DO NOWEJ PODSTAWY PROGRAMOWEJ

Część 2

Programowanie aplikacji

Kwalifikacja EE.09

Programowanie, tworzenie i administrowanie stronami internetowymi i bazami danych

Podręcznik do nauki zawodu
technik informatyk

Jolanta Pokorska

Helion

Redaktor prowadzący: Joanna Zaręba

Projekt okładki: Jan Paluch

Fotografia na okładce została wykorzystana za zgodą Shutterstock.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie?e092ti
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-4836-3

Printed in Poland.

Spis treści

Wstęp

Kwalifikacja EE.09. Programowanie, tworzenie i administrowanie stronami internetowymi i bazami danych. Część 2. Programowanie aplikacji. Podręcznik do nauki zawodu technik informatyk jest drugim z grupy podręczników przygotowanych dla kwalifikacji *EE.09. Programowanie, tworzenie i administrowanie stronami internetowymi i bazami danych*. Kwalifikacja ta wraz z kwalifikacją *EE.08. Montaż i eksploatacja systemów komputerowych, urządzeń peryferyjnych i sieci* obejmuje efekty kształcenia, których osiągnięcie jest niezbędne do uzyskania dyplomu potwierdzającego kwalifikacje zawodowe w zawodzie technik informatyk.

Treści zawarte w drugiej części podręcznika zostały oparte na podstawie programowej kształcenia w zawodach wprowadzonej rozporządzeniem Ministra Edukacji Narodowej z dnia 31 marca 2017 r.

Obejmują one zagadnienia teoretyczne prowadzące do uzyskania wymienionych w podstawie programowej efektów kształcenia, projekty różnych zadań oraz ich realizację praktyczną. Tak skonstruowany podręcznik pomaga uczniowi w zdobywaniu wymaganej wiedzy oraz umożliwia mu samodzielne poszerzanie umiejętności.

Podręcznik składa się z czterech rozdziałów. Ich budowa pozwala na realizację treści programowych w sposób wybrany przez nauczyciela.

Rozdział 1., „Podstawy programowania", zawiera omówienie podstawowych zagadnień związanych z programowaniem aplikacji desktopowych. Dotyczy efektów związanych z posługiwaniem się podstawowymi pojęciami z zakresu programowania oraz ze stosowaniem zasad algorytmicznego rozwiązywania problemów. Efektami tymi są: rozpoznawanie pojęć dotyczących języków programowania, kompilatorów, algorytmów i paradygmatów programowania obiektowego, stosowanie podstawowych zasad programowania i gotowych rozwiązań programistycznych oraz wykorzystywanie środowisk programistycznych.

Rozdział 2., „Programowanie w języku C++", zawiera omówienie zagadnień związanych z programowaniem w języku C++. Dotyczy efektów związanych z posługiwaniem się językiem C++ oraz z tworzeniem prostych aplikacji za pomocą tego języka. Efektami tymi są: wykorzystywanie środowisk programistycznych, stosowanie podstawowych typów danych i tworzenie własnych typów danych, stosowanie instrukcji, funkcji i procedur w języku C++, tworzenie własnych funkcji i procedur, kompilowanie i uruchamianie kodów źródłowych.

Rozdział 3., „Obiektowy C++", zawiera omówienie zagadnień związanych z wykorzystywaniem paradygmatu programowania obiektowego w języku C++ do tworzenia aplikacji. Dotyczy efektów związanych z posługiwaniem się obiektowym językiem C++ podczas tworzenia prostych aplikacji. Efektami tymi są: stosowanie podstawowych elementów

programowania obiektowego (klas, obiektów, metod i właściwości), tworzenie własnych klas, obiektów, metod i właściwości, stosowanie specyfikatorów dostępu, budowanie hierarchii klas, wykorzystywanie mechanizmu dziedziczenia i polimorfizmu.

Rozdział 4., „Programowanie w języku C#", zawiera omówienie zagadnień związanych z programowaniem obiektowym w języku C#. Dotyczy efektów związanych z programowaniem w języku C# oraz z tworzeniem prostych aplikacji za pomocą tego języka. Efektami tymi są: stosowanie podstawowych typów danych i tworzenie własnych typów danych, stosowanie instrukcji języka C#, stosowanie i tworzenie własnych klas, obiektów, metod i właściwości, stosowanie specyfikatorów dostępu, budowanie hierarchii klas, wykorzystywanie mechanizmu dziedziczenia, tworzenie aplikacji z wykorzystaniem graficznego interfejsu, kompilowanie i uruchamianie kodów źródłowych.

2.10. Funkcje

Funkcje to wydzielone części programu wykonujące jakieś operacje. Są szczególnie użyteczne, gdy trzeba wielokrotnie wykonywać te same czynności. Mogą również pomóc w uporządkowaniu rozbudowanego programu poprzez jego podział na mniejsze bloki.

Tworząc funkcję, należy podać jej typ, nazwę oraz w nawiasach okrągłych listę argumentów przekazywanych do funkcji, oddzielonych przecinkiem. Następnie w nawiasach klamrowych należy zapisać instrukcje wykonywane przez funkcję, np.:

```
int licz(int a, int b)
{
   // instrukcje
}
```

Typ deklarowanej funkcji określa, jakiego typu wartość zwróci funkcja. Nazwa funkcji powinna mówić, jakie jest jej przeznaczenie. Lista argumentów odzwierciedla dane, na których będzie ona działała. Funkcja może posiadać jeden argument, wiele argumentów lub nie posiadać ich wcale. W nawiasach klamrowych powinny zostać zapisane wszystkie instrukcje, które są niezbędne do prawidłowego jej działania. Ostatnią instrukcją powinno być polecenie `return`, które określa, jaka wartość zostanie zwrócona przez funkcję. Instrukcja `return` powoduje dodatkowo natychmiastowe przerwanie działania funkcji i powrót do miejsca jej wywołania.

Główną i najważniejszą funkcją w języku C++ jest poznana już funkcja `main()`. Funkcja ta ma zdefiniowany typ, nie posiada argumentów, dlatego nawiasy okrągłe są puste, zwraca wartość przez polecenie `return 0`, a w nawiasach klamrowych jest zapisany ciąg poleceń, które są niezbędne do prawidłowego działania programu.

Przykład 2.71

```
int suma(int a, int b)
{
   int c;
   c = a + b;
   return c;
}
```

W podanym przykładzie została zdefiniowana funkcja sumowania dwóch liczb. Wartości do sumowania zostaną przekazane do funkcji przez zmienne `a` i `b`. Wewnątrz funkcji wartości zmiennych zostaną zsumowane i zapisane w zmiennej `c`. Po zakończeniu działania funkcja zwróci wartość zapisaną w zmiennej `c`. Zwrócona wartość zmiennej będzie typu `int`.

Podczas deklarowania funkcji należy zawsze podawać typ wartości zwracanej przez funkcję. Jeżeli funkcja nie zwraca żadnej wartości, powinien to być typ `void`. W ten

sposób przekazujemy do kompilatora informację, że funkcja nie zwraca żadnej wartości. W takim przypadku funkcja nie musi zawierać instrukcji `return`.

Przykład 2.72

```
void tekst(string imie)
{
   cout << imie << " witaj w domu."  << endl;
}
string podaj_imie()
{
   string imie;
   cout "Wpisz swoje imie: ";
   cin >> imie;
   return imie;
}
```

2.10.1. Sposoby definiowania funkcji

W języku C++ funkcje można tworzyć na kilka sposobów:

- tworząc definicję funkcji i deklarując jej zawartość powyżej funkcji `main()`;
- tworząc prototyp funkcji powyżej funkcji `main()`i deklarując jej zawartość poniżej funkcji `main()`;
- tworząc funkcje we własnej bibliotece.

2.10.1.1. Deklarowanie i definiowanie funkcji przed funkcją main()

Pierwszy sposób tworzenia funkcji polega na jej zadeklarowaniu i zdefiniowaniu jej zawartości przed funkcją `main()`.

Przykład 2.73

```
#include <iostream>
#include <string>
using namespace std;
// definicja funkcji suma() i jej zawartości
int suma(int a, int b)
{
   int c;
```

```
    c = a + b;
    return c;
}
// funkcja main()
int main()
{
    // instrukcje funkcji main()
    return 0;
}
```

2.10.1.2. Prototyp funkcji

Drugi sposób polega na zadeklarowaniu funkcji przed funkcją `main()` i zdefiniowaniu jej zawartości po zakończeniu funkcji `main()`. W tym przypadku deklaracja funkcji musi zostać zakończona średnikiem.

Wiersz zawierający deklarację funkcji nazywany jest prototypem i informuje kompilator o funkcjach, które będą używane w programie.

Przykład 2.74

```
#include <iostream>
#include <string>
using namespace std;
// deklaracja funkcji iloczyn() bez definiowania jej zawartości
int iloczyn(int a, int b);     // średnik na końcu prototypu funkcji
// funkcja main()
int main()
{
    // instrukcje funkcji main()
    return 0;
}
// definiowanie zawartości funkcji iloczyn() po funkcji main()
int iloczyn(int a, int b)
{
    int c;
    c = a * b;
    return c;
}
```

2.10.1.3. Biblioteki funkcji

Trzeci sposób polega na umieszczeniu wszystkich zdefiniowanych funkcji w oddzielnym pliku lub plikach nazwanych bibliotekami.

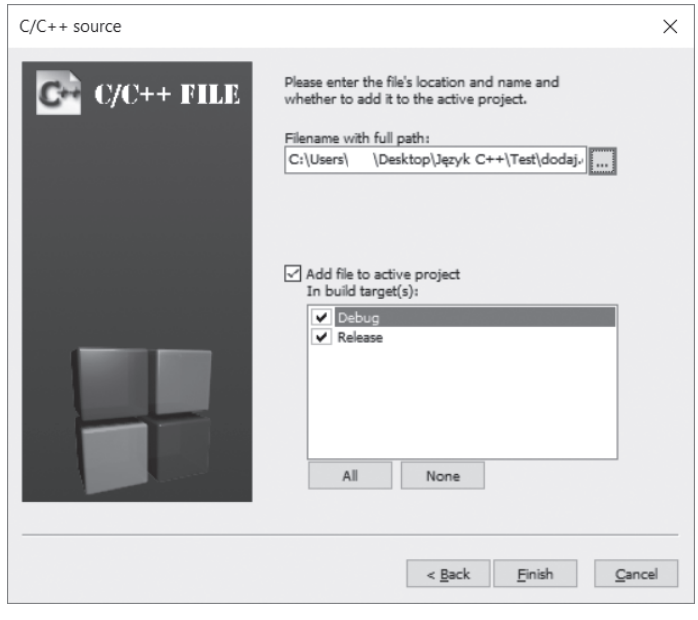
Aby wyodrębnić funkcje w osobnym pliku, należy przygotować dwa pliki:

- Plik źródłowy z rozszerzeniem **.cpp*. W tym pliku powinien zostać umieszczony kod źródłowy funkcji.
- Plik nagłówkowy z rozszerzeniem **.h*. W tym pliku powinien zostać umieszczony opis funkcji, czyli jej prototyp.

Przykład 2.75

Tworzenie pliku źródłowego

Aby utworzyć plik źródłowy w Code::Blocks, należy wybrać w menu *File/New/File…* i w otwartym oknie zaznaczyć *C/C++ source*, a następnie kliknąć przycisk *Go*. W otwartym oknie kreatora należy wybrać język *C++* i w kolejnym oknie wpisać nazwę tworzonego pliku. Plik należy zapisać w tym samym folderze, w którym znajduje się plik *main.cpp*, należy zaznaczyć wszystkie opcje dostępne w oknie i zakończyć naciśnięciem przycisku *Finish* (rysunek 2.19).



Rysunek 2.19. Tworzenie pliku źródłowego

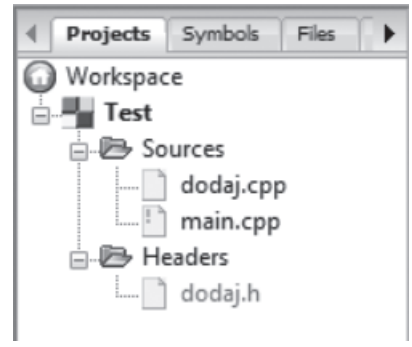
Tworzenie pliku nagłówkowego

Plik nagłówkowy tworzy się podobnie jak plik źródłowy. Należy wybrać w menu *File/New/File…* i w otwartym oknie zaznaczyć *C/C++ header*, a następnie kliknąć przycisk

Go. W otwartym oknie kreatora należy wybrać język *C++* i w kolejnym oknie wpisać nazwę tworzonego pliku. Nazwa powinna być taka sama jak nazwa pliku źródłowego, ale z rozszerzeniem *.h*. Plik należy zapisać w tym samym folderze i zaznaczyć wszystkie opcje dostępne w oknie, a potem zakończyć naciśnięciem przycisku *Finish*.

Utworzone pliki można zobaczyć w oknie projektu (rysunek 2.20).

Po zakończeniu tych czynności można przystąpić do wpisywania kodów do utworzonych plików.



Rysunek 2.20.
Pliki utworzone w projekcie

Plik źródłowy

```
#include "dodaj.h"    // informuje o dołączeniu pliku ze zdefiniowaną biblioteką
int suma(int a, int b)    // treść funkcji
{
   int c;
   c = a + b;
   return c;
}
```

Plik nagłówkowy

Plik ten zawiera gotowy fragment kodu, który powinien pozostać niezmieniony, a swój kod należy umieścić za drugą linią istniejącego kodu. Powinien on zawierać prototyp funkcji.

```
#ifndef DODAJ_H_INCLUDED
#define DODAJ_H_INCLUDED
int suma(int a, int b);
#endif // DODAJ_H_INCLUDED
```

Dołączenie plików do programu

Ostatnim etapem pracy powinno być dołączenie powstałych plików do pliku *main.cpp*.

```
#include <iostream>
#include "dodaj.h"    // dołączenie biblioteki
using namespace std;
int main()
{
   int liczba1 = 5, liczba2 = 7, wynik;
```

```
    cout << "Pierwsza liczba: " << liczba1 << endl;
    cout << "Druga liczba: " << liczba2 << endl;
    wynik = suma(liczba1, liczba2);
    cout << "Wynik dodawania liczb: " << wynik << endl;
    return 0;
}
```

W jednym pliku biblioteki można umieścić dowolną liczbę funkcji. Można je pogrupować według przeznaczenia i utworzyć kilka bibliotek.

2.10.1.4. Przeciążenie funkcji

Można tworzyć wiele funkcji o tej samej nazwie; wtedy każda z nich powinna mieć inną listę argumentów, różne typy argumentów lub różną ich liczbę lub zwracać wartość innego typu. Definiowanie wielu funkcji o tej samej nazwie określamy mianem **przeciążenia funkcji**.

Przykład 2.76

```
int dodaj(int a, int b);
float dodaj(float a, float b);
int dodaj(int a, int b, int c);
```

Przeciążenie funkcji jest również nazywane polimorfizmem funkcji. W programowaniu obiektowym oznacza to, że funkcja ma wiele postaci.

W momencie wywołania argumenty będą decydowały o tym, która funkcja zostanie wykonana.

2.10.2. Wywołanie funkcji

Wywołanie funkcji nastąpi po podaniu w kodzie programu jej nazwy wraz z parametrami umieszczonymi w nawiasach okrągłych. Po wywołaniu funkcji zostaną wykonane instrukcje w niej zawarte, a po ich zakończeniu nastąpi powrót do miejsca programu, w którym funkcja została wywołana.

Funkcja może zostać wywołana z dowolnego miejsca programu, może zostać również wywołana z wnętrza innej funkcji.

Przykład 2.77

```
#include <iostream>
#include <string>
using namespace std;
int suma(int a, int b)
```

```
{
   int c = a + b;
   return c;
}
int iloczyn(int a, int b);
int main()
{
   int licz1 = 2, licz2 = 3, licz3;
   licz3 = suma(licz1, licz2);
   cout << "Wynik sumowania liczb " << licz1
<< " i " << licz2 << " wynosi " << licz3 << endl;
   licz3 = suma(5, 9);
   cout << "Wynik sumowania liczb 5 i 9 wynosi " << licz3 << endl;
   licz3 = iloczyn(licz1, licz2);
   cout << "Wynik mnozenia liczb " << licz1
<< " i " << licz2 << " wynosi " << licz3 << endl;
   licz3 = iloczyn(5, 9);
   cout << "Wynik mnozenia liczb 5 i 9 wynosi " << licz3 << endl;
   return 0;
}
int iloczyn(int a, int b)
{
   int c;
   c = a * b;
   return c;
}
```

W podanym przykładzie podstawowa czynność sumowania dwóch liczb została zapisana w oddzielnej funkcji `suma()`. Funkcja została zdefiniowana w części nagłówkowej programu, przed funkcją `main()`. Wartości do sumowania zostaną przekazane do funkcji przez argumenty `a` i `b` podczas jej wywołania. Deklaracja argumentów funkcji została poprzedzona deklaracją ich typów, tak jak w deklaracji zwykłych zmiennych. Funkcja została wywołana w programie dwa razy, a zwrócony przez nią wynik został przypisany do zmiennej `licz3`. Pierwszy raz funkcja została wywołana z wartościami przekazanymi przez zmienne `licz1` i `licz2`. Drugi raz została wywołana z wartościami liczbowymi `5` i `9`.

Natomiast iloczyn dwóch liczb został zapisany w funkcji `iloczyn()`. Funkcja została zadeklarowana w części nagłówkowej programu, a jej definicja znalazła się za funkcją

`main()`. Pierwszy raz funkcja `iloczyn()` została wywołana z wartościami przekazanymi przez zmienne `licz1` i `licz2`. Drugi raz została wywołana z wartościami liczbowymi `5` i `9`.

Ćwiczenie 2.18

Napisz program, który po podaniu przez użytkownika liczby wyświetli jej wartość podniesioną do kwadratu. Obliczanie kwadratu liczby zrealizuj za pomocą funkcji. Zakończenie wprowadzania liczb powinno nastąpić po wpisaniu liczby `0`.

Rozwiązanie

```
#include <iostream>
using namespace std;
double kwadrat(double x);
int main()
{
   double liczba, wynik;
   do
   {
      cout << "Podaj liczbe: ";
      cin >> liczba;
      wynik = kwadrat(liczba);
      cout << "Kwadrat liczby " << liczba << " to " << wynik << endl;
   }
   while (liczba != 0);
   return 0;
}
double kwadrat(double x)
{
   double do_kwadratu;
   do_kwadratu = x * x;
   return do_kwadratu;
}
```

Zadanie 2.12

Napisz program, który będzie zawierał funkcję rysującą prostokąt za pomocą znaków gwiazdki (`*`). Wartości określające wysokość i szerokość prostokąta będą przekazywane do funkcji za pomocą argumentów po podaniu tych wartości przez użytkownika.

Rozwiązanie

```
#include <iostream>
using namespace std;
void prostokat(int k, int w);
int main()
{
   int szer, wys;
   cout << "Podaj szerokosc prostokata: ";
   cin >> szer;
   cout << "Podaj wysokosc prostokata: ";
   cin >> wys;
   prostokat (szer, wys);
   return 0;
}
void prostokat(int k, int w)
{
   for (int wiersz = 0; wiersz < w; wiersz++)
   {
      for (int kol = 0); kol < k; kol++)
      {
         cout << "*";
      }
      cout << endl;
   }
}
```

2.10.3. Zasięg zmiennych funkcji

Zasięg zmiennej to obszar, w którym można się odwoływać bezpośrednio do zmiennej. Jak wiadomo, zmienna może mieć zasięg lokalny lub globalny.

Zasięg lokalny zmiennej dotyczy pojedynczego bloku instrukcji, a w przypadku funkcji dotyczy tylko funkcji, w której zmienna została zdefiniowana i poza którą zmienna nie jest widoczna. Po zakończeniu wykonywania funkcji taka zmienna przestaje istnieć. Jeżeli funkcja jest wykonywana wielokrotnie, za każdym razem zmienna lokalna jest tworzona od nowa.

Zmienna posiadająca zasięg globalny jest dostępna w całym programie. Takie zmienne powinny być deklarowane przed deklaracją wszystkich funkcji. Wartość przypisana do

zmiennej globalnej obowiązuje we wszystkich funkcjach, w których występuje odwołanie do tej zmiennej. Zmienne globalne są dostępne z dowolnego miejsca programu, a zmiana wartości przypisanej do zmiennej obowiązuje we wszystkich miejscach jej użycia w programie.

2.10.3.1. Zmienne statyczne

Zmienne statyczne są zmiennymi lokalnymi funkcji, które zachowują swoją wartość między kolejnymi wywołaniami tej funkcji. Domyślnie zmienna lokalna jest tworzona w chwili wywołania funkcji i znika, gdy funkcja zakończy działanie. Gdy zmienna lokalna zostanie zadeklarowana jako statyczna, będzie istniała tak długo, jak długo będzie wykonywany program. Mimo to będzie dostępna tylko wewnątrz funkcji, w której została zadeklarowana.

Deklaracja zmiennej statycznej ma postać:

```
static int liczba;
```

Przykład 2.78

```
#include <iostream>
#include <string>
using namespace std;
void pokaz();
int main()
{
   pokaz();
   pokaz();
   pokaz();
   pokaz();
   return 0;
}
void pokaz()
{
   int i = 1;
   cout << "Funkcja wywolana " << i << " raz(y)" << endl;
   i++;
}
```

W podanym przykładzie wartość zmiennej `i` będzie ustawiana na `1` przy każdym wywołaniu funkcji `pokaz()`, co spowoduje wyświetlenie tego samego komunikatu, mówiącego, że funkcja została wywołana jeden raz.

Jeżeli przy tworzeniu zmiennej zostanie zadeklarowane, że jest to zmienna statyczna, jej wartość zostanie zachowana między kolejnymi wywołaniami funkcji.

Przykład 2.79

```
#include <iostream>
#include <string>
using namespace std;
void pokaz();
int main()
{
   pokaz();
   pokaz();
   pokaz();
   pokaz();
   return 0;
}
void pokaz()
{
   static int i = 1;
   cout << "Funkcja wywolana " << i << " raz(y)" << endl;
   i++;
}
```

W podanym przykładzie zmienna `i` została zadeklarowana jako zmienna statyczna (`static int i = 1;`), dlatego przy opuszczaniu funkcji jej wartość zostanie zapamiętana. Przy ponownym wywołaniu funkcji instrukcja przypisująca początkową wartość zmiennej będzie ignorowana, a stosowana będzie wartość zapamiętana przy poprzednim wywołaniu funkcji. Wartość zmiennej `i` będzie zwiększana o 1.

2.10.4. Sposoby przekazywania argumentów do funkcji

Argumenty mogą być przekazywane do funkcji na dwa sposoby:

- przez wartość,
- za pomocą referencji.

2.10.4.1. Przekazywanie argumentów przez wartość

Domyślnym sposobem przekazywania argumentów do funkcji jest przekazywanie przez wartość. Ten sposób przekazywania był stosowany w przykładach prezentowanych do tej pory. W praktyce oznaczało to, że do funkcji przekazywane są kopie argumentów źródłowych i wszystkie operacje wykonywane są na kopiach. Zostało to zilustrowane w poniższym przykładzie.

Przykład 2.80

```
#include <iostream>
using namespace std;
// funkcja dwa() podwaja wartość podanej liczby
int oblicz(int a)
{
   int wynik;
   a = a * 2;
   return a;
}
int main()
{
   int dwa_razy, liczba = 7;
   cout << "Wartosc liczby wynosi: " << liczba << endl;
   dwa_razy = oblicz(liczba);
   cout << "Podwojenie liczby wynosi: " << dwa_razy << endl;
   cout << "Wartosc liczby wynosi: " << liczba << endl;
   dwa_razy = oblicz(liczba);
   cout << "Podwojenie liczby wynosi: " << dwa_razy << endl;
   return 0;
}
```

Działanie podanego w przykładzie kodu jest następujące: program przydziela zmiennej `liczba` obszar pamięci i zapisuje w nim wartość `7`. Odwołanie do tego miejsca pamięci będzie się odbywało przez nazwę zmiennej `liczba`. Następnie rozpoczyna się wykonywanie funkcji `oblicz()`, która kopiuje wartość przekazanej zmiennej `liczba` i skopiowaną wartość zapisuje w pamięci przydzielonej zmiennej `a`. Na tej zmiennej wykonuje obliczenia i wynik zapisuje w pamięci przydzielonej zmiennej `a`, czyli wartość zmiennej `a` ulega zmianie. Wartość zmiennej `a` jest przekazywana do zmiennej `dwa_razy`. Wartość zmiennej `liczba` nie uległa zmianie. Ponowne wywołanie funkcji znowu ustawi wartość zmiennej `a` na wartość przekazaną przez zmienną `liczba`, czyli wartość `7`.

2.10.4.2. Przekazywanie argumentów przez referencję

Jeżeli zastosujemy przekazywanie argumentów za pomocą referencji, to funkcja zamiast tworzyć kopie, będzie operować na oryginalnych zmiennych. Przy wyborze takiej metody przed przekazywanymi argumentami należy umieścić znak & (ampersand).

Przykład 2.81

```
#include <iostream>
using namespace std;
// funkcja oblicz() podwaja wartość podanej liczby
int oblicz(int& a)
{
   int wynik;
   a = a * 2;
   return a;
}
int main()
{
   int dwa_razy, liczba = 7;
   cout << "Wartosc liczby wynosi: " << liczba << endl;
   dwa_razy = oblicz(liczba);
   cout << "Podwojenie liczby wynosi: " << dwa_razy << endl;
   cout << "Wartosc liczby wynosi: " << liczba << endl;
   dwa_razy = oblicz(liczba);
   cout << "Podwojenie liczby wynosi: " << dwa_razy << endl;
   return 0;
}
```

W podanym przykładzie została zdefiniowana funkcja `oblicz()`, która podwaja wartość przekazaną do zmiennej. Zmienna do funkcji jest przekazywana przez referencję, czyli wszystkie działania są przeprowadzane na oryginalnej zmiennej. Początkowa wartość zmiennej `liczba` wynosi `7`. Po wywołaniu funkcji `oblicz()` jej wartość zmieni się na `14`, a po ponownym wywołaniu funkcji ponownie zostanie podwojona.

Wartość zmiennej `liczba` ulega zmianie po wywołaniu funkcji `oblicz()`, ponieważ funkcja działa na oryginalnych danych.

2.10.4.3. Domyślne argumenty funkcji

W języku C++ można tworzyć funkcje, których argumenty będą miały określone wartości domyślne. Mechanizm ten pozwala na wywołanie funkcji bez podawania jej

argumentów. Jako argumenty zostaną wstawione wartości zdefiniowane podczas tworzenia funkcji. Jeżeli natomiast argumenty będą podane, funkcja zostanie wywołana z tymi argumentami.

Przykład 2.82

```
#include <iostream>
#include <string>
using namespace std;
void kolor(string kol = "niebieski");
int main()
{
   string moj_kolor;
   moj_kolor = "czerwony";
   kolor(moj_kolor);
   moj_kolor = "zielony";
   kolor(moj_kolor);
   kolor();
   return 0;
}
void kolor(string kol)
{
   cout << "Moj ulubiony kolor to: " << kol << endl;
}
```

Utworzona w przykładzie funkcja `kolor()` posiada zdefiniowany argument domyślny. Przy pierwszym i drugim wywołaniu funkcja zostanie wykonana z podaną wartością argumentu, przy trzecim wartość argumentu zostanie pominięta i funkcja zostanie wykonana z domyślną wartością tego argumentu.

Wartości domyślne argumentów powinny być określane wyłącznie w prototypie funkcji. Można je przypisać dowolnej liczbie argumentów. Ale jeżeli wartość domyślna zostanie przypisana określonemu argumentowi, to musi on wystąpić jako ostatni na liście lub wszystkie następne argumenty również muszą posiadać wartości domyślne.

Zadanie 2.13

Napisz program obliczający średnią ocen z matematyki. Oceny wprowadzane przez użytkownika będą zapisywane w 10-elementowej tablicy. Kiedy tablica zostanie wypełniona, powinien się pojawić komunikat informujący o tym, że nie można wprowadzać kolejnych ocen. Tablica może zostać wypełniona tylko częściowo i po wpisaniu przez użytkownika liczby 0 jej wypełnianie się zakończy. Obliczanie średniej ocen zrealizuj

za pomocą funkcji. Funkcja powinna liczyć średnią tylko z ocen wprowadzonych przez użytkownika.

Rozwiązanie

```
#include <iostream>
using namespace std;
float srednia(float s, int l);
int main()
{
   float ocena, t_oceny[10], suma;
   int i;
   cout << "Obliczanie sredniej ocen z matematyki.
Mozna wprowadzic do 10 ocen." <<endl;
   for (i = 0; i < 10; )
   {
      cout << "Podaj ocene: ";
      cin >> ocena;
      if (ocena == 0)
      {
         break;
      }
      else
      {
         if (ocena > 6 || ocena < 1)
         {
            cout << "Bledna ocena !!! - " << ocena << endl;
            continue;
         }
         else
         {
            t_oceny[i] = ocena;
            suma += t_oceny[i];
            i++;
         }
      }
```

```
    }
    cout << "Srednia ocen wynosi: " << srednia(suma, i) << endl;
    return 0;
}
float srednia(float s, int l)
{
    return s / l;
}
```

2.10.5. Funkcje wbudowane

Język C++ udostępnia wiele gotowych funkcji, dzięki którym można ułatwić i uprościć pisanie programów. Aby korzystać z tych funkcji, należy dołączyć do programu zawierające je biblioteki.

2.10.5.1. Funkcje matematyczne

Większość funkcji matematycznych jest zawarta w pliku `cmath`. Aby z nich korzystać, należy dołączyć do programu bibliotekę `cmath` przez wpisanie w nagłówku programu kodu:

```
#include <cmath>
```

2.10.5.1.1. Funkcja pow()

`pow()` to funkcja potęgowania. Język C++ nie posiada operatora potęgowania, więc żeby wykonać potęgowanie liczby, należy użyć tej funkcji w postaci:

```
double pow(double base, double exponent);
```

gdzie:

- argument `base` to podstawa potęgi;
- argument `exponent` to wykładnik potęgowania.

Przykład 2.83

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    int liczba;
    double wynik;
```

```
    cout << "Podaj liczbe calkowita: ";
    cin >> liczba;
    wynik = pow(liczba, 2);
    cout << "Kwadrat liczby " << liczba << " wynosi " << wynik << endl;
    return 0;
}
```

2.10.5.1.2. Funkcja sqrt()

Funkcja `sqrt()` służy do obliczenia pierwiastka kwadratowego wybranej liczby. Zapisywana jest w postaci:

```
double sqrt(double x);
```

gdzie argument `x` to liczba, z której wyliczany jest pierwiastek.

2.10.5.1.3. Funkcja exp()

Funkcja `exp()` to funkcja wykładnicza obliczająca wartość e^x. Zapisywana jest w postaci:

```
double exp(double x);
```

2.10.5.1.4. Funkcja fabs()

Funkcja `fabs()` oblicza wartość bezwzględną z podanej zmiennej. Zapisywana jest w postaci:

```
int fabs(int a);
```

2.11. Operacje na plikach

Programy tworzone w języku C++ mogą się odwoływać do danych zapisanych w plikach, mogą również przechowywać w plikach dane uzyskane podczas wykonywania programu.

2.11.1. Zapisywanie do pliku

Aby wyświetlić na ekranie określoną informację lub pobrać informację z klawiatury, należało zdefiniować strumień wejścia-wyjścia. Podobnie, aby skomunikować się z plikiem i zapisać informację do pliku lub odczytać ją z niego, należy zdefiniować **strumień plikowy**. Najpierw należy dołączyć bibliotekę, która udostępni polecenia do komunikowania się z plikiem. Jest to biblioteka `fstream`, umieszczana w nagłówku programu w postaci:

```
#include <fstream>
```

Do komunikowania się z plikiem dla każdego pliku należy utworzyć osobny strumień.

2.11.1.1. Otwieranie pliku do zapisu

Strumienie plikowe definiuje się podobnie jak zmienne. Należy podać nazwę, określić typ i podać wartość inicjującą strumień. Do zapisu do pliku stosuje się strumień typu `ofstream`. Wartością inicjującą strumienia jest ścieżka do pliku, który ma zostać otwarty.

Przykład 2.84

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   ofstream doPliku("C:/Dane/tekst.txt");     // deklaracja strumienia
   // ciąg poleceń
   return 0;
}
```

W podanym przykładzie został utworzony strumień do zapisania danych do pliku *tekst.txt*. Jeżeli plik o podanej nazwie nie istnieje, to zostanie utworzony.

Ścieżka do pliku musi zostać zapisana w cudzysłowie. Może być ona zapisana na dwa sposoby, jako:

- ścieżka bezwzględna — jest to pełna ścieżka dostępu do pliku;
- ścieżka względna — określa położenie pliku względem miejsca, w którym został uruchomiony program.

Aby użyć zmiennej typu string, która przechowuje ścieżkę do pliku, do utworzenia strumienia, konieczne jest użycie dostępnej w dołączonej bibliotece funkcji `c_str()`, która dokona konwersji typu na typ oczekiwany dla wartości inicjującej strumień.

```
string const nazwaPlik("C:/Dane/tekst.txt");     // stała zawierająca ścieżkę do pliku
ofstream doPliku(nazwaPlik.c_str());     // definicja strumienia do zapisu do pliku
```

Ścieżka do pliku została zapisana w zadeklarowanej stałej typu `string`. Następnie został utworzony strumień typu `ofstream` do pliku o podanej ścieżce (zapisanej w stałej `nazwaPlik`).

Podczas otwierania pliku należy sprawdzić, czy operacja jego otwarcia się powiodła. Można do tego użyć instrukcji `if` w postaci:

```
ofstream doPliku("C:/Dane/tekst.txt");
if (doPliku)    // czy plik otwarty
{
```

```
   // OK, wykonanie poleceń do pliku
}
else
{
   cout << "Blad-plik nie zostal otwarty." << endl;
}
```

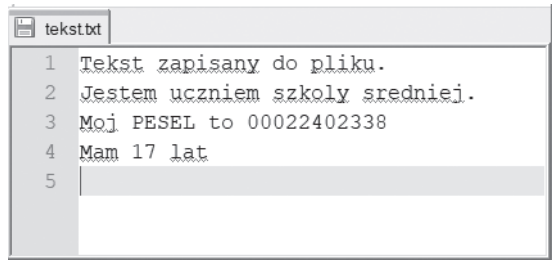
2.11.1.2. Zapisywanie danych do pliku

Przy zapisywaniu danych do pliku, podobnie jak przy wyświetlaniu danych, używany jest operator <<.

Przykład 2.85

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
   string const nazwaPlik("C:/Dane/tekst.txt");
   ofstream doPliku(nazwaPlik.c_str());
   if (doPliku)
   {
      int wiek = 17;
      string pe = "00022402338";
      doPliku << "Tekst zapisany do pliku." << endl;
      doPliku << "Jestem uczniem szkoly sredniej." << endl;
      doPliku << "Moj PESEL to " << pe << endl;
      doPliku << "Mam " << wiek << " lat" << endl;
   }
   else
   {
      cout << "Blad-plik nie zostal otwarty." << endl;
   }
   return 0;
}
```

W wyniku wykonania kodu podanego w przykładzie na dysku *C:*, w folderze *Dane*, został utworzony plik *tekst.txt*. Jego zawartość została pokazana na rysunku 2.21.



Rysunek 2.21. Wynik zapisu do pliku

Przy ponownym uruchomieniu programu, gdy plik już istnieje, jego zawartość zostanie usunięta i plik zostanie zapisany ponownie.

Istnieje możliwość dopisywania nowych danych do istniejącego pliku bez usuwania już istniejących. W tym celu podczas otwierania pliku należy zadeklarować ten sposób zapisywania do pliku. Służy do tego specjalny parametr `ios::app`, dodawany do instrukcji tworzącej strumień w postaci:

```
ofstream doPliku("C:/Dane/tekst.txt", ios::app);
```

Zadanie 2.14

Napisz program, który będzie pobierał od użytkownika jego dane i zapisywał je do pliku.

2.11.2. Odczytywanie z pliku

2.11.2.1. Otwieranie pliku do odczytu

Do odczytu z pliku stosuje się taką samą zasadę jak do zapisu do pliku, tylko zamiast strumienia `ofstream` używa się strumienia `ifstream`. Wartością strumienia jest, podobnie jak poprzednio, ścieżka do pliku, który ma zostać otwarty.

```
ifstream zPliku("C:/Dane/tekst.txt");
if (zPliku)    // czy plik otwarty
{
   // OK, wykonanie poleceń do pliku
}
else
{
   cout << "Blad-plik nie zostal otwarty." <<endl;
}
```

Zawartość pliku może zostać odczytana na jeden z trzech sposobów:

- przy użyciu funkcji `getline()` — odczyt odbywa się linia po linii;
- przy użyciu operatora `>>` — odczyt odbywa się słowo po słowie;
- przy użyciu funkcji `get()` — odczyt odbywa się znak po znaku.

2.11.2.1.1. Odczyt linia po linii

W tym trybie odczytywana jest jedna linia tekstu, która zapisywana jest jako łańcuch znaków.

Przykład 2.86

```
string liniaTekstu;
getline(zPliku, liniaTekstu);    // odczyt jednej linii tekstu
```

2.11.2.1.2. Odczyt słowo po słowie

Odczytywane jest jedno słowo z pliku, które następnie zapisywane jest do zmiennej. Odczyt rozpoczyna się od bieżącego miejsca w pliku, a kończy się na najbliższej spacji. Odczytany tekst może być traktowany jako typ `string`, `int` lub `double`, w zależności od typu zmiennej, do której zostanie zapisany.

Przykład 2.87

```
string slowo;
zPliku >> slowo;    // odczyt jednego słowa
double liczba;
zPliku >> liczba;    // odczyt liczby
```

2.11.2.1.3. Odczyt znak po znaku

Odczyt z pliku odbywa się po jednym znaku, a wynik jest zapisywany w zmiennej typu `char`. Tą metodą można odczytać wszystkie znaki zapisane w pliku, a więc również spację, znak nowego wiersza, znak tabulacji.

Przykład 2.88

```
char znak;
zPliku.get(znak);    // odczyt jednego znaku
```

2.11.2.2. Odczytywanie całego pliku

Przy odczytywaniu zawartości całego pliku istotne jest określenie tego, jak należy zakończyć odczyt, gdy pojawi się jego koniec. W celu określenia, czy dalsze odczytywanie jest możliwe, należy użyć funkcji `getline()`. Funkcja ta służy do odczytu linii tekstu, ale dodatkowo zwraca wartość logiczną określającą, czy został osiągnięty koniec pliku. Funkcja zwraca wartość `true`, jeśli można kontynuować odczytywanie z pliku, lub wartość `false`, jeśli został osiągnięty koniec pliku.

Przykład 2.89

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
   string linia;
   string const nazwaPlik("C:/Dane/tekst.txt");
   ifstream zPliku(nazwaPlik.c_str());
   if (zPliku)
   {
      while(getline(zPliku, linia))    // dopóki nie koniec pliku
         cout << linia << endl;    // wyświetl kolejną linię
   }
   else
      cout << "Blad-plik nie zostal otwarty." <<endl;
   return 0;
}
```

Otwarty plik należy zamknąć, używając funkcji `close()`. Jeżeli plik nie zostanie zamknięty za pomocą tej funkcji, zostanie zamknięty automatycznie, gdy program zakończy wykonywanie bloku, w którym zostały zdefiniowane strumienie plikowe.

Zadanie 2.15

Napisz program, który będzie pobierał od użytkownika informacje na temat przeczytanych przez niego książek (takie jak tytuł, autor, tematyka) i zapisywał je do pliku w trybie dopisywania, a następnie na życzenie użytkownika będzie wyświetlał zawartość pliku.

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

V

W

Z

Kwalifikacja EE.09

Programowanie, tworzenie i administrowanie stronami internetowymi i bazami danych

Podręcznik do nauki zawodu **technik informatyk**

Technik informatyk to nie tylko tytuł uzyskany po ukończeniu szkoły średniej, ale i zawód będący przepustką do kariery. Dzięki solidnej podstawie teoretycznej i na bieżąco wprowadzanym uaktualnieniom podręcznika uczniowie — już jako absolwenci — łatwo nawiążą współpracę z firmami informatycznymi, także tymi przygotowującymi aplikacje desktopowe dla największych przedsiębiorstw w kraju i za granicą. Autorka książki opracowała pozycję na wysokim poziomie merytorycznym, okraszoną licznymi przykładami i zadaniami umożliwiającymi praktyczne zastosowanie podanych informacji.

Budowa podręcznika i poszczególnych rozdziałów pozwala zarówno na realizację treści programowych w sposób wybrany przez nauczyciela, jak i na samodzielną pracę ucznia. W książce zawarto szereg zagadnień podzielonych na cztery główne bloki. W trakcie pracy z rozdziałem pierwszym uczeń pozna podstawowe pojęcia i zagadnienia związane z programowaniem aplikacji desktopowych, w tym zasady programowania algorytmicznego oraz obiektowego. Rozdział drugi podręcznika wprowadza w temat programowania w języku C++, natomiast trzeci skupia się na obiektowym C++ i jego przydatności w tworzeniu aplikacji desktopowych. Uczeń dowie się, jak stosować podstawowe elementy programowania obiektowego, takie jak klasy, obiekty, metody i właściwości. Ostatnia część podręcznika wprowadza w programowanie aplikacji w języku C#.

Technik Informatyk to doskonały, charakteryzujący się wysoką jakością i kompletny zestaw edukacyjny przygotowany przez dysponującego ogromnym doświadczeniem lidera na rynku książek informatycznych — wydawnictwo Helion.

W skład kwalifikacji EE.09 wchodzą także:

- *Kwalifikacja EE.09. Programowanie, tworzenie i administrowanie stronami internetowymi i bazami danych. Część 1. Tworzenie stron internetowych. Podręcznik do nauki zawodu technik informatyk*
- *Kwalifikacja EE.09. Programowanie, tworzenie i administrowanie stronami internetowymi i bazami danych. Część 3. Tworzenie i administrowanie bazami danych. Podręcznik do nauki zawodu technik informatyk*
- *Kwalifikacja EE.09. Programowanie, tworzenie i administrowanie stronami internetowymi i bazami danych. Część 4. Tworzenie aplikacji internetowych. Podręcznik do nauki zawodu technik informatyk*

Podręczniki oraz inne pomoce naukowe należące do tej serii zostały opracowane z myślą o wykształceniu kompetentnych techników, którzy bez trudu poradzą sobie z wyzwaniami w świecie współczesnej informatyki. Wiedza zawarta w serii pomoże zdać egzamin zawodowy i zyskać wiedzę praktyczną, przydatną w przyszłej pracy.

ISBN 978-83-283-4836-3
9 788328 348363