helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

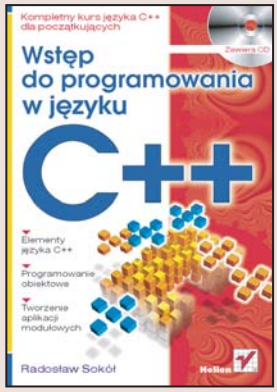
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Wstęp do programowania w języku C++



Autor: Radosław Sokół
ISBN: 83-246-0179-1
Format: B5, stron: 400

Języki C i C++ to bardzo uniwersalne platformy programistyczne o ogromnych możliwościach. Wykorzystywane są do tworzenia systemów operacyjnych i oprogramowania użytkowego. Dzięki niskiemu poziomowi abstrakcji nadają się również do tworzenia programów rezydentnych i sterowników urządzeń. C++, opracowany w latach 1983-1985, jest w pełni obiektowym językiem programowania pozwalającym na prostą implementację nawet rozbudowanych algorytmów i struktur danych. Jego popularność wyraża się w ogromnej liczbie aplikacji tworzonych za jego pomocą, bogactwie dostępnych narzędzi programistycznych i tysiącach witryn WWW poświęconych temu językowi.

Książka „Wstęp do programowania w języku C++" to podręcznik opisujący zasady tworzenia aplikacji za pomocą C++. Dowiesz się z niej, jak zainstalować kompilator MinGW i edytor kodu źródłowego Crimson Edit, poznasz podstawowe elementy i konstrukcje języka C++ i w oparciu o zdobytą wiedzę napiszesz proste programy. Nauczysz się implementować operacje wejścia i wyjścia, zarządzać pamięcią i tworzyć mechanizmy obsługi błędów. Przeczytasz także o programowaniu obiektowym i wykorzystasz mechanizmy obiektowe w swoich aplikacjach.

- Instalacja oprogramowania narzędziowego
- Proces tworzenia programów w języku C++
- Deklarowanie zmiennych
- Sterowanie przebiegiem działania programu
- Struktury danych
- Operacje wejścia i wyjścia
- Przydzielanie pamięci i stosowanie wskaźników
- Programowanie obiektowe
- Korzystanie z szablonów
- Obsługa wyjątków
- Budowanie aplikacji wielomodułowych

Jeśli chcesz łatwo i szybko opanować podstawy C++, sięgnij po tę książkę

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Wstęp 9

Część I Oprogramowanie narzędziowe 15

Rozdział 1. Instalacja pakietu MinGW 17
Instalacja pakietu MinGW 17
Konfiguracja systemu w celu łatwego korzystania z narzędzi pakietu MinGW 22
Testowanie działania pakietu MinGW 24

Rozdział 2. Crimson Editor: edytor programisty 27
Instalacja edytora Crimson Editor 29
Uruchamianie edytora 31
Dostosowywanie programu do własnych potrzeb 32
Przechodzenie do wybranego wiersza programu 35

Rozdział 3. Tworzenie i kompilacja programów w języku C++ 37
Translator, interpreter, kompilator 37
Translator 37
Interpreter 37
Kompilator 38
Kompilator JIT 39
Problem jajka i kury 39
Twój pierwszy program 40
Kompilacja programu 43
Opcje kompilacji 43
Uruchamianie skompilowanego programu 45
Podsumowanie 45

Część II Język C++ 47

Rozdział 4. Wprowadzanie, wyprowadzanie i przechowywanie danych 49
Funkcja main() 50
Umieszczanie tekstu na ekranie 52
Komentarze 56
Zmienne 57
Deklaracje zmiennych 58
Zmienne typu rzeczywistego 61

Zmienne logiczne ... 62
Zmienne łańcuchowe ... 63
Wypisywanie zawartości zmiennych ... 64
Rzutowanie typów zmiennych ... 67
Badanie rozmiaru zmiennych w pamięci ... 68
Pobieranie danych z klawiatury ... 69
Podsumowanie ... 73

Rozdział 5. Tablice, pętle i wyrażenia warunkowe ... 75
Tablice ... 76
Deklarowanie zmiennych tablicowych ... 76
Zapisywanie i odczytywanie elementów tablicy ... 77
Tablice wielowymiarowe ... 77
Zmienne łańcuchowe jako tablice ... 82
Skracanie wyrażeń algebraicznych ... 83
Wyrażenia warunkowe ... 86
Instrukcja if ... 88
Uproszczone wyrażenie warunkowe ... 91
Pętle ... 93
Pętle typu while ... 94
Pętle typu do…while ... 96
Pętle typu for ... 98
Przerywanie pętli ... 101
Podsumowanie ... 102

Rozdział 6. Programowanie proceduralne ... 105
Deklarowanie i wywoływanie podprogramów ... 106
Parametry i zmienne w podprogramach ... 109
Zmienne globalne i lokalne ... 109
Statyczne zmienne lokalne ... 113
Przekazywanie parametrów podprogramom ... 115
Parametry domyślne ... 119
Zwracanie danych przez funkcję ... 120
Zwracanie danych poprzez parametry ... 124
Parametry tekstowe ... 127
Przerywanie działania programu z wnętrza podprogramu ... 128
Programy przykładowe ... 129
Zamiana kropki dziesiętnej na przecinek ... 130
Eleganckie wczytywanie tekstu z klawiatury ... 134
Gra „Tor saneczkowy” ... 137
Podsumowanie ... 143

Rozdział 7. Wyliczenia, struktury, napisy ... 145
Wyliczenia ... 145
Definiowanie wyliczenia ... 146
Wartości liczbowe wyliczeń ... 147
Deklarowanie i używanie wyliczeń ... 148
Wyrażenie warunkowe switch ... 148
Zamiana wyliczeń na liczby ... 156
Struktury danych ... 157
Definiowanie struktury danych ... 158
Deklarowanie zmiennej strukturalnej ... 159
Odwoływanie się do struktur i elementów struktury ... 160
Kopiowanie struktur ... 161

Napisy ... 161
Kopiowanie zawartości zmiennych łańcuchowych ... 162
Dołączanie tekstu do zmiennej ... 163
Porównywanie tekstu ... 164
Wyszukiwanie tekstu ... 165
Podsumowanie ... 168

Rozdział 8. Operacje wejścia-wyjścia ... 171
Standardowe wejście i standardowe wyjście ... 172
Standardowe wyjście ... 172
Standardowe wyjście komunikatów błędów ... 174
Standardowe wejście ... 175
Tworzenie filtru danych ... 177
Parametry działania programu ... 182
Parametry podawane przy uruchamianiu ... 182
Zwracanie kodu błędu ... 185
Dostęp do plików ... 185
Własne potoki danych ... 186
Otwieranie pliku ... 186
Zamykanie pliku ... 188
Testowanie faktu otwarcia pliku ... 188
Sprawdzanie faktu dotarcia do końca pliku ... 189
Odczytywanie i zapisywanie danych ... 189
Program szpaltujący ... 190
Zapisywanie i odczytywanie struktur danych ... 195
Zmiana aktualnej pozycji wewnątrz pliku ... 201
Zapisywanie i odczytywanie napisów ... 208
Podsumowanie ... 212

Rozdział 9. Pamięć operacyjna i wskaźniki ... 215
Informacje podstawowe ... 216
Typy bloków pamięci ... 216
Terminologia ... 216
Wskaźniki ... 217
Deklarowanie wskaźników ... 218
Wskazywanie na zmienne ... 218
Odczytywanie i zapisywanie danych wskazywanych przez wskaźnik ... 220
Wskaźniki a struktury ... 221
Wskaźniki a tablice ... 221
Wskaźniki a zmienne łańcuchowe ... 222
Operacje na wskaźnikach ... 222
Pamięć przydzielana dynamicznie ... 225
Alokowanie bloku pamięci ... 227
Dealokowanie bloku pamięci ... 229
Wypełnianie i zerowanie bloku pamięci ... 232
Kopiowanie bloku pamięci ... 234
Dynamiczne struktury danych ... 236
Lista jednostronnie łączona ... 237
Lista dwustronnie łączona ... 241
Stos ... 245
Podsumowanie ... 248

Rozdział 10. Programowanie obiektowe 251
Struktury danych 252
Deklaracja struktury 252
Konstruktor 252
Konstruktory pobierające parametry 253
Konstruktor z parametrami domyślnymi 255
Destruktor 255
Funkcje składowe 258
Dziedziczenie 264
Dziedziczenie proste 264
Prawa dostępu do pól i funkcji 267
Struktury a klasy 270
Dziedziczenie wielokrotne 271
Dziedziczenie a konstruktory 272
Dziedziczenie a destruktory 273
Pełna wersja programu-magazynu 273
Nakrywanie funkcji i funkcje wirtualne 280
Nakrywanie funkcji 280
Funkcje nakryte a wskaźniki do obiektów 285
Funkcje wirtualne 287
Funkcje prawdziwie wirtualne 289
Operator this 290
Przeciążanie operatorów 291
Tworzenie operatorów 292
Operatory ++ i -- 295
Operator indeksowy [] 297
Konstruktor kopiujący 299
Operator przypisania a konstruktor kopiujący 300
Statyczne pola i funkcje składowe klas 301
Podsumowanie 303

Rozdział 11. Szablony C++ 309
Szablony funkcji 310
Szablony klas 313
Tworzenie szablonu klasy 314
Szablony z parametrami 314
Struktury danych realizowane za pomocą szablonów 315
Lista dwustronnie łączona 316
Stos 325
Podsumowanie 328

Rozdział 12. Obsługa sytuacji wyjątkowych 331
Czym jest sytuacja wyjątkowa 332
Czy obsługa sytuacji wyjątkowych jest szybka 332
Tworzenie bloku instrukcji mogących spowodować błąd 333
Tworzenie bloku instrukcji obsługujących błędy 334
Generowanie własnych sytuacji wyjątkowych 334
Różnicowanie obsługi sytuacji wyjątkowych 337
Zmienne i obiekty a blok instrukcji try 339
Dynamiczna alokacja pamięci a blok instrukcji try 341
Sytuacje wyjątkowe a destruktory 343
Sytuacje wyjątkowe a konstruktory 343
Przekazywanie informacji o przyczynie zgłoszenia sytuacji wyjątkowej 343
Podsumowanie 347

Rozdział 13. Budowanie programów wielomodułowych 349
Testowanie funkcjonowania 349
Asercje 350
Kompilowanie programu w wersji finalnej 352
Zastępowanie asercji własnymi fragmentami kodu 354
Uruchamianie zewnętrznych programów 356
Dzielenie programów na moduły 358
Wydzielanie modułu bibliotecznego 358
Kompilowanie programu wielomodułowego 361
Pliki nagłówkowe 363
Wielomodułowe programy obiektowe 364
Zmienne globalne w programach wielomodułowych 367
Przestrzenie nazw 368
Tworzenie przestrzeni nazw 368
Używanie przestrzeni nazw 369
Wymuszanie użycia przestrzeni nazw 371
Podsumowanie 373

Dodatki 375

Dodatek A Tabela kodów znaków ASCII 377

Dodatek B Polskie znaki diakrytyczne w programach pisanych w języku C++ ... 383

Skorowidz 387

Rozdział 5.

Tablice, pętle i wyrażenia warunkowe

W poprzednim rozdziale nauczyłeś się tworzyć programy o tak zwanym przebiegu liniowym. W przypadku takich programów komputer rozpoczyna wykonywanie ich kodu w ściśle określonym miejscu, realizuje kod składający się na kolejne wyrażenia w sposób sekwencyjny (jedno po drugim), po czym dochodzi do końca bloku instrukcji i kończy działanie programu.

W realnych zastosowaniach oprogramowania takie programy należą do rzadkości. Program musi prawie zawsze reagować na pewne specjalne sytuacje, wykonując jeden z kilku bloków instrukcji w zależności od tego, jaki zestaw warunków spełniają dane wejściowe lub pośrednie wyniki obliczeń. Podczas wprowadzania danych przez użytkownika program powinien weryfikować podawane informacje i zgłaszać wszelkie nieprawidłowości. Niektóre zastosowania komputerów wymagają też przeprowadzania powtarzalnych obliczeń na dużych blokach danych tego samego typu. Widzisz chyba, że próba sprostania wszystkim tym wymaganiom za pomocą programu o przebiegu liniowym byłaby z góry skazana na niepowodzenie.

Język C++, jak każdy rozbudowany język wysokiego poziomu, daje Ci jednak do dyspozycji zestaw narzędzi umożliwiających wprowadzenie do programu fragmentów wykonywanych w sposób powtarzalny (wielokrotny, w pętli) lub warunkowy (zależny od zestawu warunków) oraz upraszczających obróbkę dużych ilości danych tego samego typu. Narzędzia te to:

- tablice — umożliwiają przechowywanie pod jedną nazwą wielu zmiennych tego samego typu i odwoływanie się do kolejnych elementów za pomocą liczby wyrażającej kolejny numer elementu wewnątrz tablicy;
- struktury pętli — umożliwiają wielokrotne wykonanie bloku instrukcji przy zmieniających się parametrach pracy (na przykład ze zmieniającą się wartością jednej zmiennej służącej do wybierania konkretnego elementu z tablicy danych);

- wyrażenia warunkowe — umożliwiają wykonywanie jednego z dwóch (lub więcej) bloków instrukcji w zależności od wartości wskazanych zmiennych i warunków, które te wartości spełniają.

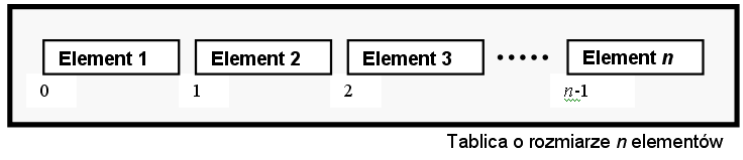
Tablice

Tak naprawdę o tablicach dowiedziałeś się już co nieco z poprzedniego rozdziału, przy okazji nauki korzystania ze zmiennych łańcuchowych. Zmienna łańcuchowa przechowująca napis składający się z pojedynczych znaków jest niczym więcej, jak tablicą znaków, pod jedną nazwą grupującą wiele pojedynczych elementów typu `char`. Definicja:

```
char Napis[200];
```

oznacza, że pod nazwą `Napis` przechowywanych będzie 200 elementów typu `char`, numerowanych kolejnymi liczbami od 0 do 199. Taki sposób numerowania, w którym pierwszy element zawsze ma indeks 0, w świecie komputerów nie jest niczym niezwykłym, a w językach C oraz C++ jest normą. Aby efektywnie korzystać z tablic, **musisz** wpoić sobie ten sposób numerowania (czyli — w języku programistów — **indeksowania**) elementów tablic (rysunek 5.1).

Rysunek 5.1. *Elementy tablicy i sposób ich indeksowania*





Tablice (inaczej nazywane **zmiennymi tablicowymi**) są komputerowym odpowiednikiem macierzy używanych w matematyce.

Deklarowanie zmiennych tablicowych

Skoro okazuje się, że znasz już doskonale sposób deklarowania tablic, możesz spróbować stworzyć tablicę elementów innego typu niż `char`. Spróbuj na przykład stworzyć zestaw zmiennych tablicowych składających się na bazę danych o ocenach ucznia szkoły:

```
float Oceny[25];
char NazwyPrzedmiotow[25][50];
float Srednia;
unsigned int LiczbaPrzedmiotow = 0;
```

Tak, znów nieco utrudniłem zadanie. Drugi wiersz powyższego przykładu deklaruje zmienną przechowującą 25 tablic (numerowanych od 0 do 24) składających się z 50 elementów typu `char`. Może Ci się to wydać skomplikowane, jednak równie dobrze możesz zinterpretować ten zapis jako deklarację zmiennej `NazwyPrzedmiotow` składającej się z 25 napisów po 50 znaków każdy.

W przypadku tablic rzadko stosuje się przypisywanie poszczególnym elementom wartości początkowych — przede wszystkim dlatego, że wprowadzanie z klawiatury setek lub tysięcy liczb nie jest ani przyjemne, ani najczęściej w ogóle potrzebne. Tablice zwykle zapełnia użytkownik programu w czasie jego działania (przez wprowadzanie tylu danych, ile jest potrzebne — reszta tablicy może leżeć odłogiem); mogą być też wypełniane wynikami obliczeń. Gdybyś koniecznie chciał wypełnić tablicę konkretnymi wartościami, odpowiedni zapis wyrażenia wygląda następująco:

```
float Oprocentowanie[4] = { 1.0, 1.4, 1.6, 3.4 };
```

Zapisywanie i odczytywanie elementów tablicy

Odwoływanie się do pojedynczych elementów zmiennej tablicowej jest niewiele bardziej skomplikowane niż odwoływanie się do pojedynczej zmiennej. Wystarczy, abyś za nazwą zmiennej umieścił zamknięty w nawiasach indeks (numer kolejnego elementu), a odwołasz się tylko do jednego, wybranego elementu tablicy:

```
int m[10];
m[5] = 32;
printf("Szosty element tablicy m ma wartość %i.\n", m[5]);
```



To nie jest błąd. Jeszcze raz przypominam, że pierwszy element tablicy ma indeks 0 (`m[0]`), zaś element `m[5]` jest szóstym elementem zmiennej tablicowej `m`.

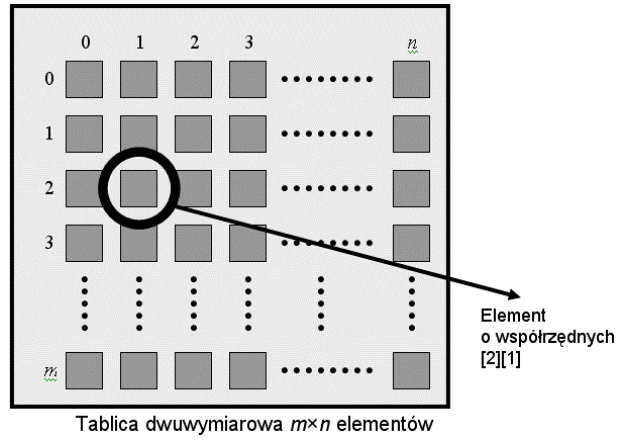
Również bezpośrednie odczytanie elementu tablicy z klawiatury za pomocą funkcji `scanf()` nie wymaga specjalnych zabiegów:

```
int m[10];
scanf("%i", &m[2]);
printf("Trzeci element tablicy m ma wartość %i.\n", m[2]);
```

Tablice wielowymiarowe

Tablica wielowymiarowa to tablica składająca się z wielu tablic. Najprostszym przykładem tablicy wielowymiarowej jest tablica dwuwymiarowa, z którą zetknąłeś się przed chwilą. Tablica dwuwymiarowa to zbiór elementów indeksowanych nie jedną, a dwiema liczbami: jedna określa numer wiersza, a druga — numer kolumny, w której znajduje się szukany element (rysunek 5.2).

W identyczny sposób możesz stworzyć tablice trój- lub czterowymiarowe, w których odwołanie się do każdego z elementów wymaga podania trzech lub czterech liczb-indeksów. Można nawet wyobrazić sobie sensowne zastosowanie tak skomplikowanej tablicy — wyobraź sobie, że budujesz bazę danych służącą do przechowywania informacji o zajętości miejsc w pociągach, przy czym każdy z 20 pociągów może mieć do 10 wagonów, w których znajdują się 3 przedziały po sześć miejsc. Treść takiego programu znajdziesz na listingu 5.1, a wynik jego działania — na rysunku 5.3.



Rysunek 5.2. *Tablica dwuwymiarowa*

Listing 5.1. *Program informujący o dostępności miejsca w pociągu*

```
#include <stdio.h>

int main()
{

   unsigned short int pociag, wagon, przedzial, miejsce;
   bool MiejsceZajete[20][10][3][6]; // 20 pociągów po 10 wagonów
                                     // po 3 przedziały po 6 miejsc.

   // *** Odczytywanie danych

   printf("Wyszukiwanie wolnych miejsc\n"
          "---------------------------\n\n");

   printf("Podaj numer pociagu (1-20):\t");
   scanf("%hu", &pociag);

   printf("Podaj numer wagonu (1-10):\t");
   scanf("%hu", &wagon);

   printf("Podaj numer przedzialu (1-3):\t");
   scanf("%hu", &przedzial);

   printf("Podaj numer miejsca (1-6):\t");
   scanf("%hu", &miejsce);

   // *** Skoryguj indeksy -- człowiek podaje od 1 w górę,
   // *** komputer chce od 0 w górę.

   pociag = pociag - 1;
   wagon = wagon - 1;
   przedzial = przedzial - 1;
   miejsce = miejsce - 1;

   // *** Wyświetl informację o zajętości miejsca.
```

```
    printf("\nStan zajetosci miejsca: %i\n",
       MiejsceZajete[pociag][wagon][przedzial][miejsce]);

    return 0;

}
```



Program przykładowy 01.cpp

Rysunek 5.3. *Efekt działania programu*

```
G:\Helion\cpp\samples\r05>g++ 01.cpp -o 01.exe

G:\Helion\cpp\samples\r05>01
Wyszukiwanie wolnych miejsc
---------------------------

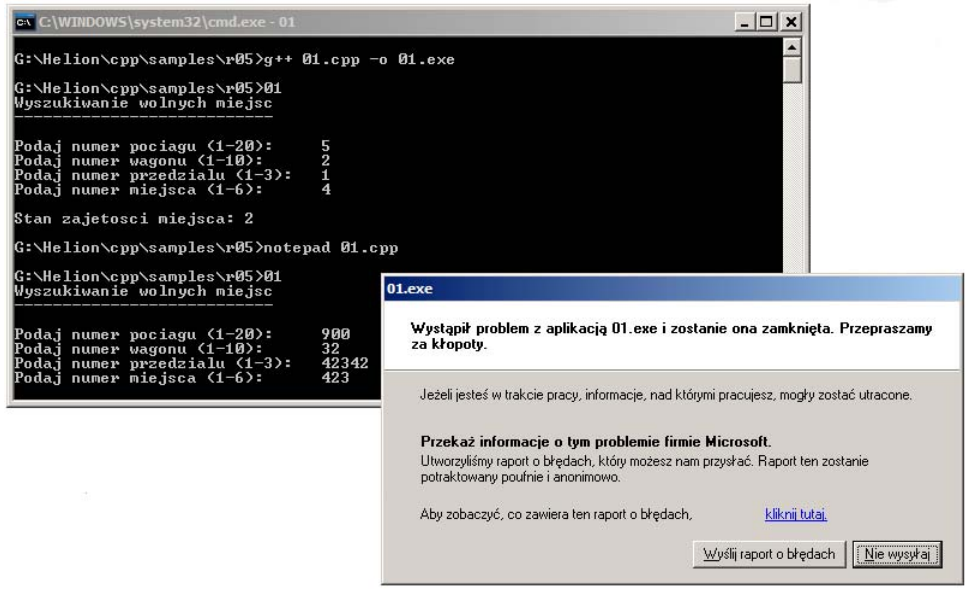
Podaj numer pociagu (1-20):     5
Podaj numer wagonu (1-10):      2
Podaj numer przedzialu (1-3):   1
Podaj numer miejsca (1-6):      4

Stan zajetosci miejsca: 2

G:\Helion\cpp\samples\r05>_
```

Użyteczność tego programu jest żadna, ponieważ dane o zajętości miejsc nie są nigdy wprowadzane. To, czy miejsce zostanie zgłoszone jako wolne (stan zajętości miejsca równy 0), czy zajęte (stan zajętości miejsca różny od 0) jest kwestią przypadku — zawartość zmiennych, do których nigdy nic nie zapisałeś jest całkowicie przypadkowa. Program jednak doskonale ilustruje trzy bardzo istotne zagadnienia, z którymi będziesz musiał się zmagać:

- Najprostszym problemem jest brak obsługi typu danych `bool` przez funkcję `printf()`. Aby w prosty sposób wyprowadzić zawartość zmiennej tego typu na ekran, należy posłużyć się kodem formatującym `%i`, jednak jego użycie spowoduje wyprowadzenie liczby: 0 dla wartości `false` oraz dowolnej innej (różnej od zera) dla wartości `true`. Na szczęście za pomocą wyrażeń warunkowych można ten kłopot szybko i elegancko rozwiązać, o czym przekonasz się nieco dalej w tym rozdziale.
- Program nie sprawdza podawanych numerów pociągów, wagonów czy miejsc. Jeżeli uruchomisz go jeszcze raz i podasz absolutnie bezsensowne indeksy, ujrzysz okno informujące o wystąpieniu błędu w aplikacji i program zostanie natychmiast przerwany przez system operacyjny (rysunek 5.4). Dopiero gdy nauczysz się stosować wyrażenia warunkowe, dowiesz się, jak zapobiegać takim sytuacjom.
- Przeliczanie indeksów z postaci logicznej dla człowieka na postać odpowiednią dla komputera i *vice versa*. Użytkownik spodziewa się, że kolejne pociągi numerowane są liczbami od 1 w górę, podczas gdy komputer bezwzględnie wymaga, by elementy tablicy odpowiadające pociągom nosiły indeksy od zera w górę. Za każdym razem, gdy wymagasz od użytkownika podania numeru elementu, musisz go skorygować, zmniejszając go o jeden, i na odwrót — gdy wyświetlasz z jakichś powodów numer elementu, powinieneś dodać do niego jeden.



Rysunek 5.4. *Tak kończy się odwoływanie do nieprawidłowych indeksów tablicy*

Podczas pisania własnych programów pamiętaj, by nie nadużywać tablic wielowymiarowych. Stosuj się do następujących zasad:

- tablice jednowymiarowe stosowane są niezwykle często — choćby jako zmienne łańcuchowe. Nie bój się ich i gdy tylko musisz zapamiętać kilka kolejnych elementów tego samego typu i o tym samym znaczeniu — użyj tablicy;
- tablice dwuwymiarowe są przydatne na przykład przy zapamiętywaniu danych typu kartograficznego — kształtu mapy pisanej przez Ciebie gry komputerowej czy położenia przedmiotów leżących w pokoju. Specjalnym przypadkiem są też tablice przechowujące zmienne łańcuchowe. Tablic dwuwymiarowych powinieneś używać zawsze tam, gdzie wydają się naturalne i gdzie nie będziesz mieć kłopotów z ich używaniem;
- tablice wyższych rzędów rzadko wykorzystuje się w praktyce. Tego typu zmienne tablicowe stosuj tylko wtedy, gdy wydaje Ci się to absolutnie konieczne.

Program przedstawiony na listingu 5.1 jest poprawny i przejrzysty, ale równie dobrze można by go zapisać bez korzystania z tablicy czterowymiarowej — w końcu każde z miejsc w pociągu może zawierać zapisany w sobie numer wagonu i przedziału. W takim przypadku wystarczy podać numer pociągu i numer miejsca, a program sam wyznaczy numer wagonu i przedziału oraz poinformuje, czy miejsce jest wolne, czy zajęte. Zmodyfikowany program zaprezentowany został na listingu 5.2, a wynik jego działania — na rysunku 5.5.

Listing 5.2. *Program informujący o stanie zajętości miejsca w pociągu napisany z wykorzystaniem tablicy dwuwymiarowej*

```
#include <stdio.h>

int main()
{

   unsigned short int pociag, wagon, przedzial, miejsce;
   bool MiejsceZajete[20][180]; // 20 pociągów po 10 wagonów
                                // po 3 przedziały po 6 miejsc,
                                // czyli 20 pociągów po 180 miejsc.

   // *** Odczytywanie danych.

   printf("Wyszukiwanie wolnych miejsc\n"
          "---------------------------\n\n");

   printf("Podaj numer pociagu (1-20):\t");
   scanf("%hu", &pociag);

   printf("Podaj numer miejsca (1-180):\t");
   scanf("%hu", &miejsce);

   // *** Skoryguj indeksy -- człowiek podaje od 1 w górę,
   // *** komputer chce od 0 w górę.

   pociag = pociag - 1;
   miejsce = miejsce - 1;

   // *** Przeliczanie numeru miejsca na numer wagonu i przedziału.
   // *** Od razu koryguj numery o jeden!

   // W pociągu jest 10 wagonów po 18 miejsc.
   wagon = (miejsce / 18) + 1;

   // W wagonie są 3 przedziały po 6 miejsc.
   przedzial = ((miejsce % 18) / 6) + 1;

   // *** Wyświetl informację o położeniu i zajętości miejsca.
   printf("\nMiejsce znajduje sie w wagonie nr %i i przedziale nr %i.\n"
          "Stan zajetosci miejsca: %i\n",
             wagon, przedzial,
             MiejsceZajete[pociag][miejsce]);

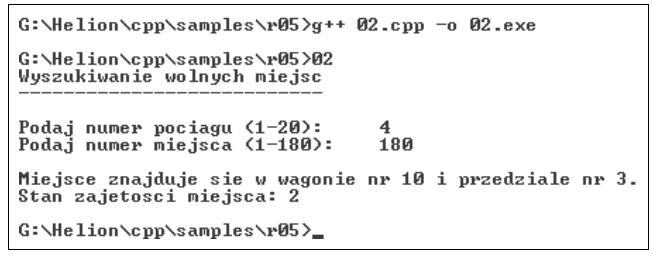
   return 0;

}
```



Program przykładowy 02.cpp

Rysunek 5.5. *Efekt działania programu*



Zmienne łańcuchowe jako tablice

Na początku tego podrozdziału wspominałem, że ze zmiennymi tablicowymi zetknąłeś się już — nieświadomie — w czasie poznawania zmiennych łańcuchowych. Dotychczas zmiennych łańcuchowych używałeś jedynie na trzy sposoby:

- nadając zmiennej łańcuchowej wartość początkową za pomocą wyrażenia `char NazwaZmiennej[długość] = "tekst";`;
- zmieniając zawartość zmiennej łańcuchowej za pomocą funkcji `scanf()` (wprowadzającej do zmiennej łańcuchowej tekst wpisywany przy użyciu klawiatury przez użytkownika);
- wyświetlając tekst przechowywany w zmiennej łańcuchowej za pomocą funkcji `printf()`.

Zmienne łańcuchowe możesz też traktować jak najzwyklejsze tablice elementów typu `char` i odczytywać oraz zapisywać ich pojedyncze elementy. Pamiętaj jedynie, że aby taka zmienna nadal mogła być traktowana jak zmienna łańcuchowa (przechowująca tekst), gdzieś wewnątrz przydzielonego jej obszaru pamięci musi znaleźć się znak o kodzie zero, oznaczający koniec tekstu. Znakowi takiemu odpowiada kod specjalny `\0`:

```
char KodZerowy = '\0';  // Przypisanie kodu zerowego zmiennej typu char.
```

Przykład programu dokonującego zmian w tekście (zapisanym w zmiennej łańcuchowej) za pomocą zwykłych operacji odwoływania się do pojedynczych elementów typu `char` tablicy przedstawiony został na listingu 5.3, a na rysunku 5.6 — efekt działania tego programu.

Listing 5.3. *Przykład traktowania zmiennej łańcuchowej jak zwykłej tablicy elementów typu char*

```
#include <stdio.h>

int main()
{

   char Napis[40] = "To jest napis 12345678901234567890";

   // Zamień dziewiąty znak z 'n' na 'Z'.
   Napis[8] = 'Z';

   // Obetnij napis po 20 znakach.
   Napis[20] = '\0';
```

```
    // Wyświetl napis.
    printf("%s\n", Napis);

    // Koniec.
    return 0;

}
```



Program przykładowy 03.cpp

Rysunek 5.6. *Efekt działania programu*

```
G:\Helion\cpp\samples\r05>g++ 03.cpp -o 03.exe

G:\Helion\cpp\samples\r05>03
To jest Zapis 1234567890123456

G:\Helion\cpp\samples\r05>_
```



Nic nie stoi na przeszkodzie, byś tablicę elementów typu `char` wykorzystywał wyłącznie jako zbiór znaków, a nie jako zmienną łańcuchową. W takim przypadku możesz przechowywać w poszczególnych elementach znaki o dowolnych kodach, pamiętaj jednak, by nigdy nie używać takiej tablicy jako parametru funkcji operujących na zmiennych łańcuchowych.



Więcej informacji na temat manipulowania tekstem zapisanym w zmiennych łańcuchowych znajdziesz w rozdziałach 7. i 9.