnymi danymi***:
   1. imie
   2. waga
   3. W klasie umieść metodę ***drukuj(),*** dzięki której wyświetlimy imię i wagę kota

   Przetestować klasę w funkcji main().

KONIEC