4

Programowanie z użyciem wskaźników

Programowanie w języku C++ jest ściśle związane z wykorzystaniem **wskaźników** (ang. *pointers*). W pełni zasadne jest stwierdzenie, że nie można skutecznie i wydajnie programować w języku C++ bez użycia wskaźników.

W tym rozdziale można się zapoznać jedynie z podstawowymi pojęciami dotyczącymi wskaźników. W szczególności są w nim omawiane wskaźniki do danych podstawowych, takich jak `int` czy `double`. Wskaźniki do innych typów danych, takich jak tablice, struktury, klasy, zostały przedstawione w następnych rozdziałach podręcznika. To samo dotyczy wykorzystania wskaźników jako parametrów i argumentów funkcji. Dlatego też dopiero po wnikliwym zapoznaniu się z materiałem zawartym w dalszych rozdziałach wiedza i umiejętności dotyczące zasad i technik wykorzystania wskaźników w programowaniu w języku C++ będą satysfakcjonujące.

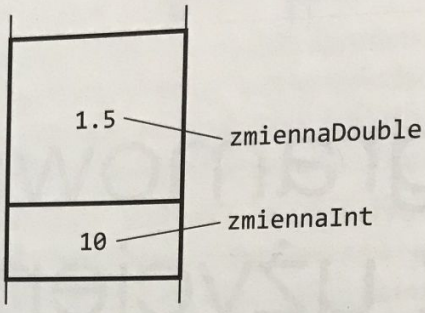
4.1. Operator adresu

Z treści zaprezentowanych w rozdziale 3. jasno wynika, że każdą zmienną zadeklarowaną w programie charakteryzuje:

- jej nazwa — identyfikator,
- typ danych, które można w niej zapamiętać,
- wartość, która faktycznie jest w niej przechowywana.

Zmienne są pamiętane w pamięci operacyjnej komputera w określonych obszarach — blokach. Położenie wspomnianych bloków, czyli ich adresy początkowe, jest zależne od systemu operacyjnego. Rozmiary bloków są zaś determinowane przez typy zmiennych zadeklarowanych w programie.

Na przykład po jawnym zadeklarowaniu i zainicjowaniu w programie zmiennych `int zmiennaInt = 10;` oraz `double zmiennaDouble = 1.5;` fragment pamięci operacyjnej odpowiadający tym zmiennym można przedstawić poglądowo w sposób pokazany na rysunku 4.1.



Rysunek 4.1. Interpretacja bloków w pamięci operacyjnej, w których są przechowywane zmienne

Identyfikatory zmiennych (tj. `zmiennaInt` i `zmiennaDouble`) można potraktować jak nazwy określonych obszarów (bloków) w pamięci operacyjnej. Zawartość tych bloków (`10` i `1.5`) jest zaś związana z wartościami, które są przechowywane w zmiennych. Rozmiar bloków pamięci skojarzonych z zadeklarowanymi zmiennymi odpowiada typom tych zmiennych. Mianowicie zmienna `zmiennaInt` typu `int` zajmuje w pamięci operacyjnej 4 bajty, a zmienna `zmiennaDouble` typu `double` 8 bajtów.

Zmienne należące do predefiniowanych typów danych podstawowych (np. `char`, `bool`, `int`, `double`), które w kodzie programu zostały zadeklarowane w sposób jawny (ang. *explicit declaration*), mają pamięć operacyjną przydzieloną — zaalokowaną w sposób statyczny (ang. *static memory allocation*). Zapotrzebowanie na pamięć wymaganą przez te zmienne jest znane już na etapie kompilacji programu i nie zmienia się w trakcie jego wykonywania.

Należące do typów podstawowych zmienne, które zostały zadeklarowane w sposób jawny, są przechowywane w pamięci operacyjnej komputera w regionie (segmencie) nazywanym **stosem** (ang. *stack*). Zapotrzebowanie programu na pamięć operacyjną na stosie jest ustalane przez kompilator w czasie kompilacji. Proces ten jest nazywany **alokacją pamięci w czasie kompilacji** (ang. *compile-time memory allocation*). Podczas działania programu rozmiar pamięci przydzielonej mu na stosie się nie zmienia.

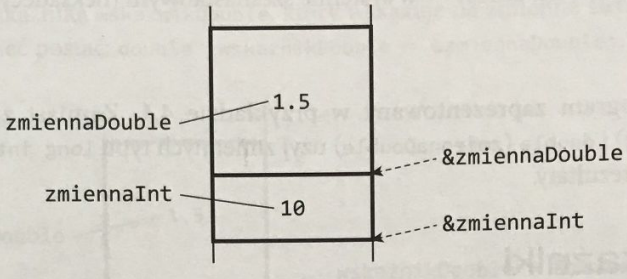
UWAGA

Więcej informacji na temat zmiennych przechowywanych na stosie (np. zmiennych należących do typów danych zdefiniowanych przez programistę), z uwzględnieniem podziału zmiennych na globalne i lokalne, można znaleźć w rozdziale 8. podręcznika, dotyczącym funkcji.

UWAGA

W programowaniu pojęcie stosu oznacza — oprócz segmentu (regionu) w pamięci operacyjnej — dynamiczną strukturę danych.

Każdy z bloków pamięci operacyjnej komputera, w których przechowywane są zadeklarowane zmienne, można zidentyfikować jednoznacznie za pomocą adresu tego bloku. W języku C++ adres bloku pamięci operacyjnej, w którym zapisana jest zmienna o określonej nazwie, można uzyskać za pomocą **operatora adresu** (ang. *address-of operator*, *address operator*): `&` (ang. *ampersand sign*), np. `&zmiennaInt`, `&zmiennaDouble`. Zilustrowano to na rysunku 4.2.



Rysunek 4.2. Interpretacja operatora adresu zmiennej

Wyrażenie `&zmiennaInt` określa adres komórki pamięci operacyjnej stanowiącej początek bloku, w którym przechowywana jest zmienna `zmiennaInt`. Wyrażenie `&zmiennaDouble` określa zaś adres początku bloku pamięci, w którym przechowywana jest zmienna `zmiennaDouble`.

Przykład 4.1

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

int main() {
    // Deklaracja i inicjalizacja zmiennej o nazwie zmiennaInt należącej do typu całkowitego int:
    int zmiennaInt = 10;

    // Prezentacja informacji dotyczących zmiennej zmiennaInt na ekranie monitora:
    cout << "Informacje dotyczące zmiennej zmiennaInt:" << endl;
    // Wyświetlenie wartości przechowywanej w zmiennej zmiennaInt:
    cout << "wartość: " << zmiennaInt << endl;
    // Prezentacja rozmiaru zmiennej zmiennaInt:
    cout << "rozmiar: " << sizeof(zmiennaInt) << endl;
    // Wykorzystanie operatora adresu w celu określenia adresu zmiennej zmiennaInt:
    cout << "adres: " << &zmiennaInt << endl;

    // Deklaracja i inicjalizacja zmiennej o nazwie zmiennaDouble należącej do typu rzeczywistego double:
    double zmiennaDouble = 1.5;
```

```
    // Prezentacja informacji dotyczących zmiennej zmiennaDouble w konsoli na ekranie monitora:
    cout << "Informacje dotyczące zmiennej zmiennaDouble:" << endl;
    cout << "wartość: " << zmiennaDouble << endl;
    cout << "rozmiar: " << sizeof(zmiennaDouble) << endl;
    cout << "adres: " << &zmiennaDouble << endl;

    return 0;
}
```

W przedstawionym programie wykorzystano dwie zmienne należące do typów podstawowych: `zmiennaInt` oraz `zmiennaDouble`. Rozmiary tych zmiennych są wyświetlane na ekranie monitora w bajtach, a ich adresy — w systemie szesnastkowym (heksadecymalnym).

Ćwiczenie 4.1

Zmodyfikuj program zaprezentowany w przykładzie 4.1. Zamiast zmiennych typu `int` (`zmiennaInt`) i `double` (`zmiennaDouble`) użyj zmiennych typu `long int` i `float`. Zinterpretuj uzyskane rezultaty.

4.2. Wskaźniki

Symboliczną reprezentację adresów zmiennych stanowią **zmienne wskaźnikowe** (ang. *pointer variables*), nazywane krótko **wskaźnikami** (ang. *pointers*). Wskaźniki to zwykłe zmienne nazwane (tj. mające identyfikatory), w których można przechowywać adresy innych zmiennych określonego typu.

Zmienna określonego typu (np. `int`, `double`), której adres został zapamiętany we wskaźniku, jest nazywana zmienną wskazywaną przez ten wskaźnik, a typ danych, do którego należy zmienna wskazywana — **typem bazowym** wskaźnika (ang. *pointer base type*).

4.2.1. Deklaracja zmiennej wskaźnikowej

Każdy wskaźnik używany w programie powinien zostać wcześniej zadeklarowany. Postać ogólna deklaracji wskaźnika jest następująca:

```
typ_bazowy *wskaźnik;
```

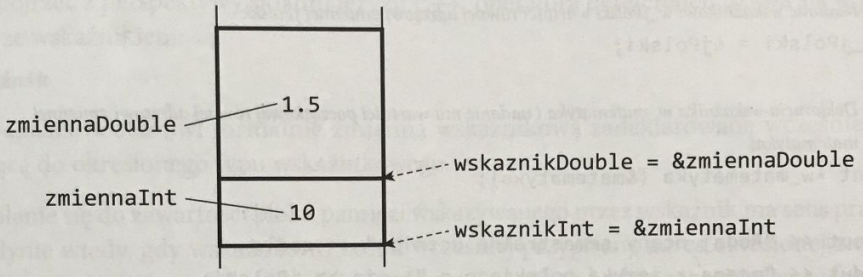
gdzie `typ_bazowy` oznacza określony typ zmiennej (np. `int`, `double`), która może być wskazywana przez `wskaźnik`.

UWAGA

Położenie symbolu `*` w deklaracji wskaźnika jest dowolne. Poprawną składnią deklaracji wskaźnika jest zarówno notacja `typ_bazowy *wskaźnik;`, jak i `typ_bazowy* wskaźnik;` oraz `typ_bazowy * wskaźnik;`. Jednakże zalecaną formą deklaracji wskaźnika jest ta, którą zastosowano w postaci ogólnej powyżej.

Wskaźnik jako zmienna zadeklarowana w programie — co wiąże się z nadaniem jej unikatowej nazwy — należy do określonego **typu wskaźnikowego** (ang. *pointer type*). Typ wskaźnikowy jest określony jednoznacznie za pomocą typu bazowego, o którym była mowa wcześniej. W ogólności typy wskaźnikowe należą do **typów pochodnych** (ang. *derived data types*), ponieważ są definiowane na podstawie typów bazowych.

Deklaracja wskaźnika w programie może być połączona z jego inicjalizacją — tak samo jak w przypadku innych zmiennych. Na przykład deklaracja zmiennej wskaźnikowej o nazwie `wskaznikInt`, w której można zapamiętać adres dowolnej zmiennej typu `int`, jest następująca: `int *wskaznikInt;`. Przypisanie wskaźnikowi `wskaznikInt` adresu zmiennej `zmiennaInt` może zaś mieć postać: `wskaznikInt = &zmiennaInt;`. Analogicznie deklaracja połączona z inicjalizacją wskaźnika `wskaznikDouble`, który wskazuje na zmienną `zmiennaDouble` typu `double`, może mieć postać: `double *wskaznikDouble = &zmiennaDouble;`. Zilustrowano to na rysunku 4.3.



Rysunek 4.3. Interpretacja zmiennej wskaźnikowej (wskaźnika)

Zmienna `zmiennaInt` jest zmienną wskazywaną przez wskaźnik `wskaznikInt`. Typem bazowym wskaźnika `wskaznikInt` jest typ całkowity `int`. Typ wskaźnikowy, do którego należy wskaźnik `wskaznikInt`, można określić (poglądowo) jako `int*` — co oznacza, że dopuszczalnym zbiorem wartości, jakie może przyjmować zmienna `wskaznikInt`, są adresy zmiennych typu całkowitego `int`, który jest typem bazowym dla tego wskaźnika. Zmienną wskazywaną przez wskaźnik `wskaznikDouble` jest zaś `zmiennaDouble`, a typem bazowym tego wskaźnika — typ `double`.

W operacjach na wskaźnikach często wykorzystuje się literał wskaźnikowy (ang. *pointer literal*) o nazwie `nullptr`, który określa wskaźnik pusty (ang. *null-pointer*). Zamiast literału `nullptr` można również używać stałej o nazwie `NULL`, predefiniowanej za pomocą makrodefinicji (ang. *macro-definition*).

UWAGA

Tematyka definiowania stałych za pomocą makrodefinicji została omówiona w podrozdziale 9.2.1 podręcznika.

ała NULL przyjmuje wartość całkowitego 0 (lub 0L), ale może być konwertowana na dowol-
typ wskaźnikowy. Na przykład poprawną instrukcją jest int *wskaznikInt = NULL;, jak
wnież double *wskaznikDouble = NULL;. Przypisanie wskaźnikowi wartości NULL oznacza,
nie wskazuje on na żadną zmienną.

rzykład 4.2

```
nclude <iostream>
ing namespace std;

t main() {
   int jPolski, matematyka;

   // Deklaracja i inicjalizacja wskaźnika w_jPolski:
   int *w_jPolski {}; // wskaźnik w_jPolski zostanie zainicjowany adresem o wartości 0
   // Nadanie wskaźnikowi w_jPolski wartości równej adresowi zmiennej jPolski:
   w_jPolski = &jPolski;

   // Deklaracja wskaźnika w_matematyka i nadanie mu wartości początkowej równej adresowi zmiennej
   // matematyka:
   int *w_matematyka {&matematyka};

   cout << "Podaj oceny semestralne ucznia " << endl;
   cout << "ocena z języka polskiego = "; cin >> jPolski;
   cout << "ocena z matematyki = "; cin >> matematyka;

   cout << endl << "Wprowadzono następujące dane:" << endl;
   cout << "wartość zmiennej jPolski = " << jPolski << endl;
   cout << "adres zmiennej jPolski = " << &jPolski << endl;
   cout << "adres przechowywany we wskaźniku w_jPolski = " << w_jPolski
        << endl;

   cout << "wartość zmiennej matematyka = " << matematyka << endl;
   cout << "adres zmiennej matematyka = " << &matematyka << endl;
   cout << "adres przechowywany we wskaźniku w_matematyka = " << w_matematyka
        << endl;

   return 0;
```

W zaprezentowanym programie przetwarzane są dwie dane: oceny semestralne uzyskane
rzez ucznia z języka polskiego i z matematyki. Oceny te są reprezentowane przez zmien-
e jPolski i matematyka. Wskaźnikom do tych zmiennych nadano nazwy, odpowiednio,
_jPolski i w_matematyka. Inicjalizacja wskaźników została przeprowadzona w notacji
C++11. Wyjście programu polega na wyświetleniu różnych informacji: wartości zmiennych

jPolski i matematyka, adresów tych zmiennych oraz, dla porównania, wartości przechowywanych we wskaźnikach w_jPolski i w_matematyka.

Ćwiczenie 4.2

Zmodyfikuj program zawarty w przykładzie 4.2. Uwzględnij oceny semestralne uzyskane przez ucznia z czterech przedmiotów: języka polskiego, matematyki, języka angielskiego i informatyki. Oblicz średnią arytmetyczną tych ocen i wyświetl wynik na ekranie monitora.

4.2.2. Operator dereferencji

Odwołanie się do zawartości bloku pamięci, w którym przechowywana jest zmienna wskazywana przez wskaźnik, można uzyskać za pomocą **operatora dereferencji** (ang. *dereference operator*), *.

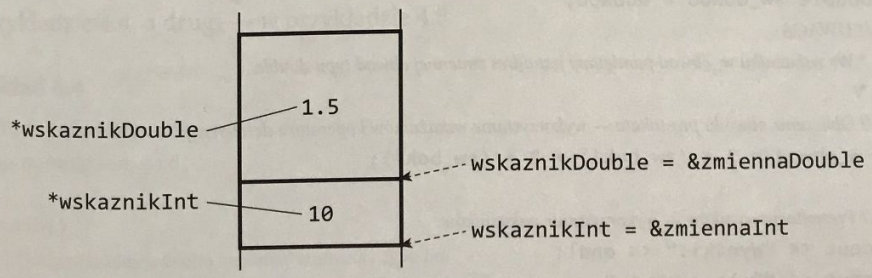
Jeśli spojrzeć z perspektywy składni języka C++, operatora dereferencji używa się w powiązaniu ze wskaźnikiem:

```
*wskaźnik
```

gdzie wskaźnik stanowi formalnie zmienną wskaźnikową zadeklarowaną wcześniej, czyli należącą do określonego typu wskaźnikowego.

Odwołanie się do zawartości bloku pamięci wskazywanego przez wskaźnik ma sens praktyczny jedynie wtedy, gdy wskaźnikowi został wcześniej przypisany adres określonej zmiennej typu bazowego. Wówczas konstrukcja językowa *wskaźnik będzie reprezentować faktyczną wartość zapisaną w zmiennej wskazywanej przez ten wskaźnik.

Na przykład odwołanie się do zawartości bloku pamięci, w której pamiętana jest zmienna o nazwie zmiennaInt wskazywana przez wskaźnik wskaznikInt, ma postać: *wskaznikInt. Analogicznie odwołanie się do wartości zmiennej zmiennaDouble wskazywanej przez wskaźnik wskaznikDouble jest realizowane przez użycie *wskaznikDouble. Zilustrowano to na rysunku 4.4.



Rysunek 4.4. Interpretacja operatora dereferencji *

Jak wynika z rysunku 4.4, wyrażenie *wskaznikInt stanowi pewnego rodzaju umowny alias nazwy zmiennaInt, a wyrażenie *wskaznikDouble — alias nazwy zmiennaDouble.

Przykład 4.3

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

int main() {
    // Deklaracja i inicjalizacja zmiennej bok1 typu double:
    double bok1 = 1;
    // Deklaracja zmiennej wskaźnikowej (wskaźnika) w_bok1 na zmienną typu double:
    double *w_bok1;
    // Przypisanie wskaźnikowi w_bok1 adresu zmiennej bok1:
    w_bok1 = &bok1;

    // Deklaracja i inicjalizacja zmiennej bok2 typu double:
    double bok2 = 2;
    // Deklaracja zmiennej wskaźnikowej (wskaźnika) w_bok2 na zmienną typu double:
    double *w_bok2;
    // Przypisanie wskaźnikowi w_bok2 adresu zmiennej bok2:
    w_bok2 = &bok2;

    // Deklaracja zmiennej pole typu double:
    double pole;
    // Deklaracja i inicjalizacja wskaźnika w_pole:
    double *w_pole = &pole;
    /* UWAGA
     * We wskaźniku w_pole zostaje zapamiętany adres zmiennej pole typu double.
     */
    // Obliczenie pola prostokąta — wykorzystanie wskaźników i operatora dereferencji:
    *w_pole = (*w_bok1) * (*w_bok2); // równoważne instrukcje: pole = bok1 * bok2;

    // Deklaracja zmiennej obwod typu double, deklaracja i inicjalizacja wskaźnika w_obwod:
    double obwod;
    double *w_obwod = &obwod;
    /* UWAGA
     * We wskaźniku w_obwod pamiętany jest adres zmiennej obwod typu double.
     */
    // Obliczenie obwodu prostokąta — wykorzystanie wskaźników i operatora dereferencji:
    *w_obwod = 2 * (*w_bok1) + 2 * (*w_bok2);

    // Prezentacja wyników — wykorzystanie wskaźników:
    cout << "Wyniki:" << endl;
    cout << "Pole wynosi " << *w_pole << endl;
    cout << "Obwód wynosi " << *w_obwod << endl;

    return 0;
}
```

W przykładzie obliczane są pole i obwód prostokąta dla danych wejściowych (długości boków) inicjowanych w programie. Przetwarzanie danych jest realizowane przy użyciu wskaźników. Wyrażenia `*w_bok1`, `*w_bok2` reprezentują dane wejściowe — stanowią aliasy nazw zmiennych, odpowiednio, `bok1` i `bok2`. Wyrażenia `*w_pole` i `*w_obwod` odpowiadają zaś, odpowiednio, polu prostokąta — zmiennej `pole`, i obwodowi — zmiennej `obwod`.

Ćwiczenie 4.3

Zmodyfikuj program z przykładu 4.3 — oprócz pola i obwodu prostokąta wyznacz dodatkowo długość jego przekątnej. Ponadto:

- dane wejściowe do programu niech będą wprowadzane z klawiatury,
- nadanie wartości początkowych wskaźników przeprowadź za pomocą inicjalizacji jednolitej.

UWAGA

Szczególnym rodzajem wskaźników są **wskaźniki ogólne** (ang. *generic pointers, general-purpose pointers*), tzw. wskaźniki typu `void` (ang. *void pointers*). Wskaźniki typu `void` pozwalają na zapamiętanie i przechowanie adresu zmiennej dowolnego typu (np. adresu zmiennej typu `int` lub `double`), ale zarazem nie są skojarzone z żadnym konkretnym, bazowym typem danych. Tym samym dereferencja wskaźników `void` nie jest możliwa. Na przykład po deklaracji `double zmienna;` można przypisać adres tej zmiennej (`&zmienna`) wskaźnikowi typu `void`: `void *wskaznik = &zmienna;`. Jednocześnie nie jest możliwe odwołanie do wartości przechowywanej w zmiennej `zmienna` za pomocą operatora dereferencji `*wskaznik`.

4.2.3. Wskaźniki i słowo kluczowe const

W programowaniu w języku C++ często można się spotkać z deklaracjami wskaźników połączonych ze słowem kluczowym `const`. Mogą wówczas zajść dwa przypadki zależne od położenia słowa kluczowego `const` w deklaracji wskaźnika. Pierwszy z nich zilustrowano w przykładzie 4.4, a drugi — w przykładzie 4.5.

Przykład 4.4

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

int main() {
    // Deklaracja i inicjalizacja zmiennej zmienna1 typu int:
    int zmienna1 = 1;
    // Deklaracja wskaźnika o nazwie wskaznik z modyfikatorem const:
    const int *wskaznik;
    // Przypisanie wskaźnikowi wskaznik adresu zmiennej o nazwie zmienna1:
```

```
    wskaznik = &zmienna1;
    // Odczyt wartości zmiennej wskazywanej przez wskaźnik wskaznik:
    cout << "Wartość zmiennej zmienna1: " << *wskaznik << endl;

    // Instrukcja zawarta w komentarzu poniżej jest błędna!
    // *wskaznik = 10;

    // Deklaracja i inicjalizacja zmiennej zmienna2 typu int:
    int zmienna2 = 2;
    // Przypisanie wskaźnikowi wskaznik adresu zmiennej zmienna2:
    wskaznik = &zmienna2;
    // Odczyt wartości zmiennej wskazywanej przez wskaźnik wskaznik:
    cout << "Wartość zmiennej zmienna2: " << *wskaznik << endl;

    return 0;
}
```

W programie zadeklarowano wskaźnik o nazwie `wskaznik` za pomocą instrukcji `const int *wskaznik;`. W ogólności `wskaznik` może wskazywać na dowolną zmienną typu `int`, ponieważ typ `int` jest jego typem bazowym. Zmiennej `wskaznik` najpierw przypisano adres zmiennej `zmienna1`, a następnie adres zmiennej `zmienna2`.

Modyfikator `const` w deklaracji wskaźnika `wskaznik` powoduje, że wskaźnik ten traktuje zmienną wskazywaną tak, jak gdyby zmienna ta była stałą `const` — pomimo tego, że stałą nie jest. Tym samym za jego pomocą można jedynie odczytać wartość przechowywaną we wskazywanym przez niego bloku pamięci (tutaj: najpierw wartość zmiennej `zmienna1`, a potem `zmienna2`), ale nie jest możliwa modyfikacja (zmiana) tej wartości.

Wskaźnik zadeklarowany w programie zgodnie z postacią ogólną:

```
const typ_bazowy *wskaźnik;
```

jest nazywany **wskaźnikiem na stałą** (ang. *pointer to constant*).

Jak to zademonstrowano w przykładzie 4.4, za pomocą wskaźnika na stałą można wyłącznie odczytywać (ang. *read-only*) wartość zmiennej wskazywanej przez taki wskaźnik. Zapis i/lub modyfikacja wartości zmiennej wskazywanej nie są możliwe. Z drugiej strony we wskaźniku na stałą można zapisywać bez ograniczeń adresy innych zmiennych zgodnych z zadeklarowanym typem bazowym. Zatem taki wskaźnik może wskazywać w programie kolejno różne zmienne.

Ćwiczenie 4.4

Na podstawie przykładu 4.4 napisz program pozwalający obliczyć pole i obwód prostokąta. Długości boków prostokąta mają być wprowadzane z klawiatury, a wyniki wyświetlane na ekranie monitora. Wykorzystaj wskaźniki na stałe `const`.

W przykładzie 4.5 zilustrowano z kolei drugi przypadek, jaki może wystąpić w deklaracji wskaźnika z użyciem słowa kluczowego `const`.

Przykład 4.5

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

int main() {
    int zmienna1 = 1;
    int zmienna2 = 2;

    // Deklaracja i inicjalizacja wskaźnika o nazwie wskaznik z wykorzystaniem słowa kluczowego const:
    int *const wskaznik = &zmienna1;
    /* UWAGA
     * Wskaźnik zadeklarowany z użyciem słowa kluczowego const jak powyżej musi zostać zainicjowany
     * podczas deklaracji. Tutaj: wskaźnikowi o nazwie wskaznik przypisano adres zmiennej zmienna1.
     */
    cout << "Wartość zmiennej zmienna1: " << *wskaznik << endl;

    // Modyfikacja wartości zmiennej wskazywanej przez wskaźnik stały:
    (*wskaznik)++;
    cout << "Zmieniona wartość zmiennej zmienna1: " << *wskaznik << endl;

    // Instrukcja w komentarzu poniżej jest błędna!
    // wskaznik = &zmienna2;

    return 0;
}
```

W programie zadeklarowano i jednocześnie zainicjowano zmienną wskaźnikową o nazwie `wskaznik`. W tym celu wykorzystano instrukcję: `int *const wskaznik = &zmienna1;`. Modyfikator `const` sprawia, że zmienna `wskaznik` nosi znamiona stałej. Oznacza to, że zmienna `wskaznik` nie może w programie zmieniać adresu, na który wskazuje. Jeśli przypisano jej wartość początkową adresu określonej zmiennej (tutaj zmiennej `zmienna1`), adres ten nie może się zmienić w dalszej części programu. Tym samym taki wskaźnik nie może wskazywać na inną zmienną. Z drugiej strony przy użyciu wskaźnika stałego można zarówno odczytywać wartość zmiennej wskazywanej, jak i zapisywać (modyfikować) tę wartość.

W ogólności wskaźnik zadeklarowany w programie zgodnie z postacią ogólną:

```
typ_bazowy *const wskaźnik;
```

jest nazywany **wskaźnikiem stałym** (ang. *constant pointer*).

Wskaźnik stały ma cechę „tylko do odczytu" (ang. *read-only*) w tym znaczeniu, że zmiana jego wartości na inną (tj. przypisanie adresu innej zmiennej) nie jest możliwa. Z drugiej strony wskaźnik stały pozwala zarówno na odczyt, jak i na zmianę (modyfikację) wartości wskazywanej przez niego zmiennej.

UWAGA

W praktyce możliwa jest również kombinacja (połączenie) obu omówionych wcześniej sposobów deklarowania wskaźników wraz ze słowem kluczowym `const`, tj. zadeklarowanie **stałego wskaźnika na stałą** (ang. *constant pointer to constant value*). Na przykład deklaracja: `const int *const wskaznik = &zmienna;` oznacza, że zmienna wskaźnikowa `wskaznik` wskazuje na zmienną, której wartość można wyłącznie odczytywać, i dodatkowo — że wskaźnik ten nie może wskazać w programie na inną zmienną, ponieważ nie można mu przypisać adresu innej zmiennej.

Ćwiczenie 4.5

Na podstawie przykładu 4.5 napisz program pozwalający obliczyć pole i obwód prostokąta. Długości boków prostokąta mają być wprowadzane z klawiatury, a wyniki wyświetlane na ekranie monitora. Wykorzystaj wskaźniki stałe.

4.3. Dynamiczna alokacja pamięci

4.3.1. Przydzielanie pamięci operacyjnej dla zmiennych

Zmienne zadeklarowane w programie bez użycia wskaźników (np. `int zmienna;`) mają statycznie przydzieloną pamięć operacyjną na stosie. Wspomniana alokacja pamięci jest realizowana podczas kompilacji programu (ang. *compile-time memory allocation*). W trakcie działania programu wielkość stosu się nie zmienia. Pozostała część pamięci operacyjnej dostępna w czasie działania programu — po przydzieleniu niezbędnej pamięci przeznaczonej na stos — jest nazywana **stertą** (ang. *heap*).

UWAGA

W dziedzinie programowania termin *sterta*, analogicznie do *stosu*, oprócz segmentu (obszaru) w pamięci operacyjnej oznacza dynamiczną strukturę danych.

Sterta, jako część pamięci operacyjnej komputera, może być również wykorzystywana przez zmienne programowe. Przydzielanie (alokacja) pamięci dla zmiennych na stercie odbywa się w sposób dynamiczny w czasie wykonywania programu. Na przykład w trakcie działania programu można — w zależności od potrzeby — utworzyć nową zmienną określonego typu.

wykonać zadane operacje na tej zmiennej, a w chwili gdy zmienna ta nie jest już potrzebna, zwolnić pamięć przydzieloną dla niej w sposób dynamiczny.

Omawiany proces alokacji pamięci operacyjnej komputera dla zmiennych programowych jest nazywany **alokacją pamięci w czasie wykonywania programu** (ang. *runtime memory allocation*) — w odróżnieniu od alokacji pamięci w czasie kompilacji.

Ten sposób przydzielania pamięci w czasie wykonywania programu ze względu na jego charakterystyczną cechę — tworzenie i niszczenie (usuwanie) zmiennych w dowolnej chwili, w zależności od potrzeby — jest nazywany **dynamiczną alokacją pamięci** (ang. *dynamic memory allocation*). Mechanizm dynamicznego przydzielania/zwalniania pamięci na stercie dla zmiennej (zmiennych) jest kontrolowany przez programistę. Jest to przeciwieństwo statycznego przydzielania pamięci (ang. *static memory allocation*), które jest sterowane przez kompilator w czasie kompilacji programu.

Proces tworzenia, przetwarzania oraz zwalniania pamięci dla zmiennych zaalokowanych w pamięci operacyjnej (na stercie) w sposób dynamiczny jest prowadzony przy wykorzystaniu wskaźników.

4.3.2. Operatory new i delete

Operator `new` służy do utworzenia nowej zmiennej określonego typu. Pamięć operacyjna potrzebna na przechowanie tej zmiennej jest przydzielana (alokowana) dynamicznie na stercie. Jej wielkość (rozmiar) zależy od typu tworzonej zmiennej. Po zaalokowaniu dla zmiennej odpowiedniego bloku (obszaru) pamięci operator `new` zwraca adres tego bloku.

Ogólna postać wyrażenia umożliwiającego utworzenie nowej zmiennej na stercie jest następująca:

```
wskaźnik = new typ_zmiennej;
```

lub

```
wskaźnik = new typ_zmiennej(wartość);
```

gdzie:

- `wskaźnik` — może wskazywać na dane (zmienne) typu `typ_zmiennej`,
- `typ_zmiennej` — typ nowo tworzonej zmiennej. Typ `typ_zmiennej` jest typem bazowym wskaźnika `wskaźnik`,
- `wartość` — wartość początkowa zmiennej. Jeśli zmiennej nie zostanie przypisana wartość początkowa, zostanie ona zainicjowana wartością przypadkową.

Na przykład utworzenie na stercie zmiennej typu `int` wskazywanej przez wskaźnik o nazwie `wsk` można zrealizować za pomocą instrukcji: `int *wsk; wsk = new int;`.

Zwolnienie pamięci przydzielonej dynamicznie dla zmiennej na stercie jest realizowane za pomocą operatora `delete`. Postać ogólna użycia tego operatora jest następująca:

```
delete wskaźnik;
```

gdzie `wskaźnik` wskazuje na blok pamięci, w którym została zaalokowana usuwana zmienna.

Na przykład zwolnienie pamięci zaalokowanej dynamicznie dla zmiennej wskazywanej przez wskaźnik `wsk` można zrealizować za pomocą instrukcji: `delete wsk;`. Zwolnienie pamięci operacyjnej przydzielonej dynamicznie dla zmiennej na stercie jest tożsame z usunięciem tej zmiennej (z pamięci). W przypadku gdy pamięć przydzielona dla zmiennej na stercie nie zostaje zwolniona przy użyciu operatora `delete`, zmienna ta blokuje zaalokowany obszar pamięci do czasu, aż działanie programu się zakończy.

Zmienne alokowane dynamicznie na stercie mają pewne szczególne cechy, które odróżniają je od zmiennych deklarowanych w programie bez użycia wskaźników (a które są przechowywane na stosie). Po pierwsze, zmienne przechowywane na stercie nie mają nazw (identyfikatorów). Tym samym można te zmienne przetwarzać wyłącznie przy użyciu wskaźników i operatora dereferencji (`*`). Po drugie, czas życia zmiennych na stercie jest kontrolowany bezpośrednio przez programistę: są tworzone, jeśli jest taka potrzeba — przy użyciu operatora `new`, i usuwane, jeśli nie są już potrzebne — za pomocą operatora `delete`. Im krótszy jest czas życia zmiennych na stercie, tym gospodarowanie pamięcią operacyjną komputera jest skuteczniejsze i wydajniejsze. Trzecią cechą odróżniającą zmienne na stosie od zmiennych na stercie jest ich wartość początkowa. Zmienne na stosie, które zostały zadeklarowane, ale nie zainicjowane, przyjmują wartość początkową domyślną — zerową, np. `0`, `""`. Niezainicjowane zmienne utworzone na stercie przyjmują z kolei wartość początkową przypadkową.

UWAGA

W co najmniej dwóch sytuacjach dobrym nawykiem programistycznym jest przypisanie wskaźnikowi wartości `NULL`: jeśli wskaźnik został zadeklarowany, ale nie przypisano mu żadnego adresu początkowego (np. `double *wskaznik = NULL;`), i po zwolnieniu pamięci operacyjnej zaalokowanej dla zmiennej na stercie, czyli po zastosowaniu operatora `delete` (np. `delete wskaznik; wskaznik = NULL;`).

Przykład 4.6

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

int main() {
    // Deklaracja zmiennej wskaźnikowej (wskaźnika) w_ocena:
    int *w_ocena;
    // Utworzenie zmiennej typu int wskazywanej przez wskaźnik w_ocena:
    w_ocena = new int;
```

```
    // Przypisanie zmiennej wskazywanej przez wskaźnik w_ocena wartości 4:
    *w_ocena = 4;
    cout << "Ocena: "<< *w_ocena << endl;

    // Usunięcie zmiennej wskazywanej przez wskaźnik w_ocena:
    delete w_ocena;

    return 0;
}
```

Zadeklarowany wskaźnik `w_ocena` może wskazywać na dowolną zmienną typu `int`. Jednakże w programie przypisano mu adres konkretnej zmiennej typu `int`, która została utworzona za pomocą operatora `new`. Pamięć operacyjna dla tej zmiennej została zaalokowana dynamicznie na stercie. W operacji przypisania wartości utworzonej zmiennej wykorzystano operator dereferencji: `*w_ocena = 4;`. Po wyświetleniu wartości zmiennej pamięć przydzielona dla niej dynamicznie jest zwalniana za pomocą operatora `delete`.

Ćwiczenie 4.6

Na podstawie przykładu 4.6 napisz program, w którym zamiast pojedynczej oceny rozpatrywane są cztery oceny semestralne ucznia z następujących przedmiotów: język polski, matematyka, język angielski, informatyka. Dane wejściowe do programu — wartości poszczególnych ocen — należy wprowadzić z klawiatury. Oblicz średnią arytmetyczną tych ocen i zaprezentuj wynik na ekranie. W celu zapamiętania ocen ucznia wykorzystaj zmienne zaalokowane w pamięci operacyjnej w sposób dynamiczny.

Przykład 4.7

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

int main() {
    // Deklaracja zmiennej wskaźnikowej (wskaźnika) na zmienną typu double i przypisanie jej wartości NULL:
    double *w_bok1 = NULL;
    // Utworzenie zmiennej (dynamicznej) typu double wskazywanej przez wskaźnik w_bok1:
    w_bok1 = new double(1);
    /* UWAGA
     * Zmienna na stercie, dla której pamięć została zaalokowana dynamicznie, została zainicjowana wartością 1.
     */

    double *w_bok2 = new double(2);
    /* UWAGA
     * W wyrażeniu powyżej następuje deklaracja wskaźnika w_bok2 oraz jego inicjalizacja.
     * Wspomniana inicjalizacja wskaźnika jest realizowana po zaalokowaniu na stercie obszaru pamięci operacyjnej
     * niezbędnego do przechowania danych typu double i zwróceniu (przez operator new) adresu początku tego obszaru.
```

```
 * Zwrócony adres zostaje następnie przypisany do wskaźnika w_bok2.
 * Zmienna na stercie, dla której pamięć została zaalokowana dynamicznie, została zainicjowana wartością 2.
 */

// Utworzenie zmiennej dynamicznej wskazywanej przez wskaźnik w_pole:
double *w_pole = new double;
// Obliczenie pola prostokąta — wykorzystanie wskaźników i operatora dereferencji:
*w_pole = (*w_bok1) * (*w_bok2);

// Utworzenie zmiennej dynamicznej wskazywanej przez wskaźnik w_obwod:
double *w_obwod = new double;
// Obliczenie obwodu prostokąta — wykorzystanie wskaźników i operatora dereferencji:
*w_obwod = 2 * (*w_bok1) + 2 * (*w_bok2);

// Zwolnienie pamięci zaalokowanej dla zmiennych wskazywanych przez wskaźniki w_bok1 i w_bok2:
delete w_bok1;
delete w_bok2;

// Prezentacja wyników — wykorzystanie wskaźników i operatora dereferencji:
cout << "Wyniki:" << endl;
cout << "Pole wynosi " << *w_pole << endl;
cout << "Obwód wynosi " << *w_obwod << endl;

// Zwolnienie pamięci zaalokowanej dla zmiennych wskazywanych przez wskaźniki w_pole i w_obwod:
delete w_pole;
delete w_obwod;

return 0;
}
```

W przykładzie 4.7 obliczane są pole i obwód prostokąta. Dane wejściowe (długości boków) są przechowywane w zmiennych przechowywanych na stercie, dla których pamięć operacyjna została zaalokowana dynamicznie. Dostęp do tych zmiennych uzyskuje się za pomocą wskaźników, odpowiednio, `w_bok1` i `w_bok2`. Długości boków zostały zainicjowane w programie wartościami, odpowiednio, `1` i `2`. Pole i obwód prostokąta również są w programie reprezentowane przez zmienne utworzone na stercie, dla których pamięć została przydzielona dynamicznie. Zmienne te są wskazywane przez wskaźniki `w_pole` (pole prostokąta) i `w_obwod` (obwód prostokąta). Przetwarzanie tych zmiennych odbywa się przy wykorzystaniu wyrażeń z operatorem dereferencji `*`, tj. `*w_pole` oraz `*w_obwod`.

Ćwiczenie 4.7

Zmodyfikuj program zawarty w przykładzie 4.7 — zamiast pola i obwodu prostokąta oblicz pole i obwód kwadratu.

UWAGA

Ze wskaźnikami związany jest mechanizm tzw. **arytmetyki wskaźników** (ang. *pointer arithmetics*). Pomimo że dotyczy on (m.in.) danych typów podstawowych, takich jak `int` czy `float`, szczególną rolę odgrywa w przetwarzaniu tablic. Dlatego ten temat został omówiony w dalszej części podręcznika — w rozdziale 5., dotyczącym tablic.

UWAGA

Wskaźniki często są parametrami i argumentami funkcji. Ta tematyka została szczegółowo omówiona w rozdziale 8., dotyczącym funkcji.

4.4. Pytania i zadania kontrolne

4.4.1. Pytania

1. Czy adres określonej zmiennej w pamięci operacyjnej oznacza to samo, co wskaźnik na tę zmienną?
2. Podaj definicję wskaźnika.
3. Czym różni się alokacja pamięci operacyjnej dla zmiennych na stosie od dynamicznej alokacji pamięci dla zmiennych na stercie?
4. Czy zmienne, dla których pamięć jest przydzielana dynamicznie, mają nazwy?
5. W jaki sposób można odwoływać się do zadeklarowanych zmiennych przechowywanych na stosie?
6. Czy słowo kluczowe `const` może występować w deklaracji wskaźnika? Jeśli tak, to jaki to ma wpływ na cechy tego wskaźnika?

4.4.2. Zadania

1. Napisz program umożliwiający obliczenie pola powierzchni i obwodu koła. Dane wejściowe mają być wprowadzane z klawiatury. Wyniki zaprezentuj na ekranie monitora w konsoli.

   Dane wejściowe (promień koła) oraz wyniki (pole i obwód koła) program powinien zapamiętywać w zmiennych zadeklarowanych w sposób jawny — przechowywanych w pamięci operacyjnej na stosie. Przetwarzanie danych (obliczenia) zrealizuj przy wykorzystaniu wskaźników i operatora dereferencji.

2. Zrób tak samo jak w zadaniu 1., z tym że promień oraz pole i obwód koła powinny być zapamiętywane w zmiennych zaalokowanych w pamięci operacyjnej w sposób dynamiczny (na stercie).

3. Napisz program umożliwiający obliczenie objętości, pola powierzchni bocznej oraz długości wszystkich krawędzi prostopadłościanu. Dane wejściowe zainicjuj w programie. Wyniki niech będą wyświetlane w konsoli. Przetwarzanie danych wykonaj przy użyciu wskaźników do jawnie zadeklarowanych zmiennych przechowywanych w pamięci operacyjnej na stosie oraz operatora dereferencji.
4. Zrób tak samo jak w zadaniu 3., z tym że do przechowania danych wejściowych i wyników obliczeń wykorzystaj zmienne, dla których pamięć operacyjna została zaalokowana dynamicznie na stercie.
5. Napisz program pozwalający sprawdzić, czy liczba całkowita o wartości wprowadzonej z klawiatury jest parzysta, czy nieparzysta. Wykorzystaj zmienne, dla których pamięć operacyjna jest przydzielana dynamicznie.
6. Napisz program — prosty kalkulator logiczny — pozwalający obliczyć iloczyn i sumę logiczną dla dwóch danych logicznych o wartościach wprowadzonych z klawiatury. Przyjmij, że wartość danej wejściowej `1` reprezentuje wartość logiczną `true`, natomiast `0` — `false`. Wykorzystaj:
   - wskaźniki do stałych zadeklarowane jako `const int*` (wariant pierwszy),
   - wskaźniki stałe zadeklarowane jako `int *const` (wariant drugi).