MINISTERSTWO EDUKACJI
i NAUKI

Andrzej Krawczyk

Programowanie w środowisku języka obiektowego 312[01].Z2.02

Poradnik dla ucznia

Wydawca
Instytut Technologii Eksploatacji – Państwowy Instytut Badawczy
Radom 2005

„Projekt współfinansowany ze środków Europejskiego Funduszu Społecznego”

Recenzenci:
mgr inż. Ireneusz Przybyłowicz
mgr inż. Andrzej Uzar

Opracowanie redakcyjne:
mgr inż. Katarzyna Maćkowska

Konsultacja:
dr inż. Bożena Zając

Korekta:
mgr inż. Tomasz Sułkowski

Poradnik stanowi obudowę dydaktyczną programu jednostki modułowej 312[01].Z2.02 Programowanie w środowisku języka obiektowego zawartego w modułowym programie nauczania dla zawodu technik informatyk.

Wydawca
Instytut Technologii Eksploatacji – Państwowy Instytut Badawczy , Radom 2005

SPIS TREŚCI

1. WPROWADZENIE

Poradnik będzie Ci pomocny w kształtowaniu umiejętności:
- obiektowego projektowania rozwiązywania zadań,
- tworzenia programów przy użyciu języka obiektowego,
- korzystania z funkcji języka obiektowego,
- optymalnego zarządzania pamięcią,
- korzystania ze struktur dynamicznych.

W poradniku zamieszczono:
- wykaz umiejętności, jakie powinieneś posiadać, by bez problemów korzystać z poradnika,
- cele kształcenia, które pozwolą Ci uświadomić sobie cel informacji i ćwiczeń zawartych w poradniku,
- materiał nauczania, czyli skondensowane wiadomości teoretyczne oraz wskazówki, które pomogą Ci zrozumieć i poprawnie wykonać zestaw ćwiczeń,
- zestaw pytań, który pozwoli Ci sprawdzić, czy zapoznałeś się i zrozumiałeś wszystkie wiadomości i możesz przystąpić do wykonania ćwiczeń,
- ćwiczenia, które pozwolą ukształtować praktyczne umiejętności programowania strukturalnego,
- sprawdzian osiągnięć, dzięki któremu ocenisz swoje wiadomości i umiejętności ukształtowane w trakcie pracy z poradnikiem,
- literaturę uzupełniającą.

Pracując z poradnikiem i wykonując ćwiczenia możesz napotkać trudności. W razie wątpliwości zwróć się o pomoc do nauczyciela.

2. WYMAGANIA WSTĘPNE

Przystępując do realizacji programu nauczania jednostki modułowej powinieneś umieć:
- sprawnie posługiwać się aparatem matematycznym,
- zarządzać zasobami na nośnikach stałych,
- modyfikować dokument tekstowy,
- wyszukiwać informacje w różnych źródłach,
- programować korzystając z możliwości języka strukturalnego.

3. CELE KSZTAŁCENIA

Po zakończonym procesie kształcenia tej jednostki modułowej będziesz potrafił:
- zastosować obiektowo zorientowaną analizę problemu,
- określić zasady programowania obiektowego,
- określić podstawowe elementy języka obiektowego,
- analizować działanie programów napisanych w językach wysokiego poziomu,
- posłużyć się stałymi, zmiennymi, wskaźnikami i referencjami,
- zastosować instrukcje, wyrażenia i makrodefinicje,
- zdefiniować i zastosować funkcje, przekazywać argumenty,
- określić pojęcie obiektu, klasy, atrybutu, metody i właściwości,
- wykorzystać dziedziczenie proste i wielokrotne,
- posłużyć się funkcjami: zaprzyjaźnionymi, operatorowymi, wirtualnymi, polimorficznymi,
- zaprojektować i zastosować biblioteki funkcji,
- posłużyć się strumieniami i obiektowo zorientowanymi operacjami,
- posłużyć się plikami do przechowywania danych,
- posłużyć się tablicami, strukturami, uniami, polami bitowymi,
- posłużyć się klasami, hermetyzacją klas, dziedziczeniem metod i atrybutów,
- posłużyć się szablonami klas i funkcji,
- zastosować klasy do obsługi wyjątków i współbieżnej realizacji zadań,
- zdefiniować pojęcie strukturyzacji, hermetyzacji i polimorfizmu,
- zaproponować konstruktor i destruktor obiektu,
- wykorzystać dziedziczenie do definiowania klas obiektów,
- konstruować listy, stosy, kolejki, drzewa i grafy,
- zastosować odpowiednie algorytmy, struktury danych i metody podczas rozwiązywania zadań,
- skompilować, scalić, uruchomić, przetestować, zoptymalizować i udokumentować program,
- skorzystać z podręczników i dokumentacji języków programowania,
- posłużyć się terminologią zawodową w języku angielskim.

4. MATERIAŁ NAUCZANIA

4.1. Definicja i własności obiektów

4.1.1. Materiał nauczania

Język obiektowy jest kolejnym krokiem ewolucji w programowaniu komputerów. Powstał w wyniku rozwoju języków strukturalnych i zmiany podejścia do problemów informatycznych. Aby zrozumieć ideę języków obiektowych należy uzmysłowić sobie czym są obiekty. Cały otaczający świat składa się z obiektów. Takimi obiektami są na przykład: krzesło, stół lub lampa. Posiadają one charakterystyczne dla danej rzeczy właściwości takie jak kolor, wymiar, faktura, funkcje, które mogą spełniać, itd. Dowolne obiekty mogą tworzyć grupy obiektów, na przykład samochód (obiekt, który skupia inne obiekty) składa się z silnika, który umożliwia ruch samochodu, a który też jest obiektem. Dalej mamy skrzynię biegów, kierownicę, siedzenia, hamulce, kierunkowskazy. Każdy z wymienionych elementów jest obiektem składowym samochodu. Każdy z nich posiada swoje parametry. W przypadku silnika może to być moc lub pojemność, w przypadku skrzyni biegów można mówić o skrzyni automatycznej lub manualnej z określoną liczbą biegów. Każdy z takich parametrów można nazwać atrybutem obiektu. W przypadku silnika nie jest najważniejsze to jaką ma pojemność czy moc, ale to co robi, czyli fakt, że wprawia samochód w ruch. Podobnie jest z obiektami, które oprócz cech mają również funkcje. Są zatem zdolne do wykonania działań. Te dwa elementy - atrybuty i działania - charakteryzują obiekty. Podejście obiektowe jest zbieżne ze sposobem rozumowania ludzkiego mózgu, który grupuje i przyporządkowuje rzeczom pewne cechy i elementy wspólne.

Programowanie strukturalne tworzy model składający się z procedur – czynności, które posługują się pewnymi typami danych – obiektami. W wielomodułowych projektach informatycznych jest to model zwykle trudniejszy do zrozumienia niż w przypadku podejścia obiektowego. Tu mamy pewną grupę obiektów (podobnie, jak w rzeczywistości), które wchodzą ze sobą w interakcje. Przykładem takiej interakcji jest silnik i skrzynia biegów. Dwa elementy, które muszą współpracować aby umożliwić poruszanie się samochodu.

Programowanie obiektowe jest więc budowaniem modelu otaczającego nas świata w pamięci komputera, za pomocą obiektów. Zgrupowanie danych i działań w obiekt powoduje, że program staje się przejrzystszy. Wszystko co można zrobić z danym obiektem, jest jego funkcją składową. Jest najlepszym znanym sposobem tworzenia systemów przetwarzających dane. W ostatnich latach zaczęto używać określenia: język w pełni obiektowy. Tego typu językiem jest Java i języki platformy Microsoft **.**NET. Określenie język w pełni obiektowy (zob. „Thinking in Java") oznacza, że każdy element (w pewnym uproszczeniu), każda na pozór zwykła zmienna (np. liczba całkowita typu integer) jest obiektem. To znaczy ma swoje funkcje składowe. w przypadku języka C++ tak nie jest. Zmienna jest tam tylko zmienną, a nie obiektem.

4.1.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.

1. Czym jest obiekt?
2. Jakie są dwie główne składowe obiektu?
3. Co powoduje, że programowanie obiektowe jest bliższe rzeczywistości?
4. Co odróżnia język obiektowy od języka w pełni obiektowego?
5. Jakie znasz języki obiektowe i w pełni obiektowe?

4.1.3. Ćwiczenia

Ćwiczenie 1

Opisz otaczające przedmioty wykorzystując ideę obiektów. Opisz ich charakterystyczne cechy i funkcje jakie pełnią.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:

1) wybierać dowolną grupę przedmiotów. Można kierować się zainteresowaniami. Zapisać w nowym dokumencie tekstowym,
2) dopisać cechy każdego z nich,
3) dopisać funkcje każdego z nich,
4) podzielić się przemyśleniami z innymi uczniami w grupie,
5) uzupełnić notatki na podstawie wypowiedzi innych uczniów.

Wyposażenie stanowiska pracy:

– komputer z zainstalowanym edytorem tekstu.

Ćwiczenie 2

Spróbuj opisać obiektowo pociąg osobowy. Wymień, z jakich elementów i obiektów się składa oraz jakie są właściwości i funkcje tych obiektów.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:

1) zapisać w dokumencie tekstowym, z jakich elementów składa się pociąg,
2) dopisać obok cechy każdego z nich,
3) dopisać obok funkcje każdego z nich,
4) podzielić się przemyśleniami z innymi uczniami w grupie,
5) uzupełnić notatki na podstawie wypowiedzi innych uczniów,

Wyposażenie stanowiska pracy:

– komputer z zainstalowanym edytorem tekstu.

4.1.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) wyjaśnić pojęcie obiektu?	☐	☐
2) określić właściwości rzeczy Ciebie otaczających?	☐	☐
3) określić funkcje rzeczy Ciebie otaczających?	☐	☐
4) określić różnice pomiędzy językiem obiektowym i językiem w pełni obiektowym?	☐	☐

4.2 Pojęcia opisujące obiektowość

4.2.1. Materiał nauczania

W literaturze przedmiotu występują pewne różnice co do tego, jakie cechy i metody języka programowania czynią go "zorientowanym obiektowo". Najważniejsze są następujące atrybuty:

Abstrakcja

Element, obiekt w systemie, jest modelem abstrakcyjnego "wykonawcy", który może wykonywać pracę, opisywać i zmieniać swój stan. Każdy obiekt może komunikować się z innymi obiektami w programie lub w systemie bez ujawniania i zaznaczenia, w jaki sposób zaimplementowano dane cechy. Abstrahowane mogą być obiekty, ale również funkcje, metody czy procesy.

Enkapsulacja

Polega na ukrywaniu implementacji - hermetyzacji. Teoria ta określa, że obiekt nie może zmieniać stanu wewnętrznego innych obiektów w nieoczekiwany sposób. Tylko wewnętrzne metody obiektu są uprawnione do zmiany jego stanu. Obiekt z uwzględnieniem hermetyzacji jest skonstruowany w ten sposób, że nie ma bezpośredniego dostępu z zewnątrz instancji, do zmiennych danego obiektu. Dostęp – odczyt i zapis wartości właściwości obiektu możliwy jest jedynie za pomocą metody i funkcji wbudowanych w obiekt. Elementy składowe, które obiekt udostępnia, tworzą interfejs obiektu. Niektóre języki programowania umożliwiają stworzenie obiektu specjalnego typu, tzw. interfejs. Jest to obiekt, który zawiera jedynie informację o metodach, które powinny znaleźć się w obiekcie tego typu. Na podstawie takiego interfejsu tworzy się obiekt końcowy. Niektóre języki osłabiają to założenie, dopuszczając pewien poziom bardziej bezpośredniego (kontrolowanego) dostępu do wszystkich składowych obiektu. Ograniczają w ten sposób poziom abstrakcji.

Polimorfizm

Referencje i kolekcje obiektów mogą dotyczyć obiektów różnego typu, a wywołanie metody dla referencji spowoduje zachowanie odpowiednie dla typu obiektu wywoływanego. Jeśli dzieje się to w czasie działania programu, to nazywa się to późnym wiązaniem lub wiązaniem dynamicznym. Niektóre języki udostępniają również statyczne, wykonywane w trakcje kompilacji programu rozwiązania polimorfizmu. Językiem oferującym statyczne podejście do polimorfizmu jest C++, który posiada na przykład szablony i przeciążanie operatorów.

Dziedziczenie

Umożliwia tworzenie nowych obiektów z wykorzystaniem już istniejących, rozszerzając ich funkcjonalność. Mając zdefiniowany obiekt typu telefon, który posiada kilka metod i funkcji, można na jego podstawie utworzyć nowe obiekty, takie jak: telefon komórkowy, stacjonarny, czy przenośny. Każdy z nich będzie posiadał podstawowe funkcje obiektu, z którego powstał plus nowe, charakterystyczne dla danego modelu. Tworząc w ten sposób obiekty potomne, nie trzeba programować od nowa całych obiektów, a jedynie funkcje, których w obiekcie podstawowym nie było. Czasami mówi się, że w większości przypadków powstają grupy obiektów zwane klasami, oraz grupy klas zwane drzewami. Odzwierciedlają one wspólne cechy obiektów.

Języki obsługujące dziedziczenie wielokrotne, czyli tworzenie obiektu z kilku innych:
- obsługujące w całości: C++ i Python,
- obsługujące w części (tylko jako interfejsy): Object Pascal, Java.

Wszystkie obiekty wywodzą się z jednego korzenia i muszą mieć nadklasę: Java, C#, Object Pascal. Języki C++ i Python nie wymagają, aby każdy obiekt miał nadklasę.

Paradygmat programowania obiektowego jest w niektórych językach nieco zmieniony:
- w niektórych językach każda klasa musi mieć nadklasę (najczęściej jest to klasa typu Object),
- w niektórych językach nadklas może być więcej niż jedna.

4.2.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.
1. Jakie elementy tworzą paradygmat programowania zorientowanego obiektowo?
2. Czym jest abstrakcja?
3. Czy powinno się modyfikować wartości własności obiektów bezpośrednio odwołując się do nich, czy w inny sposób?
4. Czym jest polimorfizm?
5. Czym jest enkapsulacja?
6. Jak działa dziedziczenie i dlaczego jest ułatwieniem dla programisty?
7. Czym jest nadklasa obiektu?

4.2.3. Ćwiczenia

Ćwiczenie 1

Zdefiniuj klasę (nazwy własności i funkcje obiektu) opisującą telewizor jako obiekt podstawowy i telewizor kineskopowy, plazmowy, LCD jako odmiany podstawowego obiektu. Rozpisując elementy składowe pamiętaj o dziedziczeniu i zachowaniu zasad abstrakcji i hermetyzacji.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) zapisać w dokumencie tekstowym klasę: telewizor,
2) dopisać własności telewizora,
3) dopisać funkcje telewizora,
4) dopisać telewizor kineskopowy jako nową klasę,
5) uzupełnić własności,
6) uzupełnić funkcje,
7) powtórzyć operacje 4-6 dla pozostałych typów telewizora.

Wyposażenie stanowiska pracy:
- komputer z zainstalowanym edytorem teksu.

Ćwiczenie 2

Jeżeli obiektem podstawowym jest myśliwiec wojskowy, to czy obiektem pochodnym może być samolot pasażerski? Swoją odpowiedź uzasadnij?

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) zapisać w dokumencie tekstowym klasę: myśliwiec wojskowy,
2) dopisać własności myśliwca,
3) dopisać funkcje myśliwca,
4) dopisać nową klasę: samolot pasażerski,
5) spróbować uzupełnić własności,
6) spróbować uzupełnić funkcje,
7) porównać otrzymane własności z wyobrażeniem samolotu pasażerskiego,
8) porównać otrzymane funkcje z punktu wyobrażeniem samolotu pasażerskiego,
9) zapisać wnioski z porównań.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym edytorem teksu.

Ćwiczenie 3

Zdefiniowana jest klasa samochód, której jedną z właściwości jest ilość paliwa. Oto postać klasy:
- składowe wartości: silnik, skrzynia biegów, kolor, typ nadwozia, ilość paliwa,
- składowe funkcje: uruchom i zatrzymaj silnik, zmień bieg w górę i w dół.

Brakuje w niej obsługi tankowania. Dopisz odpowiednie działanie. Klasa powinna być stworzona z zachowaniem zasady abstrakcji.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) napisać klasę samochód,
2) zdefiniować atrybuty klasy,
3) zdefiniować metody klasy,
4) dopisać metodę tankowania w taki sposób, by zależała od ilości paliwa,
5) sprawdzić, czy zachowałeś zasadę abstrakcji.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym edytorem teksu.

4.2.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) wyjaśnić pojęcie abstrakcji?	☐	☐
2) wyjaśnić pojęcie hermetyzacji?	☐	☐
3) wyjaśnić pojęcie dziedziczenia?	☐	☐
4) wyjaśnić pojęcie polimorfizmu?	☐	☐

5) wymienić zalety dziedziczenia? ☐ ☐

6) zdefiniować różnicę pomiędzy obiektem nadrzędnym i podrzędnym? ☐ ☐

7) wyjaśnić czym jest interfejs i jakie są jego zalety ? ☐ ☐

8) dla dowolnej klasy zdefiniować elementy jej interfejsu? ☐ ☐

4.3 Obiektowo zorientowana analiza problemów

4.3.1. Materiał nauczania

Znając już definicję obiektów i podstawowe założenia paradygmatu podejścia zorientowanego obiektowo, można przejść do analizy i/lub projektowania z wykorzystaniem obiektów. Istnieje teoria, że każdą rzecz, element, sytuację czy proces da się opisać za pomocą obiektu lub grupy obiektów. Jest to podejście, które wymaga od osoby opisującej większego nakładu pracy, ale też gwarantuje dokładniejsze, wierniejsze i bardziej trafne opisanie i przeanalizowanie problemu.

Założeniem podejścia obiektowego do problemów jest „rozbicie sytuacji" na poszczególne składowe rozumiane jako obiekty i opisanie każdego z wyłonionych obiektów, w szczególności zwrócenie uwagi na wyliczenie własności i funkcji. Mając określone obiekty, ustala się zależności między nimi. Definiuje się zmienne, a także określa w jaki sposób funkcje jednego obiektu wpływają na inny i które z nich są niezbędne dla prawidłowego działania procesu, a które stanowią opcję dodatkową. Na tym etapie tworzy się graf obiektów i zależności.

Projektowanie i analizowanie zagadnień jest czynnością, która powinna poprzedzać programowanie systemu. Czas poświęcony na projektowanie nie tylko nie jest stracony. Projektowanie chroni przed potrzebą wielokrotnego przepisywania fragmentów kodu.

Istnieje kilka systemów wspomagających opisywanie problemów, czyli zagadnień informatycznych. Jednym z najpopularniejszych i wartych poznania w przyszłości jest język UML (ang. Unified Modeling Language, czyli Ujednolicony Język Modelowania; zob. „UML dla każdego"). Graficzny język opisu UML dzieli zagadnienia na następujące diagramy: klasy (class), obiekty (object), pakiety, przypadki użycia (use case), struktury złożone, wdrożenia (diagram abstrakcyjny), komponenty, czynności (activity), maszyny stanowe, zależności czasowe i interakcje.

4.3.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.

1. Jakie są korzyści z analizowania problemu w sposób obiektowy?
2. Dlaczego programowanie systemu informatycznego powinno być poprzedzone projektowaniem?
3. Czy istnieje język graficznego opisu problemu?

4.3.3. Ćwiczenia

Ćwiczenie 1

Rozrysuj wybrany problem wykorzystując metodę zorientowaną obiektowo. Może to być na przykład stacja obsługi pojazdów. Przeanalizuj czynności i obiekty, które biorą udział w procesie (możesz wykorzystać język UML).

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) spróbować wyłonić składowe problemu,
2) opisać składowe,
3) utworzyć sieć połączeń pomiędzy poszczególnymi elementami.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym edytorem teksu.

4.3.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) zdefiniować zasadę projektowania zorientowanego obiektowo?	☐	☐
2) narysować graf obiektów?	☐	☐
3) wyjaśnić na czym polega idea języka UML?	☐	☐

4.4 Struktura programu w C++

4.4.1. Materiał nauczania

Programy napisane w języku obiektowym, podobnie jak w językach strukturalnych, posiadają strukturę, czyli zbiór zasad zapisywania programu. Najprostszy program w języku C++ może mieć następującą postać:

```
#include <iostream.h>

int main()
{
    cout << "Witaj";
    cout << endl << "świecie";
    return 0;
}
```

Na początku programu znajdują się dyrektywy preprocesora. Preprocesor jest to cześć środowiska programistycznego, która przed kompilacją przegląda kod pod kątem specjalnych instrukcji sterujących pracą linkera i kompilatora. Tego typu dyrektywa w przypadku powyższego fragmentu kodu jest dyrektywa `#include <iostream.h>`, której zadaniem jest dołączenie do naszego programu biblioteki nagłówkowej `iostream.h`. Biblioteka ta zawiera kod i funkcje, które są konieczne, aby móc sterować strumieniami (m. in. wyprowadzać informację na ekran).

Kolejnym elementem jest funkcja `main` – podstawowy element programu. Jest to odpowiednik bloku `begin … end.` w języku Pascal. Jest to funkcja, która zostaje wywołana po kompilacji programu i zawiera główny blok kodu programu. Funkcję `main` należy poznać dokładniej. Jest to najważniejszy element. Pierwszy uruchamiany, z którego wywoływane są następnie kolejne funkcje i procedury. Funkcja `main` występuje w prostych programach konsolowych (takich jak powyższy), ale również w programach Windows API, czy przy budowie programów wykorzystujących środowiska Borland C++. Programowanie Windows API (programowanie aplikacji z wykorzystaniem systemu graficznego Windows) u swych podstaw także posiada funkcję główną. Jej zadaniem jest stworzenie okna głównego aplikacji i rozpoczęcie nieskończonej pętli komunikatów systemowych.

Przedstawiony program nie jest aż tak skomplikowany. Jego zadaniem jest wyświetlenie napisu: „Witaj świecie" w dwóch liniach. Na pierwszy rzut oka kod może wydawać się trudny do zrozumienia. Brak tu typowych instrukcji wyprowadzających dane na ekran typu `write` lub `print`. Zamiast nich jest instrukcja `cout` i operator `<<`. Te dwa elementy są odpowiednikiem wcześniej wspomnianych instrukcji, ale działają na innych zasadach. Wykorzystują przesyłanie danych za pomocą strumieni, czyli połączeń pomiędzy nadawcą i odbiorcą w postaci nieprzerwanej sekwencji danych. w tym przypadku koniec strumienia to ekran, ale może to być również plik, port szeregowy, USB albo drukarka.

Zmodyfikujmy trochę kod programu:

```
#include <iostream.h>
#include <conio.h>
#pragma hdrstop

int main()
{
    cout << "Witaj";
    cout << endl << "świecie";
    getch();

    return 0;
}
```

Dopisany kod został pogrubiony. Dyrektywa przyłączenia pliku conio.h jest potrzebna, aby można było skorzystać z funkcji `getch()`. Dyrektywa `#pragma hdrstop` informuje preprocesor, że w tym miejscu kończą się pliki nagłówkowe, które należy importować (może nieznacznie przyspieszyć kompilację). Ostatnim dopisanym elementem jest funkcja `getch()` (z biblioteki conio.h), która oczekuje na znak wprowadzony przez użytkownika. Po skompilowaniu zmodyfikowanego programu, uruchomi się on i nie zakończy działania zaraz po wyświetleniu napisu (jak to było wcześniej), ale będzie oczekiwał na dowolny znak, wprowadzony z klawiatury przez użytkownika.

Występują dwa typowe schematy struktury programu:

Dyrektywy preprocesora
Deklaracje stałych
Deklaracje funkcji i klas
Funkcja main
Definicje funkcji

Dyrektywy preprocesora
Deklaracje stałych
Deklaracje funkcji i definicje funkcji Deklaracje klas i definicje klas
Funkcja main

4.4.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.
1. Jakie są elementy struktury kodu programu w języku C++?
2. Jaka dyrektywa jest stosowana do linkowania bibliotek?
3. Do czego służy zapis `cout << "Janek";` ?
4. Jaka jest różnica pomiędzy wyprowadzaniem danych w C++ i języku Pascal?

4.4.3. Ćwiczenia

Ćwiczenie 1

Odszukaj w Internecie przykłady programów napisanych w języku C++. Przyjrzyj się ich strukturze i zapisowi.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) zapisać w edytorze tekstowym hasła określające rodzaj programu. Myśleć raczej o niewielkich programach. Można wykorzystać tematy, o których uczyłeś się na algorytmice.
2) uruchomić dobrą wyszukiwarkę i wpisz zapytanie.
3) przekopiować kody kilku programów do dokumentu tekstowego.
4) oznaczyć programy, których kod wydaje Ci się czytelny i zrozumiały. Opierać się na doświadczeniach z programowania strukturalnego.
5) zapisać, które cechy mogą powodować, że w punkcie 4 wybrałeś właśnie te programy.

Wyposażenie stanowiska pracy:
– komputer z zainstalowaną przeglądarką internetową i edytorem tekstu.

Ćwiczenie 2

Sprawdź, co się stanie, jeżeli z kodu programu wykasujesz dwie pierwsze linijki. Czy program będzie działał prawidłowo? Czy się skompiluje?

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) wybrać dowolny z programów skopiowanych w Ćwiczeniu 1,
2) skompilować ten program,
3) usunąć dwie pierwsze linie,
4) zanotować reakcję kompilatora,
5) podzielić się spostrzeżeniami z innymi uczniami w grupie.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++ i edytorem tekstu (lub środowisko programistyczne)

4.4.4. Sprawdzian postępów

Czy potrafisz:	Tak	Nie
1) omówić strukturę programu w języku C++?	☐	☐
2) wyjaśnić, dlaczego na początku programu trzeba określać biblioteki, które wykorzystuje się w kodzie?	☐	☐
3) wyjaśnić różnice pomiędzy wyprowadzaniem informacji w Pascalu i C++?	☐	☐
4) określić istotę funkcji głównej `main`?	☐	☐

4.5. Deklaracje, stałe, zmienne, wyrażenia i operatory

4.5.1. Materiał nauczania

Dane przechowywane są w pamięci komputera, z poziomu programowania asemblera dostęp do nich możliwy jest za pomocą adresów lub bloków pamięci zapisywanych w kodzie szesnastkowym. Języki wysokiego poziomu ułatwiają korzystanie z danych przechowywanych w pamięci za pomocą zmiennych i stałych. Zmienne i stałe są podstawowymi elementami każdego języka programowania. Zmienne mogą reprezentować różne wartości w czasie wykonywania (działania) programu. W języku C++ nazwy zmiennych i stałych muszą być zadeklarowane prze użyciem. Deklaracja zmiennych składa się z typu zmiennej i jej nazwy. Podczas deklarowania dopuszcza się również podanie wartości domyślnej.

Zmienne można podzielić na zmienne typów prosty chi złożonych. Do typów prostych należą:

int - liczby całkowite,
float - liczby zmiennoprzecinkowe,
double - liczby zmiennoprzecinkowe o podwójnej długości (precyzji),
char - znaki,
void - typ pusty.

Przykłady deklaracji zmiennych:

```
int wiek;
float pieniadz;
char znak;
```

Deklarując zmienną można przypisać jej wartość domyślną.

```
int wiek = 24;
float pieniadz = 760.00;
```

Często w programach wykorzystuje się stałe. Stała jest to nazywa pewnej wartości, która w całym programie nie ulega zmianie. Stałe deklaruje się dodając przed typem słowo kluczowe const:

```
const float pi = 3.14;
```

W języku C++ wielkość liter w nazwie zmiennych ma znacznie. Zmienne o nazwach: pi, Pi, pI, PI będą czterema różnymi zmiennymi. Na zmiennych i stałych można przeprowadzać operacje arytmetyczne i logiczne. Na zmiennych można wykonywać operacje przypisania.

Poniżej przykład programu, który wykorzystuje zmienne i operator mnożenia:

```
#include <iostream.h>
#include <conio.h>
#pragma hdrstop

const float pi = 3.14;
int r; //deklaracja zmiennych
float obwod;

int main()
{
  cout << "Podaj promień okręgu w metrach: ";
  cin >> r;
  obwod = 2 * pi * r;
cout << "Obwód wynosi: " << obwod;
cout << endl << "Naciśnij jakiś klawisz...";
getch();
return 0;
}
```

Stosuje się następujące operatory arytmetyczne (według priorytetu):

`++` - operator inkrementacji,
`--` - operator dekrementacji.
`*` - operator mnożenia,
`/` - operator dzielenia,
`%` - operator reszty z dzielenia,
`+` - operator dodawania,
`-` - operator odejmowania,

Operator inkrementacji zwiększa wartość zmiennej o jeden. Istnieją dwa jego zastosowania: jako pre- i post-inkrementacja. Preinkrementacja zwiększa wartość zmiennej przed wykonaniem pozostałych działań w wyrażeniu. Postinkrementacja najpierw wykonuje działania, a następnie zwiększa wartość. Oto dwa przykłady prezentujące tą zasadę:

```
a = 2;
wynik = a++ * 2;
```

Zmienna `wynik` będzie miała wartość 4, zmienna `a` wartość 3. w drugim przykładzie:

```
a = 2;
wynik = ++a * 2;
```

zmienna `wynik` będzie miała wartość 6, zmienna `a` wartość 3.

Analogiczne działa operator dekrementacji: --.

Stosuje się następujące operatory logiczne:

`!` - operator zaprzeczenia,
`>` - większe niż,
`>=` - większe lub równe,
`<` - mniejsze niż,
`<=` - mniejsze lub równie,

`==` - równe,
`!=` - różne,
`||` - operator alternatywy,
`&&` - operator koniunkcji.

Stosuje się następujące operatory przypisania:

`=` - przypisuje,
`+=` - dodaje i przypisuje,
`-=` - odejmuje i przypisuje,
`*=` - mnoży i przypisuje,
`/=` - dzieli i przypisuje,
`&=` - przypisuje resztę z dzielenia.

Wyrażenie: `wynik += 5`; jest odpowiednikiem zapisu: `wyniki = wynik + 5`.
Operatory wraz ze stałymi i zmiennymi, na które działają, tworzą wyrażenia. Warto wspomnieć, że w języki C/C++ mają jeszcze inne specyficzne operatory. Można również definiować własne operatory.

4.5.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz czy jesteś przygotowany do wykonania ćwiczeń.
1. Wymień typy proste.
2. Jaka jest różnica pomiędzy stałymi i zmiennymi?
3. Czy w momencie deklaracji można przypisać wartość do stałych? A do zmiennych?
4. Czy wielkość liter w nazwach zmiennych lub stałych ma znacznie?
5. Jakie operatory oferuje C++?
6. Jaka jest różnica pomiędzy post- i pre-dekrementacją?

4.5.3. Ćwiczenia

Ćwiczenie 1

Napisz program liczący drugą potęgę liczby podanej przez użytkownika.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) przedstawić algorytm realizujący zadanie, o którym mowa w ćwiczeniu,
2) wprowadzić odpowiadający kod,
3) skompilować program,
4) sprawdzić działanie programu.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 2

Napisz program, który policzy pole i obwód trapezu równobocznego, na podstawie danych podanych przez użytkownika.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) przedstawić algorytm realizujący funkcję, o której mowa w ćwiczeniu,
2) wprowadzić odpowiadający kod,
3) skompilować kod,
4) sprawdzić działanie programu.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

4.5.4. Sprawdzian postępów

Czy potrafisz:	Tak	Nie
1) zdefiniować pojęcie zmiennej i stałej?	☐	☐
2) wymienić zalety i zastosowania zmiennych?	☐	☐
3) wymienić podstawowe typy zmiennych?	☐	☐
4) zadeklarować zmienną i stałą?	☐	☐
5) zdefiniować wyrażenia arytmetyczne? Logiczne?	☐	☐
6) posługiwać się inkrementacją i dekrementacją?	☐	☐
7) posługiwać się operatorami arytmetycznymi i przypisania?	☐	☐

4.6. Instrukcje

4.6.1. Materiał nauczania

Język programowania C++ posiada kilka wbudowanych instrukcji: instrukcje sterujące, instrukcje warunkowe, pętle.

Instrukcje warunkowe

Zadaniem instrukcji warunkowej jest sterowanie zmiennymi, operacjami i przebiegiem programu na podstawie stanu warunku. Konstrukcja instrukcji warunkowej w języku C++ jest następująca:

```
if(warunek)
{
  kody wykonany, gdy warunek jest spełniony
}
else
{
  kody wykonany, gdy warunek nie jest spełniony
}
```

Wyróżnia się warunki w postaci zmiennej, wyrażenia logicznego lub kilku warunków. Warunek można zastąpić dowolną zmienną typu `int`. Jeżeli jej wartość jest różna od zera, to warunek jest spełniony. Gdy wartość zmiennej wynosi zero, to warunek nie jest spełniony. Inaczej rzecz ujmując, w języku C++ nie ma typowej zmiennej logicznej. Zamiast niej wykorzystywana jest zmienna typu całkowitego. Co prawda w języku C++ wprowadzono typ logiczny `bool`, ale jest to bardziej zabieg stylistyczny (wprowadzony w celu zgodności z innymi językami i przejrzystości kodu) niż wymóg konieczny.

Taka interpretacja zmiennych całkowitych ma sporo zalet, ale należy pamiętać o jej specyfice. Bardzo często powtarzającym się błędem jest zapis `if(a=4)`. Ten pseudo-warunek będzie zawsze spełniony, ponieważ nie następuje sprawdzenie warunku równoważności, lecz przypisania. Aby zapis był prawidłowy, należy użyć symbolu złożonego składającego się z dwóch znaków równości: `if(a==4)`. Wyrażenia logiczne składają się ze stałych i zmiennych połączonych operatorem logicznym.

W funkcji warunkowej może występować kilka warunków logicznych. Każdy z nich powinien być ujęty w nawiasy, a pomiędzy nawiasami musi wystąpić jeden z dwóch operatorów: koniunkcji (&&) lub alternatywy (||).

Przykładowy program:

```
#include <iostream.h>
#include <conio.h>

int wiek;

int main()
{
    wiek  = 13;
    if((wiek>12)&&(wiek<18)) // instrukcja warunkowa
    {
       cout << "Wiek jest pomiędzy 12 a 18";
       getch();
    }
    return 0;
}
```

Pętle

Pętle są niezbędnym elementem każdego języka programowania. Umożliwiają wielokrotne wykonanie wyrażenia lub fragmentu kodu. Dzięki nim otrzymuje się przydatny mechanizm programistyczny ułatwiający rozwiązywanie wielu problemów algorytmicznych przy znaczącej oszczędności kodu. Jeżeli dowolna instrukcja ma zostać wykonana więcej niż dwa razy, to warto w tym miejscu programu zastosować pętle.

Pętla for:

Pętla oparta na zmiennej liczbowej – zmiennej sterującej. Oto konstrukcja instrukcji for:

```
for (instrukcja1; warunek1; instrukcja2)
{
    kod wykonywany w pętli;
}
```

Instrukcja pierwsza jest przypisaniem wartości początkowej zmiennej sterującej. Warunek kończący iterację musi być spełniony, aby instrukcje umieszczone w pętli wykonały się kolejny raz. Instrukcja druga zmienia wartość zmiennej sterującej. Prosty przykład pętli for:

```
#include <iostream.h>
#include <conio.h>
int main()
{
    int x;
    for (x = 0; x < 10; x++)
    {
      cout << "Wartość zmiennej x: " << x << endl;
    }
    return 0;
}
```

Kod wykonywany w pętli może wykorzystać instrukcję `break`, która przerywa wykonanie kodu i pętli, albo instrukcję `continue`, która przerywa wykonanie kodu wewnątrz pętli i wykonuje kolejna iterację pętli.

Pętla while:

Pętla `while` jest wykonywana, o ile warunek jest spełniony. Konstrukcja jest następująca:

```
while (warunek)
{
    kod wykonywany w pętli;
}
```

Warunkiem może być dowolne wyrażenie lub wyrażenia logiczne. Stosując pętlę `while` należy pamiętać, że w kodzie pętli muszą istnieć instrukcje, które gwarantują przerwanie jej działania poprzez zmianę warunku pętli lub wykonanie bezwarunkowego wyjścia z pętli. Prosty przykład pętli:

```
int i = 1;
while(i <10)
{
      cout << "Wartość zmiennej i: " << i << endl;
      i++;
}
```

Pętla do…while:

Pętla `do…while` jest wykonywana dopóki warunek jest prawdziwy. Typowa konstrukcja wygląda następująco:

```
do
{
  kod wykonywany w pętli
}
while(warunek);
```

Przykład:

```
int x=1;
do
{
    x++;
    cout << x;
}
while(x<10);
```

Z przedstawionych definicji wynikają również różnice w stosowaniu. Pętla `do..while` zawsze wykonywana jest przynajmniej jeden raz.

Instrukcje switch:

Instrukcja `switch` jest instrukcją, która wykonuje odpowiednie instrukcje zapisane po słowie `case` w zależności od wartości zmiennej sterującej.

```
switch(zmiena_sterujaca)
{
  case wartosc1: kod wykonany w przypadku wystąpienia danej wartości
         break;
  case wartosc2: kod wykonany w przypadku wystąpienia danej wartości
         break;
  default: kod wykonany w przypadku wystąpienia danej wartości

}
```

Przykład:

```
switch(wiek)
{
  case 16: cout << ”Masz 16 lat”;
         break;
  case 18: cout << ”Masz 18 lat”;
         break;
  default: cout << ”Nie masz ani 16, ani 18 lat”;

}
```

4.6.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.

1. Jakie instrukcje występują w języku C++?
2. Jaka jest konstrukcja instrukcji warunkowej?
3. Jaka jest konstrukcja pętli `for`?
4. Jaka jest konstrukcja pętli `while`?
5. Jaka jest konstrukcja pętli `do…while`?
6. W jakim celu stosuje się pętle?

4.6.3. Ćwiczenia

Ćwiczenie 1

Napisz program, który obliczy dowolną potęgę liczby 2. Użytkownik powinien móc wpisać wartość wykładnika. Następnie zmodyfikuj program w taki sposób, aby wyliczał dowolną potęgę podstawy podanej przez użytkownika.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) przedstawić algorytm realizujący funkcję, o której mowa w pierwszej części ćwiczenia,
2) wprowadzić odpowiadający kod,
3) skompilować kod,
4) sprawdzić działanie programu,
5) zmodyfikować kod zgodnie z drugą częścią ćwiczenia,
6) sprawdzić działanie programu.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 2

Napisz program, który wylicza sumę ciągu arytmetycznego dla wartości podanych przez użytkownika. Wykorzystaj pętlę `for`. Następnie zmodyfikuj program w taki sposób, by korzystał z pętli `while` i pętli `do while`.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) opracować algorytm realizujący funkcję, o której mowa w pierwszej części ćwiczenia,
2) wprowadzić odpowiadający kod i skompilować go,
3) sprawdzić działanie programu,
4) zmodyfikować kod zgodnie z drugą częścią ćwiczenia,
5) sprawdzić działanie programu.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 3

Napisz program, którego celem będzie wyświetlenie tabliczki mnożenia (do 100).

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) opracować i przedstawić algorytm realizujący funkcję, o której mowa w ćwiczeniu,
2) wprowadzić odpowiadający kod i skompilować go,
3) sprawdzić działanie programu. Zadbaj o poprawne formatowanie.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

4.6.4. Sprawdzian postępów

Czy potrafisz:	Tak	Nie
1) zdefiniować instrukcje warunkowe?	☐	☐
2) zastosować instrukcje warunkowe proste i złożone?	☐	☐
3) wymienić sytuacje, w których wykorzystuje się instrukcje warunkowe?	☐	☐
4) zdefiniować pętle?	☐	☐
5) wymienić różnice i podobieństwa pomiędzy różnymi pętlami?	☐	☐
6) wymienić zastosowanie pętli?	☐	☐
7) napisać program wykorzystujący pętle?	☐	☐
8) napisać program wykorzystujący instrukcje warunkowe?	☐	☐

4.7. Pliki. Preprocesor

4.7.1. Materiał nauczania

Kompilator przetwarza kod języka C++ na język maszynowy. Podstawowymi plikami poddawanymi kompilacji są pliki o rozszerzeniach *.cpp i *.h. Do tej pory korzystaliśmy przede wszystkim z plików o rozszerzeniach „cpp". Pliki te zawierają kod programu i kody klas. Wykorzystuje się również pliki nagłówkowe o rozszerzeniach „h". Pliki te zawierają listę deklaracji funkcji lub deklarację klasy, która jest zakodowana w pliku o identycznej nazwie i rozszerzeniu „cpp". Po kompilacji, w katalogu z kodem programu zostanie zapisany plik wynikowy. Jest to plik o rozszerzeniu „obj". Następnie uruchamiany jest konsolidator, który łączy pliki *.obj z plikami bibliotek *.lib i plikami zasobów *.res (w przypadku systemów Windows). Efektem końcowym jest plik wykonywalny o rozszerzeniu „exe".

Opisany powyżej schemat działania przedstawia typową kolejność. Programista może wpływać na kompilator wykorzystując dyrektywy preprocesora. Dyrektywy te są swoistym językiem, którego polecenia są interpretowane przed rozpoczęciem procesu kompilacji. Można powiedzieć, że jest to metoda konfigurowania procesu kompilacji. Pisząc przykładowe programy w poprzednich rozdziałach skorzystaliśmy z poleceń preprocesora. Były to dyrektywa: `#include`.

Dyrektywa #include

Dyrektywa ta nakazuje preprocesorowi wstawić plik nagłówkowy. Nazwa pliku powinna być podana pomiędzy znakami < > (większości i mniejszości) lub " " (cudzysłów). Plik o nazwie podanej pomiędzy znakami < > jest wyszukiwany w katalogu z plikami nagłówkowymi. W systemie Windows ścieżka do tego pliku zapisana jest w konfiguracji kompilatora, systemie Linux lub Unix katalog nagłówkowy znajduje się w /usr/include. Czasami również w przypadku niektórych kompilatorów jest to katalog

/usr/local/include/c++/[numer wersji] lub /usr/include/c++/[numer wersji]). Plik o nazwie podanej w cudzysłowie jest szukany w katalogu bieżącym. Przykładowo:

```
#include <stdlib.h>
```

wstawia plik stdlib.h znajdujący się w katalogu z nagłówkami, zaś:

```
#include "czlowiek.h"
```

wstawia plik czlowiek.h znajdujący się w bieżącym katalogu. Jeśli plik nie zostanie odnaleziony, kompilator szuka go w katalogach z plikami nagłówkowymi. Pisząc pierwszy program wykorzystaliśmy dyrektywę:

```
#include <iostream>
```

Dyrektywa ta nakazuje wczytać deklaracje przydatne do wykonania operacji wejścia i wyjścia. Standard ISO C++ zakładał użycie zapisu <iostream.h>. Obecnie zalecane jest stosowanie nagłówków standardowych bez rozszerzenia „h", ale jednocześnie istnieją ważne wyjątki i różnice.

Dyrektywa #define

Dyrektywa ta służy do tworzenia makrodefinicji. Makrodefinicje pozwalają zastąpić dowolny ciąg znaków w kodzie programu innym ciągiem znaków (np.: w całym kodzie, a nie w ciągu znaków). Dzięki temu makrodefinicje mogą służyć do definiowania stałych. Na przykład:

```
#define LICZBA_PI 3.14
```

Ważne jest, aby rozumieć zasadę działania stałej utworzonej w ten sposób. Preprocesor jest wywoływany przed kompilatorem. Przegląda on kod program i zamienia wszystkie miejsca gdzie występuje zapis `LICZBA_PI` na wartość 3.14 i dopiero wtedy kompiluje program. W efekcie program jest kompilowany z liczbą 3.14 we wszystkich wyrażeniach, w których występowała fraza `LICZBA_PI`. Zaleca się, by definicje makr były zapisywane drukowanymi literami, co odróżni je w kodzie od innych zmiennych i stałych. Dzięki temu uniknie się potencjalnych błędów. Makrodefinicje mogą zawierać wyrażenia i parametry.

Kompilacja warunkowa

Jest to ciąg dyrektyw, które umożliwiają wyłączenie lub włączenie fragmentu kodu do kompilacji. Dyrektywy kompilacji warunkowej w dużym uproszczeniu odpowiadają komentarzom, dzięki którym programista może fragment kodu wyłączyć z kompilacji, z tą różnicą, że dzieje się to automatycznie, na podstawie wcześniej zdefiniowanych warunków. Często stosuje się tę technikę w sytuacji, gdy kod programu ma być kompilowany na różnych procesorach albo za pomocą różnych kompilatorów. Można wtedy przygotować kod dla różnych kompilatorów. Na przykład, w celu zapewnienia zgodności sprzętowej:

```
#if !defined( __STDLIB_H )
   Kod programu kompilowany przy spełnionym warunku
#endif
```

Poza wyżej opisanymi dyrektywami występuje jeszcze kilka rzadziej stosowanych. Są nimi:

- `#error` - powoduje wyprowadzenie błędu kompilacji z podanym jako argument komunikatem (używane tylko w połączeniu z dyrektywami warunkowymi),
- #pragma - zmienia ustawienia kompilatora (użycie tej dyrektywy zależy wyłącznie od implementacji).

4.7.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.
1. Jakie pliki powstają podczas i po procesie kompilacji?
2. Co to jest dyrektywa preprocesora i jakie ma zastosowanie?
3. Jakie są dyrektywy preprocesora?

4.7.3. Ćwiczenia

Ćwiczenie 1

Sprawdź jakie pliki powstają podczas procesu kompilacji i po jej zakończeniu. Korzystając z notatnika podejrzyj zawartość tych plików i sprawdź, w którym z nich znajduje się tekst, a które z nich są w postaci bitowej.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) ustalić, gdzie zapisywana jest kompilacja,
2) otworzyć folder i zapamiętać jego stan,
3) skompilować nowy program i sprawdź zawartość katalogu.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 2

Odszukaj w Internecie przykłady wykorzystania dyrektyw, a w szczególności dyrektyw warunkowych.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) uruchomić dobrą wyszukiwarkę,
2) wpisać zapytania dotyczące dyrektyw,
3) zapisać wyniki, które uznasz za interesujące,
4) podzielić się rezultatami z innymi uczniami w grupie.

Wyposażenie stanowiska pracy:
– komputer z zainstalowaną przeglądarką internetową

4.7.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) wymienić pliki, które powstają w wyniku kompilacji?	☐	☐
2) wyjaśnić pojęcie i zastosowanie dyrektyw?	☐	☐
3) omówić dyrektywę #include?	☐	☐

4) omówić dyrektywę #define?

5) wyjaśnić pojęcie dyrektywy warunkowe?

4.8. Funkcje C++

4.8.1. Materiał nauczania

Funkcje, to fragmenty kodu zgrupowane pod jedną nazwą w celu wielokrotnego użycia. Dzięki zapisaniu kodu w taki sposób, można wykorzystywać go wielokrotnie w różnych miejscach programu. Dzięki argumentom, czyli wartościom przesyłanym do funkcji, można budować konstrukcje, które zwracają różne wartości, zależnie od wartości argumentów. Składnia definiowania funkcji jest następująca:

```
typ nazwa (argumenty) {instrukcje}
```

Każda funkcja musi mieć określony typ wartości, jaką zwraca. Może to być dowolny z typów prostych (int, float, char itp.), albo złożonych (struktury lub unie). Po określeniu typu zwracanej wartości następuje nazwa funkcji (alfabet łaciński bez spacji), a w nawiasie lista argumentów, które mają być przesłane do ciała funkcji. Po nawiasie zapisywana jest instrukcja lub blok instrukcji ujęty w nawiasy klamrowe. Funkcja, która nie oczekuje argumentów może mieć następującą postać:

```
int Rok()
{
      int rok;
      rok = 2005;
      return rok;
}
```

Jest to funkcja typu `int`, czyli zwracana wartość będzie liczbą całkowitą. W ciele funkcji znajduje się definicja zmiennej `rok`, przypisanie oraz zwrócenie wartości za pomocą instrukcji `return`. Zasada działania jest bardzo podobna do funkcji znanych z języka Pascal, a różnice znajdują się przede wszystkim w zapisie.

Każda funkcja może posiadać argumenty. Deklaracja argumentu składa się z typu i nazwy. Liczba argumentów funkcji może być dowolna. Oto przykład:

```
int Mnozenie(int a, int b)
{
      int wynik;
wynik = a * b;
return wynik;
}
```

Istnieje specjalny typ funkcji: `void`. Nie zwraca ona żadnej wartości (odpowiednik procedury z języka Pascal).

Przykład:

```
void WyswietlNapis()
{
cout << ”To jest napis, który wyświetla funkcja”;
}
```

Oto przykład całego programu:

```
#include <iostream.h>

void WyswietlNapis()
{
cout << "To jest napis, który wyświetla nasza funkcja";
}

int main(int argc, char *argv[])
{
    WyswietlNapis();
    return 0;
}
```

W powyższym przykładzie funkcja `WyswietlNapis` została zadeklarowana i zdefiniowana przed funkcją `main`. Można rozdzielić te operacje i zadeklarować własną funkcję przed `main`, a zdefiniować na końcu programu. Zwiększa się czytelność programu:

```
#include <iostream.h>
#include <conio.h>

int Potega(int x);

int main(int argc, char *argv[])
{
    for (int x=0; x <= 10; x++)
    {
        cout << "Kwadrat " << x << " : " << Potega(x) << endl;
    }
    getch();
    return 0;
}

int Potega(int x)
{
return x*x;
}
```

Istnieje możliwość definiowania tzw. prototypu funkcji. Wystarczy wówczas określić ilość i rodzaj argumentów, natomiast nie trzeba definiować nazw.

Funkcje mogą być przeładowywane. Pojęcie to oznacza, że w kodzie programu może istnieć kilka funkcji o tej samej nazwie pod warunkiem, że będą się one różnić argumentami. W ten sposób można na przykład zadeklarować własną funkcję potęgowania na dwa sposoby:

```
int Potega(int x)
{
return x*x;
}

int Potega(int x, int y)
{
int wynik = x
for(int a = 2; a <= y; a++)
      {
      wynik = wynik * x;
      }
      return wynik;
}
```

Po takim zadeklarowaniu funkcji, w zależności od tego jak zostanie ona wywołana, może realizować różny, choć najczęściej zbliżony logicznie kod:

```
wynik1 = Potega(2);
wynik2 = Potega(2,5);
```

Zmienna `wynik1` przyjmie wartość 4, a zmienna `wynik2` przyjmie wartość 2^5=32.

Argumenty do funkcji można przekazać na trzy sposoby. Pierwszym jest przekazanie argumentów poprzez wartość. Taka sytuacja ma miejsce wtedy, gdy w wywołaniu występuje jawnie wartość, na przykład: `Potega(2)`. w takim przypadku do funkcji zostaje przekazana stała. Jeżeli w ciele funkcji wykonane zostaną jakiekolwiek operacje na argumencie, to nie będą one miały żadnego wpływu na wartość argumentu po zakończeniu działania funkcji.

Można też przekazać argument poprzez zmienną. Na przykład:

```
int a = 3;
cout << Potega(a);
```

W ten sposób w ciele funkcji pojawi się wartość zmiennej obliczonej w poprzedzających liniach kodu. Tu również operacje na argumencie nie będą miały wpływu na wartość parametru, czyli zmiennej globalnej `a`.

Korzystanie z trzeciego sposobu, czyli przekazanie wartości przez referencję spowoduje, że każda operacja na argumencie w ciele funkcji spowoduje zmianę wartości zmiennej globalnej. Jeżeli zmienna ma być przekazana funkcji przez referencję, to należy w deklaracji funkcji dopisać znak & do typu argumentu:

```
#include <iostream.h>

void podwoj(int& a, int& b, int& c)
{
  a*=2;
  b*=2;
  c*=2;
}

int main ()
{
  int x=1, y=3, z=7;
  podwoj(x, y, z);
  cout << "x=" << x << ", y=" << y << ", z=" << z;
  return 0;
}
```

Deklarowanie argumentów funkcji dopuszcza również stosowanie wartości domyślnych. Powodują one, że argument może być pominięty w odwołaniu. Należy pamiętać o tym, że tylko końcowe argumenty mogą mieć wartości domyślne. Oto przykład:

```
int potega (int x, int y = 2) // poprawny zapis
int potega (int x = 1, int y) // niepoprawny zapis
```

4.8.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.

1. Jaka idea leży u podstaw stosowania funkcji?
2. Jakie są podstawowe zastosowania funkcji?
3. Czym są argumenty funkcji?
4. Na czym polega i co daje przeładowywanie funkcji?
5. Jakie są trzy metody przekazywania wartości do funkcji?
6. Co to są wartości domyślne argumentów?

4.8.3. Ćwiczenia

Ćwiczenie 1

Napisz funkcję, która wyświetli komunikat tekstowy. Pozwól użytkownikowi wpisać tekst. Wykorzystaj funkcję kilkakrotnie.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) wpisać kod programu, który pobiera komunikat od użytkownika i wyświetla go na ekranie,
2) sprawdzić działanie,
3) zadeklarować blok kodu z punktu 1 jako funkcję,
4) zamknąć wywołanie funkcji w pętli. Przemyśl sposób zakończenia pętli,
5) sprawdzić działanie programu.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 2

Napisz funkcję, która obliczy średnią arytmetyczną dwóch liczb. Oblicz z jej pomocą średnią wartość kilku par liczb.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) zbudować funkcję, która obliczy średnią arytmetyczną dwóch liczb,
2) zamknąć wywołanie funkcji w pętli. Przemyśl sposób zakończenia pętli,
3) sprawdzić działanie programu.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 3

Napisz funkcje wykonujące następujące działania arytmetyczne: sumę ciągu geometrycznego, NWD, NWW.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) napisać kod funkcji wykonujących wymagane obliczenia,
2) przepisać funkcje w taki sposób, by wykorzystywały referencje,
3) sprawdzić działanie na przykładowych danych.

Wyposażenie stanowiska pracy:
– Komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 4

Napisz funkcję, która obliczy obwód prostokąta. Dopisz jej wersję przeładowaną, która w przypadku wystąpienia tylko jednego argumentu obliczy obwód kwadratu.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) napisać funkcję obliczającą obwód prostokąta. Sprawdź działanie,
2) dopisać funkcję o tej samej nazwie obliczającą obwód kwadratu,
3) sprawdzić działanie wywołując funkcję z różnymi argumentami.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 5

Sprawdź czy w programach, które do tej pory napisałeś nie powtarzają się identyczne fragmenty kodów. Jeżeli tak jest, to napisz funkcje, które realizują te zadania. Wstaw je do jednego pliku i za pomocą dyrektywy #include włącz ten plik do wcześniej napisanych programów.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) zapisać powtarzające się fragmenty kodu jako funkcje,
2) zapisać funkcje w osobnym pliku,
3) dołączyć plik do programu jako bibliotekę.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

4.8.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) zdefiniować pojęcie funkcji i jej argumentów?	☐	☐
2) wyjaśnić, czym jest typ funkcji, a w szczególności typ void?	☐	☐
3) wymienić zastosowania i napisać funkcje przeładowane?	☐	☐
4) przesyłać wartości do funkcji w oparciu o trzy sposoby?	☐	☐
5) pisać funkcje wykorzystując wartości domyślne?	☐	☐

4.9. Tablice

4.9.1. Materiał nauczania

Tablica to złożona struktura danych, która przechowuje wartości jednego typu. Wykorzystując tablice unika się niewygodnego nazywania zmiennych. Zamiast nazywać zmienne: osoba1, osoba2, osoba3 i tak dalej do osobaN, można utworzyć tablicę jednowymiarową o N elementach. Tablice deklaruje się podając najpierw typ zmiennych, które będą przechowywane w tablicy, następnie nazwę tablicy i w nawiasach kwadratowych jej wymiar:

```
typ nazwa [wymiar][wymiary…]
```

Przykład:

```
int wartosc[10];   //tworzy tablicę licz całkowitych o 10 elementach

float stanKonta[100]; //tworzy tablicę stu elementową, w której będą przechoywane liczby rzeczywiste
```

Tablice w języku C++ wykorzystują indeksy rozpoczynające się od zera. Oznacza to, że w przypadku pierwszej z powyższych definicji tablic, pierwszy element identyfikowany jest za pomocą indeksu [0], a ostatni [9]. Odwołanie się do elementu tablic o indeksie 10 spowoduje komunikat o przekroczeniu zakresu. Aby pobrać lub zapisać wartość w tablicy, podaje się jej nazwę i w nawiasach kwadratowych indeks (lub indeksy w przypadku tablicy wielowymiarowej). Nie ma wyraźnego ograniczenia co do wielkości tablic, czy liczby wymiarów. Ograniczeniem jest tylko pamięć komputera. Przykład:

```
#include <iostream.h>
#include <conio.h>

int main(int argc, char *argv[])
{
   int tablica[10][10];
   for (int x = 0; x < 10; x++)
     for (int y = 0; y < 10; y++)
     {
       tablica[x][y] = (x+1)*(y+1);
     }
   cout << "Tabliczna mnożenia" << endl;
   for (int x = 0; x < 10; x++)
   {
     for (int y = 0; y < 10; y++)
     {
       cout << tablica[x][y] << " ";
     }
      cout << endl;
   }
   getch();
   return 0;
}
```

Deklarując tablice można jednocześnie przypisać jej wartości początkowe. Składnia przypisania jest następująca:

```
int wartosc[4] = {1, 33, 56, 32};
float stanKonta[4] = {120.20, 999.99, 1.01, 1};
```

Dobry programista przy deklarowaniu rozmiarów tablicy skorzysta ze stałych. Ułatwi to ewentualne modyfikowanie rozmiaru tablicy, który kryje się w stałych występujących na początku kodu:

```
#include <iostream.h>
#include <conio.h>

#define HEIGHT 10
#define WIDTH 10
int main()
{
   int tablica[WIDTH][HEIGHT];
   for (int x = 0; x < WIDTH; x++)
     for (int y = 0; y < HEIGHT; y++)
     {
       tablica[x][y] = (x+1)*(y+1);
     }
   cout << "Tabliczna mnożenia" << endl;
   for (int x = 0; x < WIDTH; x++)
   {
     for (int y = 0; y < HEIGHT; y++)
     {
       cout << tablica[x][y] << " ";
     }
      cout << endl;
   }
   getch();
   return 0;
}
```

Tablice mogą być przekazywane do funkcji. Wystarczy w definicji funkcji zdefiniować argument jako tablicę.

Pojęciowo bliskim do tablic typem zmiennych są ciągi znakowe. W języku Pascal programista dysponował typem string, który w rzeczywistości był jednowymiarową tablicą znaków. W języku ISO C++ nie istnieje typ zmiennej „string". Zamiast tego stosuje się tablice znaków, czyli tablice składające się z elementów typu char. Każdy element z tak zadeklarowanej tablicy zawierać będzie jeden dowolny znak. Ostatni natomiast musi zawierać znak \0. Jest to znacznik końca ciągu znakowego. Składnia deklarowania i przypisywania wartości jest następująca:

```
char imie[20];
imie[0] = 'J';
imie[1] = 'u';
imie[2] = 's';
imie[3] = 't';
imie[4] = 'y';
imie[5] = 'n';
imie[6] = 'a';
imie[7] = '\0';
```

Takie przypisywanie wartości jest dość uciążliwe, dlatego warto wykorzystać inną własność języka C++: pojedynczy znak zapisywany jest w apostrofach '', natomiast ciąg znaków powinien być zapisany w cudzysłowiu:

```
char imie[20] = "Tomek";          //poprawna definicja
imie = "Adam";                    //błędne przypisanie.
```

Drugie przypisanie jest błędne, ponieważ w języku C++ sama nazwa tablicy nie reprezentuje całej tablicy, a jedynie adres pierwszego elementu. Stąd próba przypisania zmiennej `imie` wartości "Adam" zamienia adres tablicy na inną liczbę. Jeżeli trzeba

w programie zmienić wartość ciągu znaków (tablicy znaków), to należy skorzystać z funkcji strcpy (skrót od string copy), czyli kopiowania ciągu znakowego. Prawidłowe przypisanie wartości powinno wyglądać następująco:

```
strcpy(imie,"Justyna");
```

Korzystając z tej instrukcji powinniśmy pamiętać o dołączeniu biblioteki string.h.

Jak widać, implementacja ciągów znakowych w języku C++ jest bliższa tablicom niż prostym typom, jak to miało miejsce w języku Pascal.

Na zakończenie należy wspomnieć o jeszcze dwóch ważnych cechach tablic. Mianowicie, tablice mogą być definiowane bez określania wymiaru, poprzez pozostawienie pustych nawiasów kwadratowych. Taka definicja umożliwia zmianę wymiaru tablicy podczas pracy programu. Drugą istotną cechą, a właściwie metodą na policzenie wielkości tablicy, jest następujące podstawienie:

```
wymiar = sizeof(nazwa_tablicy)/sizeof(typ_tablic);
```

Przykład:

```
wymiar = sizeof(tablica)/sizeof(int);
```

4.9.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.

1. Czym są tablice?
2. W jaki sposób wykorzystuje się tablice?
3. Czym są tablice wielowymiarowe?
4. Jaka jest definicja i możliwości wykorzystania ciągu znakowego w C++?
5. Jak wylicza się rozmiar tablicy?

4.9.3. Ćwiczenia

Ćwiczenie 1

Utwórz tablicę jednowymiarową o rozmiarze 10 i wypełnij ją literami. Wyświetl wszystkie litery. Wyświetl wstecz. Policz, ile razy wystąpiła wybrana litera.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:

1) zadeklarować tablicę i wpisz litery, najlepiej stanowiące słowo,
2) dopisać polecenie, które określi rozmiar tablicy i dołącz do kodu pętlę, która wyświetli litery w porządku odwrotnym,
3) dołączyć do pętli licznik wystąpień wybranej litery korzystając z doświadczeń z programowania strukturalnego.

Wyposażenie stanowiska pracy:

– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 2

Elementy dwóch tablic o rozmiarze 2 reprezentują składowe x i y wektora. Oblicz sumę i różnicę wektorów. Oblicz składowe w przypadku sumy wektorów prostopadłych.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) narysować wektory na kartce lub w dokumencie tekstowym i oblicz sumy. Będzie można sprawdzić poprawność działania programu,
2) zdefiniować tablice w kodzie programu. Wpisz współrzędne wektorów,
3) dopisać instrukcje obliczające sumę wektorów:
   $[a_x,a_y] + [b_x,b_y] = [a_x+b_x,a_y+b_y]$
4) dopisać instrukcje obliczające różnicę wektorów:
   $[a_x,a_y] - [b_x,b_y] = [a_x-b_x,a_y-b_y]$
5) dopisać instrukcje obliczające sumę wektorów prostopadłych. Można skorzystać z najprostszej postaci, gdy jeden z wektorów jest równoległy do osi Ox, a drugi do osi Oy.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 3

Wyzeruj elementy na przekątnych macierzy o rozmiarze podanym przez użytkownika.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) rozważyć elementy tablicy znajdujące się na przekątnej. Jakie mają indeksy? Znajdź regułę wiążącą elementy na drugiej przekątnej,
2) wypełnić tablicę dowolnymi wartościami,
3) wyzerować elementy na przekątnych, korzystając z punktu 1,
4) wyświetlić tablicę na ekranie, żeby sprawdzić poprawność działania programu.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 4

Napisz program, który obliczy wyznacznik macierzy 2x2.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) znaleźć informację, jak oblicza się wyznacznik macierzy. Wykonać przykładowe obliczenie na kartce,
2) wpisać kod programu umożliwiający wczytanie elementów tablicy (macierzy),
3) wyświetlić na ekranie tablicę i jej wyznacznik.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 5

Napisz program, który zsumuje dwie tablice 2x2.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) znaleźć informacje, w jaki sposób dodaje się tablice,
2) wpisać kod programu umożliwiający wczytanie elementów tablicy (macierzy) i wykonanie działania,
3) wyświetlić na ekranie wszystkie trzy tablice.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 6

Znając definicje tablic, ciągu znakowego i zasadę działania funkcji strcpy, napisz własną implementację funkcji strcpy.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) znaleźć i przeczytaj uważnie opis wbudowanej funkcji strcpy,
2) naszkicować algorytm działania funkcji strcpy,
3) wpisać własny kod,
4) przetestować działanie funkcji.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 7

Napisz program, który będzie sprawdzał, czy wprowadzony przez użytkownika wyraz jest palindromem. Wykorzystaj do budowy programu funkcje, tablice i pętle.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) narysować algorytm sprawdzający warunki określone w zadaniu,
2) zaplanować sposób zakończenia działania programu, czyli wyjścia z pętli,
3) napisać kod wczytujący wyraz,
4) napisać kod sprawdzający, czy wyraz spełnia zadane warunki,
5) zamknąć wykonanie w pętli,
6) sprawdzić działanie programu.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 8

Bardzo dobrym sposobem testowania programów wykonujących operacje arytmetyczne na tablicach jest wypełnienie ich wartościami losowymi. Napisz program, który zapisuje liczby losowe do tablicy. Następnie oblicza sumę wszystkich elementów tablicy.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) planując wykonanie ćwiczenia należy wziąć pod uwagę sposób określania wymiaru i rozmiaru tablicy,
2) zbudować funkcję wypełniającą tablicę liczbami,
3) wyświetlić na ekranie tablicę,

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

4.9.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) zdefiniować pojęcie tablicy?	☐	☐
2) zadeklarować typ i wymiar tablicy?	☐	☐
3) przypisać wartości początkowe tablicy?	☐	☐
4) wykonać operacje pobrania i przypisania wartości elementom tablicy?	☐	☐
5) zdefiniować ciąg znaków w języku C++?	☐	☐
6) wykonać operacje na ciągach znaków?	☐	☐

4.10. Struktury, unie, pola bitowe

4.10.1. Materiał nauczania

Struktura, to zmienne pogrupowane w logiczną całość, reprezentowane przez zmienną strukturalną. Zmienna ta jest najprościej ujmując nazwą grupy zmiennych. Grupowanie zmiennych w struktury czyni kod bardziej przejrzystym:

```
struct
{
  char imie[20];
  int wiek;
  float waga;
} osoba;
```

Powyższy kod definiuje zmienną `osoba`. Zmienna ta jest typu strukturalnego. Struktury w języku C++ są odpowiednikiem rekordów z języka Pascal, ale wzbogaconych o funkcje. Za pomocą operatora kropki uzyskuje się dostęp do elementów składowych struktury:

```
osoba.wiek = 24;
osoba.waga = 56.5;
strcpy(osoba.imie, „Dominik”);
```

Istnieje możliwość deklarowania struktur zagnieżdżonych czyli takich, w których w pewnej strukturze znajduje się definicja kolejnej. Oto przykład:

```
struct
{
  char imie[20];
  int wiek;
  float waga;
  struct
  {
    float pensja;
  }praca;
}osoba;

osoba.praca.pensja = 860.00;
```

Możliwe jest również najpierw deklarowanie struktury, a dopiero w dalszej części kodu jej wykorzystanie. Wymaga to nadania nazwy strukturze:

```
struct produkty
{
  char nazwa[30];
  float cena;
} ;

produkty jablko;
produkty telewizor;
```

Podobną budowę, ale inne znaczenie, mają unie. Unia jest typem zmiennej, który może przechowywać jedną wartość w zmiennych wchodzących w jej skład. Jeżeli składowa struktury, zawierać jedne albo drugi typ zmiennej to musimy skorzystać z unii. Przykład zapisu unii:

```
  Union
{
          Kobieta: int;
          Mezczyzna: int;
} plec;
```

Gdy wykorzystamy powyżej zapisaną unię w strukturze, kod będzie miał postać:

```
struct
{
  char imie[20];
  int wiek;
  float waga;
  struct
  {
    float pensja;
  }praca;
  union
  {
    kobieta: int;
    mezczyzna: int;
  }plec;
}osoba;
```

Tak utworzona zmienna strukturalna z unią wewnątrz będzie można wykorzystać do zapisywania danych osobowych. Będzie można określić płeć jako mężczyzna albo kobieta:

```
osoba.plec.mezczyzna = 1;
```

albo

```
osoba.plec.kobieta = 1;
```

Nie będzie można przypisać wartości jednej i drugiej zmiennej. Tylko jedna z nich może mieć wartość.

W miejscu deklarowania struktury lub unii w kodzie, wykorzystuje się też często pola bitowe. Pod tą definicją kryje się zawężanie wielkości bitowej typu. Można powiedzieć, że wykorzystywanie pola bitowego może zmniejszyć ilość bitów, jaką zajmuje zmienna w pamięci. Na przykład, gdy zmienną typu int zmniejszy się do 1 bitu to powstanie zmienna, która może przyjmować jedną z dwóch wartości: 0 lub 1. Zmniejszając zmienną int do 8 bitów, czyli jednego bajta, uzyskuje się zmienną o wartościach od -128 do 127. Zmianę długości pola bitowego zmiennej deklaruje się zaraz za jej nazwą, zgodnie ze składnią:

```
int mezczyzna : 1;
```

4.10.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.

1. Czym są struktury w języku C++ i jakie jest ich zastosowanie?
2. Jaki jest układ definicji struktury?
3. Czym są unie i jak się je stosuje?
4. W jakim celu wykorzystuje się pola bitowe?

4.10.3. Ćwiczenia

Ćwiczenie 1

Zbuduj trzy struktury, które gromadzić będą odpowiednio dane z dowodu osobistego, prawa jazdy i dowodu rejestracyjnego pojazdu.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:

1) zanotować dane, które powinny zostać zgromadzone dla każdego z dokumentów,
2) zaproponować strukturę do każdego dokumentu,
3) podzielić się uwagami z pozostałymi uczniami w grupie,
4) utworzyć struktury.

Wyposażenie stanowiska pracy:

– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 2

Napisz program, który będzie gromadził dane o pracownikach. Każdy pracownik powinien być opisany przez zmienne ujęte w strukturze. Dane gromadzone przez system to: imię, nazwisko, dzień, miesiąc i rok urodzenia, staż pracy w latach, zawód wyuczony i wykonywany, dział w jakim pracuje (produkcja, pakowanie, sprzedaż, administracja), pensja. Pracowników w firmie może być maksimum 50. Jaki mechanizm zastosujesz?

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) zaplanować strukturę, która będzie przechowywała dane,
2) zaplanować narzędzie wprowadzania danych,
3) zaplanować narzędzie wyświetlania danych,
4) sprawdzić działanie programu.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

4.10.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) zdefiniować strukturę?	☐	☐
2) poprawnie zapisać elementy składowe struktury?	☐	☐
3) wykorzystać strukturę w programie?	☐	☐
4) zdefiniować unię i zastosować ją?	☐	☐
5) poprawnie wykorzystać unie w strukturze?	☐	☐
6) wykorzystać pole bitowe?	☐	☐

4.11. Wskaźniki, referencje

4.11.1. Materiał nauczania

Zmienna to umowny adres w pamięci systemu operacyjnego w chwili wykorzystania zmiennej, procesor modyfikuje jej nazwę na adres, aby dostać się do danych. W programowaniu zachodzi czasami potrzeba bezpośredniego odwołania się do pamięci. Ma to miejsce w przypadku, gdy trzeba skorzystać ze zmiennych dynamicznych, czyli takich zmiennych, które nie zostały zadeklarowane na początku programu, ale powstają w trakcie jego działania.

Wskaźnik to taka zmienna, która przechowuje adres pamięci, a nie wartość. Na przykład:

```
int rok=2005;
int *wskaznik;
wskaznik =&rok;
```

Pierwsza instrukcja przypisuje zmiennej rok wartość 2005. Druga deklaruje wskaźnik o nazwie `wskaznik`. Wskaźnik nie jest typu int, ale przechowuje adres pamięci o rozmiarze 4 bajtów, w którym znajdzie się wartość typu int. Trzecia instrukcja za pomocą operatora przypisania i symbolu & zapisuje we wskaźniku adres pamięci, w której znajduje się zmienna rok. Przykład programu ilustrującego działanie wskaźników:

```
#include <iostream.h>
#include <conio.h>

int main()
```

```
{
  int rok=2005;
  int *wskaznik;
  wskaznik=&rok;
  cout << "rok=" << rok << endl;
  cout << "*wskaznik=" << *wskaznik << endl;
  *wskaznik=2006;
  cout << "rok=" << rok << endl;
  cout << "*wskaznik=" << wskaznik << endl;
  getch();
  return 0;
}
```

Wskaźniki mogą również odnosić się do funkcji. Oto przykład:

```
#include <iostream.h>
#include <conio.h>

int dodawanie (int a, int b){ return (a+b); }

int odejmowanie (int a, int b){ return (a-b); }

int (*minus)(int,int) = odejmowanie;

int operacja (int x, int y, int (*funkcjawywolana)(int,int))
{
  int wynik = (*funkcjawywolana)(x,y);
  return (wynik);
}

int main ()
{
  int zm1,zm2;
  zm1 = operacja(2, 2, dodawanie);
  zm2 = operacja(2, 2, odejmowanie);
  cout << zm1 << endl;
  cout << zm2;
  getch();
  return 0;
}
```

W języku C++ wykorzystuje się często referencje. Zmienne typu referencyjnego są podobne do wskaźników. Różnią się tym, że adres na który wskazują nie może zostać zmieniony. Stosuje się je najczęściej w funkcjach oraz aby utworzyć kilka zmiennych o różnych nazwach, które w efekcie będą wskazywać na tą samą wartość. Deklaruje się w następujący sposób:

```
int a;
int& referencja=a;
```

W momencie tworzenia zmiennej referencyjnej należy od razu przypisać zmienną, do której ma się odnosić. Tego przypisania nie można później zmienić. Dzięki takiej deklaracji można przypisać zmiennej wartość na dwa sposoby:

```
a = 10;
referencja = 10;
```

4.11.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz czy jesteś przygotowany do wykonania ćwiczeń.
1. Podaj definicję wskaźnika.
2. Jak odwołać się do adresu pamięci, a jak do wartości ukrytej pod wskaźnikiem?
3. Jak odczytać adres zmiennej w pamięci?
4. Czym jest referencja i kiedy się ją wykorzystuje?

4.11.3. Ćwiczenia

Ćwiczenie 1

Napisz program, który wylosuje elementy tablicy 10-cio elementowej typu int. Następnie wyświetl na ekranie adres w pamięci poszczególnych elementów tablicy i ich wartość. Wyświetl adres zmiennej, pod którą przechowywana jest tablica.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) napisać kod losujący wartości elementów tablicy,
2) zaplanować wyświetlanie tablicy na ekranie,
3) wyświetlić adresy elementów tablicy,
4) dopisać kod pokazujący adres tablicy.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

4.11.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) zdefiniować wskaźnik?	☐	☐
2) podać przykład zastosowania wskaźników?	☐	☐
3) odczytać adres zmiennej w pamięci?	☐	☐
4) pobrać wartość ukrytą pod adresem pamięci przechowywanym przez wskaźnik?	☐	☐
5) utworzyć referencję i wykorzystać ją w programie?	☐	☐

4.12. Zarządzanie pamięcią

4.12.1. Materiał nauczania

Język C++ jest ogromnie użyteczny wszędzie tam, gdzie potrzebne są zmienne dynamiczne, to znaczy takie, które powstają w trakcie wykonania programu. Rodzaj i ilość zależą od innych danych lub procesów w programie. Jest to rozwiązanie dużo lepsze niż tworzenie tablicy o dużych wymiarach, ponieważ tablica mimo wszystko może okazać się za mała. W mniejszym stopniu obciąża się też pamięć komputera.

Aby skorzystać ze zmiennych dynamicznych potrzebne będą wskaźniki i dwa nowe operatory. Operator `new`, który rezerwuje miejsce w pamięci i operator `delete`, który zwalnia miejsce w pamięci. Należy wspomnieć o bardzo poważnym i częstym błędzie. Zawsze trzeba pamiętać, aby zwalniać zajętą pamięć, w przeciwnym wypadku, wraz z tworzeniem nowych zmiennych powstanie problem określany „wyciekiem pamięci". Problem ten został rozwiązany w językach Java i w środowisku .NET. Tam, specjalny proces systemowy zajmuje się oczyszczaniem pamięci.

Przykład:

```
#include <cstdlib>
#include <iostream.h>
#include <conio.h>

int main ()
{
  int *tablica;
  tablica = new int[4];

  for (int x=0; x<5; x++)
  {
    tablica[x] = x;
    cout << tablica[x] <<endl;
  }

  getch();
  return 0;
}
```

W pierwszej instrukcji funkcji głównej występuje deklaracja nowego wskaźnika. W następnej instrukcji wskaźnikowi przypisywany jest adres nowo utworzonej tablicy liczb całkowitych o wymiarze 4. W dalszej części następuje dynamiczne odwołanie do elementów tablicy w taki sam sposób, jakby była to zwykła tablica. Program warto wzbogacić o sprawdzenie, czy udało się utworzyć nową tablicę w pamięci:

```
int *tablica;
tablica = new int[4];
if (tablica == NULL)
{
   Komunikat o błedzie
};
```

Powyższa instrukcja warunkowa sprawdza, czy wskaźnik tablica ma zdefiniowaną wartość, czy jest nie zdefiniowany (`NULL`), a więc nie udało się przydzielić pamięci. Na koniec pozostaje dodać bardzo ważny element: usunięcie danych z pamięci. W tym celu stosuje się instrukcję `delete`. Jeżeli po instrukcji `delete` zapisze się nazwę wskaźnika, to usunięty zostanie pojedynczy element. Aby skasować tablicę, przed nazwą wskaźnika należy umieścić puste nawiasy kwadratowe:

```
delete [] tablica;
```

4.12.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.
1. Czym jest zarządzanie pamięcią?
2. Jakie zagrożenie niesie nie kasowanie zmiennych dynamicznych, gdy nie są już dostępne?
3. Jak tworzy się, modyfikuje i kasuje zmienne dynamiczne?

4.12.3. Ćwiczenia

Ćwiczenie 1

Napisz program, który będzie pobierać od użytkownika liczby tak długo, aż wpisana zostanie wartość zero. Pobrane liczby program powinien zapisywać w tablicy dynamicznej, a następnie wyświetlić posortowane rosnąco.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) zadeklarować dynamiczną tablicę o rozmiarze podanym przez użytkownika w momencie wykonywania programu,
2) dopisać funkcję pobierania danych,
3) sprawdzić działanie programu.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 2

Podczas tworzenia bazy danych nie wiadomo ile będzie ona zawierać rekordów. Napisz program, który zbiera podstawowe dane o samochodzie i przechowuje je w tablicy dynamicznej. Elementami tablicy powinny być rekordy. Interfejs programu powinien mieć opcje dopisywania, modyfikowania i kasowania danych.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) zaplanować postać struktury przechowującej dane,
2) zdefiniować dynamiczną tablicę,
3) dopisać kod wprowadzania danych. Pamiętaj o sprawdzaniu dostępnej pamięci,
4) dopisać kod wyświetlania danych i sprawdź funkcjonowanie,
5) dopisać kod modyfikowania danych,
6) dopisać kod usuwania danych,
7) sprawdzić działanie programu.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

4.12.4. Sprawdzian postępów

Czy potrafisz:	Tak	Nie
1) zdefiniować zmienną dynamiczną?	☐	☐
2) używać operatora new?	☐	☐
3) odwoływać się do zmiennych dynamicznych?	☐	☐
4) usuwać zmienne dynamiczne?	☐	☐
5) wyjaśnić pojęcie „wycieków pamięci”?	☐	☐

4.13. Definicje klasy. Atrybuty i metody. Hermetyzacja.

4.13.1. Materiał nauczania

Klasy są to struktury wzbogacone o metody, czyli funkcje wykonujące konkretne zadania. Są to pogrupowane zmienne wraz z metodami, które na tych zmiennych wykonują operacje. Klasa, która zostanie zaimplementowa tworzy obiekt. Dokładniej, obiekt jest implementacją klasy, czyli realnym przypadkiem definicji klasy. Klasy różnią się od struktur jeszcze jednym bardzo istotnym elementem. Jej elementy składowe mają określony zasięg. Oznacza to, że można zdefiniować, czy dana zmienna lub metoda ma być dostępna z zewnątrz, czy tylko z wewnątrz klasy. Typowa definicja klasy zaczyna się od słowa kluczowego `class`, po którym następuje nazwa klasy i w nawiasach klamrowych definicja zmiennych (zwanych atrybutami klasy) oraz funkcji (zwanych metodami klasy). Przykład:

```
class Samochod
{
   public:
      float pojemnosc;
      char* marka;
      char* model;
      float zbiornikPaliwa;

      void WyswietlNazwe()
      {
          cout << "Samochod: " << marka << " " << model;
      }
};
```

Powyższa prosta definicja klasy zawiera słowo kluczowe `public`. Informuje ono, że wszystkie definicje, które wystąpią poniżej są publiczne, czyli można odwoływać się do nich bezpośrednio z zewnątrz klasy:

```
int main()
{
          Samochod skoda;
```

```
            strcpy(skoda.marka, ”Skoda”);
            strcyp(skoda.model, ”Fabia”);
            skoda.WysietlNazwe();
            getch();
            return 0;
    }
```

W wielu deklaracjach klas występuje słowo kluczowe `private`. Określa ono, że elementy będą dostępne jedynie z wewnątrz klasy, czyli z innych funkcji danej klasy, a niedostępne z zewnątrz. Na przykład:

```
class Samochod
{
   private:
        float pojemnosc;
        char* marka;
        char* model;
        float zbiornikPaliwa;
   public:
        void WyswietlNazwe()
        {
            cout << "Samochod: " << marka << " " << model;
        }
        float getPojemnosc()
        {
            return pojemnosc;
        }
        void setPojemnosc(float poj)
        {
            pojemnosc = poj;
        }
        char* getMarka()
        {
            return marka;
        }
        void setMarka(char* mark)
        {
            marka = strcpy(mark);
        }
        char* getModel()
        {
            return model;
        }
        void setModel(char* mod)
        {
            model = strcpy(mod);
        }
        float getZbiornikPaliwa()
        {
            return zbiornikPaliwa;
        }
        void setZbiornikPaliwa(float ilosc)
        {
            zbiornikPaliwa = ilosc;
        }
 };
```

Zdefiniowana powyżej klasa posiada atrybuty, które są prywatne i funkcje, które są publiczne. Kod ten jest również przykładem klasy, która jest zgodna z zasadą hermetyzacji. Oznacza to, że wszystkie zmienne (atrybuty klasy) są prywatne i tylko za pomocą odpowiednich metod można zmieniać i pobierać ich wartość. w programie, każdej zmiennej

odpowiadają dwie metody: jedna z przedrostkiem `set` (ustaw), a druga z przedrostkiem `get` (pobierz). To rozwiązanie może się wydawać niepotrzebne, bo do każdej zmiennej tworzy się dwie funkcje, zamiast zmienić jej status na publiczny i móc modyfikować je z zewnątrz. Ale taki sposób jest dużo bezpieczniejszy, ponieważ to nie programista bezpośrednio zmienia jej wartość, ale funkcja. Bardzo często funkcja ta ma bardziej skomplikowany kod niż w powyższym przykładzie i wywołuje inne działania i metody klasy. Często modyfikuje się kilka wartości. Stosując taką konstrukcję programista jest bliżej warstwowego podejścia do pisania aplikacji i dzieli system na wyraźne elementy niezależne od siebie, ale komunikujące się ze sobą w ściśle określony sposób. Przykładem może być warstwa interfejsu, obliczeniowa i danych.

4.13.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.
1. Czym jest klasa obiektu, a czym jej implementacja?
2. Jak tworzy się klasy?
3. Jakie są rodzaje zasięgów zmiennych i metod?
4. Jak powinna wyglądać klasa zgodna z zasadą hermetyzacji?

4.13.3. Ćwiczenia

Ćwiczenie 1

Znajdź w Internecie dokładniejsze informacje na temat warstw w aplikacji. Dlaczego ten sposób programowania jest tak istotny?

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) wyszukać strony zawierające informacje o projektowaniu z wykorzystaniem warstw,
2) wynotować różnice pomiędzy projektowaniem tradycyjnym, a warstwowym,
3) sformułować wnioski,
4) podzielić się wnioskami z innymi uczniami w grupie.

Wyposażenie stanowiska pracy:
– komputer z zainstalowaną przeglądarką internetową.

Ćwiczenie 2

Napisz klasę, która będzie odpowiednikiem parkingu samochodowego. Klasa ta powinna zwierać dane o ilości samochodów i ilości wolnych miejsc. Każdy samochód powinien być opisany przez model, markę i numer rejestracyjny. Klasa musi mieć metody służące do przyjmowania samochodu na parking i jego zwrotu.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) napisać klasę,
2) zdefiniować atrybuty klasy,

3) zdefiniować metody klasy,
4) zastanowić się, czy należy skorzystać z podklas.

Wyposażenie stanowiska pracy:

– komputer z zainstalowanym kompilatorem języka C++.

4.13.4. Sprawdzian postępów

Czy potrafisz:	Tak	Nie
1) zdefiniować klasę i jej składowe?	☐	☐
2) tworzyć obiekt na podstawie klasy?	☐	☐
3) wykorzystać atrybuty publiczne i prywatne?	☐	☐
4) odwzorować elementy otaczającego świata w postaci klasy?	☐	☐

4.14. Konstruktory i destruktory klasy

4.14.1. Materiał nauczania

Klasy mogą posiadać specjalne metody, tak zwane konstruktory i destruktory. Konstruktor wywoływany jest w momencie tworzenia obiektu. Jest to bardzo wygodny mechanizm inicjowania składowych klasy. Za jego pomocą można zaprogramować działania, które są wykonywane w chwili tworzenia obiektu. Może to być na przykład przypisanie wartości domyślnych i wywołanie funkcji wewnętrznych. Korzystając z konstruktora można mieć pewność, że klasa została prawidłowo zainicjowana. Destruktor z kolei jest uruchamiany w momencie kiedy obiekt ma zostać usunięty. Może wykonać funkcje wewnętrzne, które powinny być wykonane w ostatniej chwili (zapis danych do pliku) lub skasować zmienne.

Przykład klasy z konstruktorem i destruktorem:

```
#include <cstdlib>
#include <iostream.h>
#include <conio.h>

class Prostokat {
  private:
    int *szerokosc, *wysokosc;
  public:
    Prostokat(int a, int b)
    {
        szerokosc = new int;
        wysokosc = new int;
        *szerokosc = a;
        *wysokosc = b;
    }
    ~Prostokat()
    {
        delete szerokosc;
        delete wysokosc;
    }
    int obszar()
    {
```

```
            return (*szerokosc * *wysokosc);
        }
    };

    int main () {
      Prostokat prost (10,20);
      Prostokat prost2 (20,20);
      cout << "pole prostokata: " << prost.obszar() << endl;
      cout << "pole prostokata2: " << prost2.obszar() << endl;
      getch();
      return 0;
    }
```

Aby utworzyć konstruktor, tworzy się metodę o nazwie identycznej jak nazwa klasy. Na przykład metoda: `Prostokat(int a, int b)`. Tworząc destruktor postępuje się podobnie, ale nazwę poprzedza się znakiem tyldy (~). Konstruktorów może być kilka. Aby utworzyć więcej niż jeden konstruktor stosuje się metodę przeciążania funkcji. Destruktor może być tylko jeden.

4.14.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.

1. Podaj definicję konstruktora i destruktora.
2. Jak się tworzy konstruktory i destruktory?
3. Ile może być konstruktorów, a ile destruktorów w jednej klasie?

4.14.3. Ćwiczenia

Ćwiczenie 1

Do klasy z poprzedniego rozdziału, (obsługa parkingu) dopisz odpowiednie konstruktory i destruktory.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:

1) przejrzeć kod klasy obsługi parkingu,
2) dopisać konstruktor domyślny,
3) dopisać konstruktor z parametrem,
4) dopisać destruktor.

Wyposażenie stanowiska pracy:

– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 2

Napisz klasę, która będzie mogła gromadzić informacje o miejscach w samolocie. Konstruktor powinien definiować maksymalną ilość miejsc, a destruktor zwalniać wszystkie miejsca. Klasa powinna mieć zmienne służące do przechowywania podstawowych danych o pasażerach (imię, nazwisko, płeć, klasa – biznes / ekonomiczna). Klasa powinna posiadać metody odpowiedzialne za przyjęcie pasażera na pokład i zwalnianie pojedynczego miejsca na pokładzie.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) napisać klasę,
2) zdefiniować atrybuty klasy,
3) zdefiniować metody klasy,
4) napisać konstruktor,
5) napisać destruktor.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

4.14.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) zdefiniować konstruktor?	☐	☐
2) zdefiniować destruktor?	☐	☐
3) wykorzystać w programie klasy z konstruktorem i destruktorem?	☐	☐
4) utworzyć konstruktory przeciążone?	☐	☐

4.15. Funkcje i klasy zaprzyjaźnione

4.15.1. Materiał nauczania

Funkcje zaprzyjaźnione to pomosty pomiędzy klasami. Nie są to metody danej klasy, ale mogą odwoływać się do jej zmiennych, w tym też atrybutów prywatnych. Funkcje zaprzyjaźnione deklaruje się globalnie w klasie, która ma z nich korzystać:

```
#include <iostream.h>
#include <conio.h>

class Dzialanie
{
    int y;
    int x;
  public:
    Dzialanie(){ y=2; x=3;}
    friend void dodaj(Dzialanie &object);
    int getY(){ return y;}

};

void dodaj(Dzialanie &object)
{
   object.y+=object.x;
}

int main()
{
```

```
    Dzialanie obiekt;
    cout << obiekt.getY() << endl;
    dodaj(obiekt); // dostep do private:
    cout << obiekt.getY() << endl;
    getch();
    return 0;
}
```

W języku C++ można tworzyć klasy zaprzyjaźnione umieszczając w jednej z klas słowo kluczowe `friend class` i nazwa klasy zaprzyjaźnionej. Oto przykład, w którym klasa `Prostokat` jest klasą zaprzyjaźnioną klasy `Kwadrat`, czyli `Prostokat` ma dostęp do zmiennych prywatnych i chronionych klasy `Kwadrat`:

```
#include <iostream.h>
#include <conio.h>
class Kwadrat;

class Prostokat {
  private:
    int szerokosc, wysokosc;
  public:
    void zamien (Kwadrat kw);
    int pole (void) {return (szerokosc * wysokosc);}
};

class Kwadrat{
  private:
    int sciana;
  public:
    friend class Prostokat;
    void setSciana(int szerokosc){sciana=szerokosc;}
};

void Prostokat::zamien(Kwadrat kw)
{
     szerokosc = kw.sciana;
     wysokosc = kw.sciana;
}

int main () {
  Kwadrat kwadrat;
  Prostokat prostokat;
  kwadrat.setSciana(10);
  prostokat.zamien(kwadrat);
  cout << "Pole wynosi: " << prostokat.pole();
  getch();
  return 0;
}
```

Za instrukcjami dołączania znajduje się zapis `class Kwadrat`. Jest to na pierwszy rzut oka bardzo dziwny zapis, który może wydawać się błędny lub zbędny – przecież poniżej w kodzie znajduje się zadeklarowana klasa `Kwadrat`. Jest to predefinicja klasy. Jest niezbędna, ponieważ klasa `Prostokąt` wykorzystuje klasę `Kwadrat` zanim wystąpi pełna definicja klasy. Zapowiedź, że klasa `Kwadrat` będzie dalej zdefiniowana. z tych samych względów metoda zmian klasy `prostokat` musi być zadeklarowana poza klasą.

Zapis `void Prostokat::zamien(Kwadrat kw)` służy do deklarowania metody na zewnątrz klasy. Taka definicja składa się z typu funkcji, następnie nazwy klasy, podwójnego dwukropka i nazwy metody.

4.15.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.
1. Czym jest funkcja zaprzyjaźniona?
2. Czym jest klasa zaprzyjaźniona?
3. Czy klasa może mieć więcej niż jedną funkcję zaprzyjaźnioną?
4. Czy jedna funkcja może być zaprzyjaźniona z więcej niż jedną klasą?
5. Jakie mają zastosowanie klasy i funkcje zaprzyjaźnione?

4.15.3. Ćwiczenia

Ćwiczenie 1

Napisz program w którym jedna funkcja będzie zaprzyjaźniona z dwiema różnymi klasami i będzie wykonywała obliczenia na zmiennych tych klas.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) zdefiniuj dwie klasy,
2) w jednej z klas zaproponować funkcje i uczynić ją zaprzyjaźnioną dla drugiej klasy,
3) zapisz kod w drugiej klasie wykorzystujący tą funkcję zaprzyjaźnioną.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 2

Napisz klasę figury, która będzie posiadała zaprzyjaźnione klasy prostokąt, kwadrat, trójkąt.

Sposób wykonania:

Aby wykonać ćwiczenie powinieneś:
1) zdefiniować klasy,
2) napisać klasę i uczynić ją zaprzyjaźnioną z inną klasą.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

4.15.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) zdefiniować funkcje zaprzyjaźnioną?	☐	☐
2) zdefiniować klasę zaprzyjaźnioną?	☐	☐
3) skorzystać z funkcji i klas zaprzyjaźnionych?	☐	☐
4) wymienić zastosowania funkcji zaprzyjaźnionych?	☐	☐

4.16. Dziedziczenie

4.16.1. Materiał nauczania

Dziedziczenie jest jedną z podstawowych zasad programowania obiektowego. Umożliwia tworzenie obiektów potomnych, których struktura bazuje na innych obiektach. Oznacza to, że mając jakiś obiekt, na przykład telefon, można utworzyć kolejny, który odziedziczy funkcje podstawowe i zostanie wzbogacony o nowe. Mając obiekt można utworzyć klasę pochodną. Przykład:

```
class pojazd
{
  public:
    float zuzycie;
    int pojemnoscBaku;
    int zasieg;
    void obliczZasieg(){   zasieg = pojemnoscBaku / zuzycie *
100;}
};

class samochod: public pojazd
{
public:
  int iloscMiejsc;
};

class ciezarowka: public pojazd
{
public:
  int iloscMiejsc
  float ladownosc;
  int zaladowany;
  void zaladuj(){ zaladowany = 1;}
  void rozladuj(){ zaladowany = 0;}
};
```

Klasy powstałe dzięki dziedziczeniu są pełnowartościowymi klasami. Oznacza to, że obiekt, który zostanie utworzony na bazie klasy pochodnej, będzie miał pełną funkcjonalność. Nie będzie żadnych różnic w korzystaniu z takiego obiektu w kodzie programu. Do jego składowych można się odwołać w znany sposób, niezależnie od tego, czy zostały one zdefiniowane, czy odziedziczone.

4.16.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.

1. Jak wykorzystuje się dziedziczenie?
2. Dziedziczenie – upraszcza, czy komplikuje programowanie?

4.16.3. Ćwiczenia

Ćwiczenie 1

Napisz program, który będzie tworzył trzy obiekty: prostokąt, trójkąt i kwadrat, na podstawie odpowiednich klas. Klasy poszczególnych obiektów powinny wywodzić się z jednej klasy bazowej: figura, która będzie miała tylko jeden atrybut typu int: pole. Zadaniem każdej z klas pochodnych jest przechowywanie danych potrzebnych do obliczenia pola i metodę liczącą to pole.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) zdefiniować klasę bazową,
2) zdefiniować pozostałe klasy wykorzystując klasę bazową,
3) zaprogramować atrybuty i metody klasy.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

4.16.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) wykorzystać dziedziczenie?	☐	☐
2) wyjaśnić zalety dziedziczenia?	☐	☐
3) wymienić zastosowania dziedziczenia?	☐	☐

4.17. Funkcje operatorowe, przeciążanie operatorów

4.17.1. Materiał nauczania

Język programowania C++ daje możliwość tworzenia swoich własnych operatorów. Jest to przydatny mechanizm, który umożliwia zaprogramowanie nowych działań na zmiennych, lub obiektach. Deklaracja nowego operatora powinna wyglądać następująco:

```
Typ_zwracany operator symbol_znaku (parametry);
```

Przykład, ilustrujący klasę opisującą macierz o wymiarach 2 na 2 i operator dodawania dwóch macierzy:

```
#include <cstdlib>
#include <iostream.h>
#include <conio.h>

class Macierz2_2 {
  public:
    int x [2][2];
    Macierz2_2 ()
    {
        Macierz2_2(0,0,0,0);
    };
    Macierz2_2 (int xx11, int xx12, int xx21, int xx22)
    {
        x[0][0] = xx11;
        x[0][1] = xx12;
        x[1][0] = xx21;
        x[1][1] = xx22;
    };
    Macierz2_2 operator + (Macierz2_2);
};
```

```
Macierz2_2 Macierz2_2::operator+ (Macierz2_2 param) {
  Macierz2_2 mac;
  mac.x[0][0] = x[0][0] + param.x[0][0];
  mac.x[0][1] = x[0][1] + param.x[0][1];
  mac.x[1][0] = x[1][0] + param.x[1][0];
  mac.x[1][1] = x[1][1] + param.x[1][1];
  return (mac);
}

int main () {
  Macierz2_2 a(1,2,3,4);
  Macierz2_2 b(2,2,2,2);
  Macierz2_2 c;
  c = a + b;
  cout << "[" << c.x[0][0] << " " << c.x[0][1] << "]" << endl;
  cout << "[" << c.x[1][0] << " " << c.x[1][1] << "]" << endl;
  getch();
  return 0;
}
```

Deklaracja operatora to specjalna funkcja, która za obiekt bazowy uznaje obiekt stojący po lewej stronie operatora. Obiekt przesłany do operatora jako argument, to obiekt stojący po prawej stronie. Domyślnie w C++ nie ma zdefiniowanego operatora do sumowania ze sobą dwóch obiektów typu `Macierz2_2`. Dlatego tworząc nową klasę `Macierz2_2` należy utworzyć również dla niej operator dodawania.

4.17.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.

1. Jak tworzy się operatory?
2. Jak definiuje się operatory dla obiektów?
3. Ile argumentów można przesłać do funkcji operatorowej?
4. Czy można nadpisać operator domyślny, czyli zdefiniowany przez programistów tworzących język C++?

4.17.3. Ćwiczenia

Ćwiczenie 1

Wzbogać klasę Macierz2_2 o operatory dodawania, odejmowania i operator, który zwraca wymiar macierzy.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:

1) przejrzeć klasę macierz,
2) zdefiniować operator dodawania,
3) zdefiniować operator odejmowania,
4) zdefiniować operator zwracający wymiar macierzy,
5) zdefiniować dwie macierze na podstawie tej klasy i sprawdzić działanie operatorów.

Wyposażenie stanowiska pracy:

– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 2

Napisz klasę, która przechowuje imię i nazwisko oraz wysokość wynagrodzenia. Napisz operator, który umożliwi sumowanie ze sobą wynagrodzeń pochodzących z obiektów powstałych na bazie klasy.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) zdefiniować klasę,
2) zaprogramować operatory,
3) utworzyć tablicę 10 osób i wypełnić danymi,
4) zsumować wynagrodzenia korzystając z operatora.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

4.17.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) zdefiniować operatory?	☐	☐
2) wyjaśnić na czym polega przeciążanie operatorów?	☐	☐
3) zaprogramować nowy operator?	☐	☐
4) przeładować operator już istniejący?	☐	☐

4.18. Polimorfizm

4.18.1. Materiał nauczania

Dzięki polimorfizmowi (zob. 4.2.1.) można odwołać się do atrybutów i metod klasy bazowej. Najlepiej zilustruje to poniższy przykład:

```
#include <iostream.h>
#include <conio.h>

class Figura {
  protected:
    int szerokosc, wysokosc;
  public:
    void setWartosci (int x, int y) { szerokosc=x; wysokosc=y;
}
  };

class Prostokat: public Figura {
  public:
    int Pole(void){ return (szerokosc * wysokosc); }
  };
```

```
class Trojkat: public Figura {
  public:
    int Pole(void){ return (szerokosc * wysokosc / 2); }
  };

int main () {
  Prostokat prostokat;
  Trojkat trojkat;
  Figura *figura1 = &prostokat;
  Figura *figura2 = &trojkat;
  figura1->setWartosci (4,5);
  figura2->setWartosci (4,5);
  cout << prostokat.Pole() << endl;
  cout << trojkat.Pole() << endl;
  getch();
  return 0;
}
```

W ciele funkcji głównej tworzone są wskaźniki typu Figura (klasy bazowej). Wskazują one na obiekt `prostokat` i `trojkat`. Kolejne dwie instrukcje wywołują funkcję `setWartosci`, która zapamiętuje wymiary figur.

4.18.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.

1. Jak wykorzystuje się polimorfizm?
2. W jaki sposób polimorfizm upraszcza programowanie?

4.18.3. Ćwiczenia

Ćwiczenie 1

Napisz klasy `róża`, `stokrotka` i `paproć`, które dziedziczą z klas `kwiat-jednoroczny` i `kwiat-wieloroczny`. Te klasy dziedziczą z jednej klasy `Kwiat`. Klasa `Kwiat` ma atrybuty: `kolor` i `wysokość`. Klasa `Kwiat-wieloroczny` posiada atrybut zawierający informację w latach, jak długo kwiat rośnie. Klasy będące pochodną od kwiatów `jedno-` i `wielorocznych` powinny mieć atrybut `kolor-kwiatów`, `kolor-łodygi` i metodę `rośnij`, która będzie zwiększać wysokość kwiatów o 1cm. Każda klasa końcowa powinna mieć metodę `koniecRoku`, która zwiększa `wiek-kwiatu` o rok. Jeżeli `kwiat` będzie starszy niż `gatunek`, to powinien zwiędnąć.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:

1) zdefiniować klasy bazowe,
2) zdefiniować klasy pochodne,
3) wykorzystać polimorfizm do wywołania metody koniecRoku,
4) utworzyć 6 obiektów różnych typów.

Wyposażenie stanowiska pracy:

– komputer z zainstalowanym kompilatorem języka C++.

4.18.4. Sprawdzian postępów

Czy potrafisz:	Tak	Nie
1) zaprogramować klasy i obiekty tak, aby można było skorzystać z polimorfizmu?	☐	☐
2) wyjaśnić jakie są wady, a jakie zalety korzystania z polimorfizmu?	☐	☐

4.19. Szablony klas i funkcji

4.19.1. Materiał nauczania

Szablony umożliwiają definiowanie funkcji w sytuacji, gdy zachodzi potrzeba, by funkcja operowała na różnych typach zmiennych, a więc zwracała wartości różnego typu. Szablon jest w pewnym sensie wzorcem funkcji. Dopiero w momencie jego wykorzystania deklaruje się typ argumentów lub funkcji. Wykorzystując ten mechanizm można deklarować funkcje, które będą mogły obsłużyć różne argumenty, bez potrzeby stosowania metody przeładowania funkcji. Definicja szablonu funkcji wygląda następująco:

```
template <class identyfikator> deklaracja_funkcji;
template <typename identyfikator> deklaracja_ funkcji;
```

Pierwsza instrukcja deklaruje szablon funkcji, która ma zwracać obiekt dowolnej klasy. Druga zaś deklaruje szablon funkcji, która zwraca zmienną dowolnego typu.

Poniżej znajduje się przykład szablonu funkcji, która zwraca większy element dowolnego typu porządkowego, bez wskazywania na jakikolwiek typ:

```
template <class TYP>
TYP Maksymum(TYP a, TYP b)
{
 return (a>b?a:b);
}
```

Wykorzystanie możliwości tak zdefiniowanej funkcji (szablonu) w programie, odbywa się poprzez wywołanie tej funkcji wraz z określeniem rodzaju szablonu:

```
nazwa_funkcji <typy_dla_szablonu> (paramtery);
```

Oto przykład programu wykorzystującego szablon funkcji:

```
#include <iostream.h>
#include <conio.h>

template <class TYP>
TYP Maks (TYP a, TYP b) {
  TYP result;
  result = (a>b)? a : b;
  return (result);
}

int main() {
  int x=1, y=2;
  int wynik1;
  long m=9, n=3;
  float wynik2;
```

```
  wynik1=Maks<int>(x,y);
  wynik2=Maks<long>(m,n);
  cout << wynik1 << endl;
  cout << wynik2 << endl;
  getch();
  return 0;
}
```

W funkcji `main` zostaje policzony `wynik1` na podstawie funkcji `Maks`. w momencie wykorzystania szablonu funkcji należy podać typ. W programie, za pierwszym razem jest to typ int a za drugim razem typ long.

W języku C++ można pisać również szablony klas, czyli definiować takie klasy, które nie będą miały z góry określonego jednego typu zmiennych. Będzie można definiować typ w miejscu wykorzystania szablonu klasy. Oto przykład:

```
#include <cstdlib>
#include <iostream>

template <class TYP>
class Tablica {
    TYP element1, element2, element3;
  public:
    Tablica (TYP e1, TYP e2, TYP e3)
      {element1=e1; element2=e2; element3=e3;}
    TYP Iloczyn ()
    {
        TYP result;
        result = element1 * element2 * element3;
        return result;
    }
};


int main () {
  Tablica <int> tab (1, 2, 3);
  cout << "Iloczyn: " << tab.Iloczyn();
  return 0;
}
```

W momencie deklarowania obiektu, czyli w pierwszej instrukcji funkcji main, następuje utworzenie obiektu na podstawie szablonu klasy.

4.19.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.

1. Czym są szablony klas i funkcji?
2. Jakie zastosowanie mają szablony klas i funkcji?
3. Jak zaprogramować szablon i poprawnie wykorzystać go w programie?

4.19.3. Ćwiczenia

Ćwiczenie 1

Napisz funkcję, która obliczy dowolną potęgę liczby dowolnego typu.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) napisać funkcję, która policzy potęgę typu całkowitego,
2) przepisać tę funkcję w taki sposób, by korzystała z szablonu.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

4.19.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) zdefiniować i zaprogramować szablon funkcji?	☐	☐
2) zdefiniować i zaprogramować szablon klasy?	☐	☐
3) wymienić sytuacje, w których zalecane jest stosowanie szablonów?	☐	☐

4.20. Strumienie i manipulatory

4.20.1. Materiał nauczania

Strumienie określają przepływ danych od źródła lub do celu. Źródła i cele mogą być jednego z kilku typów. Może to być klawiatura, pamięć, ekran, plik lub inne urządzenie zewnętrzne. W języku C++ wyróżnia się trzy podstawowe typy strumieni:
- wejściowe – strumienie które wczytują lub pobierają dane,
- wyjściowe – strumienie, które zapamiętują lub zapisują dane,
- uniwersalne – łączą w sobie cechy dwóch poprzednich, czyli potrafią zapisywać i odczytywać dane.

W poprzednich rozdziałach bardzo często nieświadomie wykorzystywane były strumienie. Miało to miejsce wszędzie tam, gdzie dane na ekran wyprowadzał strumień `cout` (który zastępuje konstrukcję `printf("%f",zmienna)` z języka C), lub dane z klawiatury pobierał strumień `cin`. Stosuje się dwa operatory:
- `<<` - operator wysyłania do strumienia,
- `>>` - operator pobierania ze strumienia.

Wykonując operacje na plikach stosuje się klasy:
- `ofstream` - zapis do pliku,
- `ifstream` - odczyt z pliku,
- `fstream` – łączy odczyt i zapis do pliku.

Przykład ilustrujący zapis do pliku:

```
ofstream stream("dane.txt");
stream<<"Tekst, który zostanie zapisany w pliku";
stream.close();
```

Ze strumieniami wykorzystuje się manipulatory. Umożliwiają one wywołanie pewnych funkcji bezpośrednio w strumieniu. Jednym z podstawowych i często używanych jest manipulator endl, który wstawia znak końca linii. Inne standardowe manipulatory dostępne w języku C++ to: dec, hex,. setprecision(int).

Manipulatory mogą być bezparametrowe:

```
ostream& manipulator(ostream&);
```

Poniżej przykład ilustrujący konstrukcję manipulatora, którego zadaniem jest przywrócić domyślne ustawienia trybu znakowego:

```
ostream& kolorDomyslny(ostream& sys)
{
 colorbuf *pds = (colorbuf*)sys.rdbuf();
 pds->setfore(default_foreground);
 pds->setback(default_background);

 return sys;
}
```

4.20.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.

1. Czym są strumienie i jak działają?
2. Jaka jest różnica pomiędzy strumieniami w języku C++, a obsługą ekranu lub pliku w języku C?
3. Czym są manipulatory i gdzie się je stosuje?

4.20.3. Ćwiczenia

Ćwiczenie 1

Napisz program, który zapisuje tabliczkę mnożenia do pliku.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:

1) zbudować funkcję, która zapisuje dane do pliku,
2) sprawdzić działanie,
3) dopisać kod, obliczający wartości występujące w jednym wierszu tabliczki mnożenia,
4) sprawdzić działanie programu, przeglądając zawartość pliku.

Wyposażenie stanowiska pracy:

– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 2

Napisz manipulator, który zakończy aktualną linię i na początku nowej napisze kolejny numer linii i dwukropek.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:

1) przemyśleć działanie manipulatora, o którym mowa w ćwiczeniu,
2) wpisać kod manipulatora,
3) sprawdzić działanie.

Wyposażenie stanowiska pracy:

– komputer z zainstalowanym kompilatorem języka C++.

4.20.4. Sprawdzian postępów

Czy potrafisz:	Tak	Nie
1) zdefiniować i wykorzystać strumienie?	☐	☐
2) określić różnice pomiędzy strumieniami w C++, a funkcjami z języka C?	☐	☐
3) zdefiniować manipulator?	☐	☐

4.21. Obsługa wyjątków

4.21.1. Materiał nauczania

Bardzo trudno jest napisać duży program, który będzie działał od razu bezbłędnie. Program, który będzie przewidywał wszystkie zachowania użytkownika. Ale możliwe jest napisanie programu, który będzie potrafił reagować na błędy i nieprawidłowości. Metoda ta nazywa się obsługą wyjątków. Wyjątek to sytuacja, gdy procesor nie może wykonać polecenia. Blok kodu, który ma być chroniony należy ująć w blok `try…catch`:

```
try {
  kod objęty obsługa wyjątku
  throw exception;
}
catch (type  exception)
{
  kod w razie wystąpienia błedu
}
```

Oto przykład programu z obsługa wyjątków:

```
// exceptions
#include <iostream.h>

int main () {
  char myarray[10];
  try
  {
    for (int n=0; n<=10; n++)
    {
      if (n>9) throw "Tablica poza zakresem";
      myarray[n]='z';
    }
  }
  catch (char * str)
  {
    cout << "Wyjątek: " << str << endl;
  }
  return 0;
}
```

4.21.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.
1. Czym są wyjątki?
2. Dlaczego należy stosować obsługę wyjątków?

4.21.3. Ćwiczenia

Ćwiczenie 1

Napisz program, który odczytuje i zapisuje tekst do pliku. Do programu dodaj obsługę wyjątków tak, by obsługiwała ona nieudany zapis i nieudany odczyt z pliku.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) napisać program, który zapisuje tekst do pliku,
2) dopisać obsługę wyjątku nie udanego zapisu,
3) otworzyć plik za pomocą edytora i sprawdź obsługę wyjątku,
4) dopisać kod odczytywania zawartości pliku.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

Ćwiczenie 2

Sprawdź w dostępnych źródłach informacji, czy środowisko systemowe może zapewnić obsługę wyjątków, gdy w programie nie zastosuje się narzędzi ich obsługi? Jeśli tak, to na jakim poziomie?

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) wyszukać w dostępnych źródłach potrzebne informacje. Zanotować również źródło informacji,
2) jeżeli system obsługuje wyjątki, zanotować warunki ich obsługiwania.

Wyposażenie stanowiska pracy:
– komputer z zainstalowaną przeglądarką internetową.

4.21.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) zaprogramować obsługę wyjątków?	☐	☐
2) spowodować powstanie wyjątku w kodzie programu?	☐	☐
3) określić jak działają wyjątki systemowe?	☐	☐

4.22. Projektowanie bibliotek funkcji

4.22.1. Materiał nauczania

Pojęcie bibliotek funkcji brzmi bardzo poważnie, ale oznacza zebranie niezbędnych lub przydatnych funkcji w jeden lub kilka plików o rozszerzeniach „cpp" i „h". Kiedy tworzy się rozbudowany program, bardzo często w różnych jego fragmentach korzysta się z tych samych funkcji. W takiej sytuacji wskazane jest zgromadzenie funkcji w jednym pliku o rozszerzeniu „cpp". Trzeba jednak utworzyć jeszcze tak zwany plik nagłówkowy posiadający rozszerzenie „h". Plik ten powinien zawierać nagłówki funkcji bez kodu. Tak skonstruowany plik nagłówkowy można dołączać do kodu za pomocą instrukcji `#include`.

4.22.2. Pytania sprawdzające

Odpowiadając na pytania sprawdzisz, czy jesteś przygotowany do wykonania ćwiczeń.
1. Jak definiuje się biblioteki funkcji?
2. Do czego służą biblioteki funkcji?

4.22.3. Ćwiczenia

Ćwiczenie 1

Napisz własną bibliotekę funkcji, która będzie zawierała deklaracje działań matematycznych takich jak: dodawanie, odejmowanie, mnożenie, dzielenie, potęgowanie i sumę ciągu arytmetycznego. Następnie napisz program, który z tych funkcji skorzysta.

Sposób wykonania ćwiczenia

Aby wykonać ćwiczenie powinieneś:
1) napisać odpowiednie funkcje i sprawdź ich działanie,
2) umieścić funkcje w bibliotece,
3) napisać program, który wykorzysta funkcje z biblioteki.

Wyposażenie stanowiska pracy:
– komputer z zainstalowanym kompilatorem języka C++.

4.22.4. Sprawdzian postępów

Czy potrafisz:	**Tak**	**Nie**
1) zaprojektować bibliotekę funkcji?	☐	☐
2) zaprogramować funkcje w bibliotece?	☐	☐
3) utworzyć plik nagłówkowy biblioteki?	☐	☐
4) wykorzystać bibliotekę w programie?	☐	☐

5. SPRAWDZIAN OSIĄGNIĘĆ

INSTRUKCJA DLA UCZNIA

1. Przeczytaj uważnie instrukcję.
2. Podpisz imieniem i nazwiskiem kartę odpowiedzi.
3. Zapoznaj się z zestawem pytań testowych.
4. Test zawiera 12 pytań testowych. Do każdego pytania dołączone są cztery możliwości odpowiedzi. Tylko jedna odpowiedź jest prawidłowa.
5. Udzielaj odpowiedzi wyłącznie na załączonej karcie odpowiedzi, stawiając w odpowiedniej rubryce znak × w przypadku pomyłki należy błędną odpowiedź zaznaczyć kółkiem, a następnie ponownie zakreślić odpowiedź prawidłową.
6. Pracuj samodzielnie, bo tylko wtedy będziesz miał satysfakcję z wykonanego zadania.
7. Jeżeli udzielenie odpowiedzi będzie Ci sprawiało trudność, wtedy najlepiej odłóż jego rozwiązanie na później. Wróć do niego, gdy zostanie Ci wolny czas.
8. Na rozwiązanie testu masz 40 minut.

ZESTAW PYTAŃ TESTOWYCH

1. Program napisany według podejścia obiektowego powinien spełniać następujące założenia:
   a) dziedziczenie, polikod, odwrócone interfejsy,
   b) abstrakcja, enkapsulacja, polimorfizm, dziedziczenie,
   c) przejrzystość, abstrakcja, grupowanie, struktura obiektowa,
   d) polimorfizmy, dziedziczenie, dziedziczenie odwrócone.

2. Jak powinna wyglądać deklaracja głównego bloku programu?
   a) int main()
```
{
  ...
  return 0;
}
```
   b) void main()
```
{
  ...
  return 0;
}
```
   c) int Main()
```
{
  ...
  return 0;
}
```
   d) void Main()
```
{
  ...
  return 0;
}
```
3. Prawidłowa definicja zmiennych w języku C++ ma postać:
   a) var wiek =12;
   b) var int: wiek = 12;
   c) string wiek = ‘12’;
   d) int wiek = 22;

4. Do podstawowych instrukcji w języku C++ należą konstrukcje:
   a) if, repeat…until, for, while, switch,
   b) switch, goto, foreach, for, while,
   c) if, for, do…while, while, switch,
   d) goto, ifnot, while, there,

5. Funkcje w języku C++ posiadają następujące własności:
   a) umożliwiają zgrupowanie kodu w jednym miejscu; mogą występować funkcje o tej samej nazwie i tej samej liczbie argumentów,
   b) umożliwiają zgrupowanie kodu w jednym miejscu; mogą występować funkcje o tej samej nazwie, ale różnej liczbie argumentów lub różnych typach argumentów,
   c) umożliwiają zgrupowanie kodu, ale ich ilość w kodzie programu jest ograniczona do 255,
   d) przyspieszają wykonanie programu, ale wymagają znacząco większej ilości pamięci.

6. Wskaż zdanie nieprawdziwe:
   a) zmienne definiowane w języku C++ mogą być typu: prostego, tablicowego, strukturalnego,
   b) zmienne zdefiniowane w języku C++ mogą być tablicą struktur,
   c) zmienne zdefiniowane w języku C++ mogą być strukturą tablic,
   d) nie wolno definiować zmiennych typu strukturalnego, które będą zawierały jako pod typ tablice i unie jednocześnie.

7. Wskaż zdanie NIEPRAWDZIWE: Wskaźniki w języku C++ mogą posłużyć do:
   a) przechowywania dowolnej ilości danych, nie określonej w momencie kompilacji programu,
   b) odwołania się do pierwszego elementu tablicy,
   c) obliczania wielkości pamięci zajętej przez daną zmienną,
   d) utworzenia struktury drzewa lub listy jedno i dwu kierunkowej.

8. Klasy w języku C++ dzielą składowe na następujące grupy:
   a) public, private, protected,
   b) public, private, final, protected,
   c) public, private, protect, local,
   d) publish, private, void.

9. Funkcje zaprzyjaźnione są
   a) pomostem pomiędzy funkcjami dwóch różnych klas,
   b) elementem pozostałym z języka C i nie zalecanym do stosowania w języku C++,
   c) elementem dodanym do języka C++, aby umożliwić tworzenie programów na różne platformy,
   d) elementem koniecznym każdej nowo zdefiniowanej klasy własnej.

10. Polimorfizm umożliwia:
   a) odwołanie się z klasy bazowej do atrybutów i metod klasy dziedziczącej,
   b) odwołanie się do metod klasy bazowej,
   c) wielokrotne dziedziczenie,
   d) kompilację kodu klasy na wiele platform.

11. Zaznacz błędne sformułowanie:
    a) strumienie są konstrukcją, która umożliwia pobieranie danych tylko z kamer cyfrowych – Video Streaming,
    b) strumienie służą do obsługi danych wejściowych i wyjściowych dla różnych źródeł,
    c) strumienie mogą być wykorzystane z plikami, klawiaturą i pamięcią,
    d) strumienie posiadają zdefiniowane operatory << i >>, które znacznie upraszają ich użycie.

12. Prawidłowy blok obsługi wyjątków wygląda następująco:
    a)
```
try {
  ...
  throw exception;
}
catch (type  exception)
{
  ...
}
```
    b)
```
do {
  ...
}
catch exception (e)
{
  ...
}
```
    c)
```
try {
  ...
  throw exception;
}
catch (type  exception)
{
  ...
}
finally {
  ...
}
```
    d)
```
try {
  try {
    ...
    throw exception;
    ...
  }
}
catch (type  exception)
{
  ...
}
```

KARTA ODPOWIEDZI

Imię i nazwisko .....

Programowanie w środowisku języka obiektowego

Zakreśl poprawną odpowiedź

Nr zadania	Odpowiedź				Punkty
1	a	b	c	d	
2	a	b	c	d	
3	a	b	c	d	
4	a	b	c	d	
5	a	b	c	d	
6	a	b	c	d	
7	a	b	c	d	
8	a	b	c	d	
9	a	b	c	d	
10	a	b	c	d	
11	a	b	c	d	
12	a	b	c	d	
				Razem:	

6. LITERATURA

1. Eckel B., Thinking in Java, Helion 2003
2. Liberty J., C++. Księga eksperta, Helion, 1999
3. Schmuller J., UML dla każdego, Helion 2003
4. Shtern V., C++. Inżynieria programowania, Helion, 2003
5. Stroustrup B., Język C++, WNT, 1995
6. Wirth N., Algorytmy + struktury danych = programy, WNT, 1998