helion.pl

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

Programowanie w języku C. FAQ



Autor: Steve Summit
Tłumaczenie: Przemysław Kowalczyk
ISBN: 83-7361-094-4
Tytuł oryginału: C Programming FAQs
Format: B5, stron: 400

Przysłowie „kto pyta, nie błądzi” nie zawiera całej prawdy. Nie wystarczy pytać, trzeba jeszcze znajdować odpowiedzi. Książka „Programowanie w języku C. FAQ” to zbiór kilkuset odpowiedzi na najczęściej zadawane pytania na temat tego języka programowania. Z pewnością część z przedstawionych tu pytań już pojawiła się w Twojej praktyce programistycznej (pamiętasz, ile czasu straciłeś poszukując odpowiedzi?). Inne problemy dopiero się pojawią i jeśli na Twojej półce będzie ta książka, szybko znajdziesz w niej zwięzłe, ale wyczerpujące rozwiązanie często wzbogacone przykładem kodu źródłowego.

Chociaż książka żadną miarą nie powinna być traktowana jako podręcznik, z którego można nauczyć się programowania w C, z pewnością przyda się każdej osobie używającej tego języka w codziennej praktyce. Autor porusza wiele przydatnych zagadnień obejmujących szeroki zestaw tematów.

Omówiono między innymi:

- Deklaracje
- Struktury i unie
- Puste wskaźniki
- Wyrażenia
- Makroprocesor
- Alokację pamięci
- Różnice między standardami C
- Standardową bibliotekę wejścia-wyjścia
- Kwestie związane z systemami operacyjnymi

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

Spis treści

Pytania **9**

Przedmowa **37**

Wprowadzenie **41**
Jak korzystać z tej książki? 41
Format pytań 43
Przykłady kodu 43
Organizacja książki 44

Rozdział 1. Deklaracje i inicjalizacja **47**
Podstawowe typy 47
Deklaracje wskaźników 50
Styl deklaracji 51
Klasy pamięci 54
Definicje typów 55
Kwalifikator const 59
Złożone deklaracje 59
Rozmiary tablic 62
Problemy z deklaracjami 64
Przestrzeń nazw 65
Inicjalizacja 69

Rozdział 2. Struktury, unie i typy wyliczeniowe **73**
Deklaracje struktur 73
Działania na strukturach 78
Wyrównywanie pól struktur 80
Dostęp do pól struktur 82
Różne pytania na temat struktur 83
Unie 84
Typy wyliczeniowe 85
Pola bitowe 86

Rozdział 3. Wyrażenia **89**
Kolejność obliczania 89
Inne pytania na temat wyrażeń 96
Reguły zachowywania 100

Rozdział 4. Wskaźniki ... 103
Podstawy ... 103
Działania na wskaźnikach ... 105
Wskaźniki jako parametry funkcji ... 106
Różne zastosowania wskaźników ... 110

Rozdział 5. Wskaźniki puste ... 113
Wskaźniki puste i literały wskaźnika pustego ... 113
Makrodefinicja NULL ... 116
Retrospektywa ... 121
Co można znaleźć pod adresem 0? ... 124

Rozdział 6. Tablice i wskaźniki ... 127
Podstawowe związki między tablicami i wskaźnikami ... 128
Tablicom nie można przypisywać wartości ... 131
Retrospektywa ... 132
Wskaźniki do tablic ... 134
Dynamiczne tworzenie tablic ... 136
Funkcje a tablice wielowymiarowe ... 140
Rozmiary tablic ... 143

Rozdział 7. Przydzielanie pamięci ... 145
Podstawowe problemy z przydzielaniem pamięci ... 145
Wywoływanie funkcji malloc ... 149
Problemy z funkcją malloc ... 152
Zwalnianie pamięci ... 155
Rozmiar przydzielonych bloków ... 158
Inne funkcje przydzielające pamięć ... 159

Rozdział 8. Znaki i napisy ... 165

Rozdział 9. Wyrażenia i zmienne logiczne ... 171

Rozdział 10. Preprocesor języka C ... 175
Makrodefinicje ... 175
Pliki nagłówkowe ... 180
Kompilacja warunkowa ... 183
Zaawansowane przetwarzanie ... 186
Makrodefinicje ze zmienną liczbą argumentów ... 189

Rozdział 11. Standard ANSI/ISO języka C ... 193
Standard ... 193
Prototypy funkcji ... 195
Kwalifikator const ... 198
Funkcja main ... 200
Właściwości preprocesora ... 203
Inne sprawy związane ze Standardem ANSI ... 205
Stare lub niezgodne ze Standardem kompilatory ... 208
Kwestie zgodności ... 211

Rozdział 12. Standardowa biblioteka wejścia-wyjścia ... 215
Podstawy obsługi wejścia-wyjścia ... 216
Formaty dla funkcji printf ... 218
Formaty dla funkcji scanf ... 222
Problemy z funkcją scanf ... 224
Inne funkcje z biblioteki wejścia-wyjścia ... 228

Otwieranie plików i operacje na nich ... 232
Przekierowywanie strumieni stdin i stdout ... 235
Obsługa wejścia-wyjścia w trybie binarnym ... 237

Rozdział 13. Funkcje biblioteczne ... 241
Funkcje operujące na napisach ... 241
Sortowanie ... 247
Data i czas ... 251
Liczby losowe ... 254
Inne funkcje biblioteczne ... 261

Rozdział 14. Liczby zmiennoprzecinkowe ... 265

Rozdział 15. Listy argumentów o zmiennej długości ... 273
Wywoływanie funkcji o zmiennej liczbie argumentów ... 274
Implementacja funkcji o zmiennej liczbie argumentów ... 275
Pobieranie argumentów z listy ... 280
Trudniejsze problemy ... 283

Rozdział 16. Dziwne problemy ... 287

Rozdział 17. Styl ... 293

Rozdział 18. Narzędzia i zasoby ... 299
Narzędzia ... 299
Program lint ... 301
Zasoby ... 303

Rozdział 19. Kwestie zależne od systemu operacyjnego ... 309
Klawiatura i ekran ... 310
Inne operacje wejścia-wyjścia ... 316
Pliki i katalogi ... 318
Bezpośredni dostęp do pamięci ... 324
Polecenia systemowe ... 326
Środowisko procesu ... 329
Inne operacje zależne od systemu ... 330
Retrospektywa ... 333

Rozdział 20. Różności ... 335
Przydatne techniki ... 336
Bity i bajty ... 343
Wydajność ... 348
Instrukcja switch ... 352
Różne kwestie językowe ... 354
Inne języki ... 358
Algorytmy ... 359
Inne ... 364

Słownik ... 369

Bibliografia ... 379

Skorowidz ... 383

Rozdział 6.

Tablice i wskaźniki

Siła języka C wynika między innymi z ujednoliconego traktowania tablic i wskaźników. Bardzo łatwo jest za pomocą wskaźników operować na tablicach czy symulować tablice tworzone dynamicznie. Tak zwana odpowiedniość wskaźników i tablic jest tak duża, że niektórzy programiści zapominają o zasadniczych różnicach, myśląc, że są one identyczne, albo zakładając nieistniejące między nimi podobieństwa.

Podstawą „odpowiedniości" tablic i wskaźników w języku C jest fakt, że odwołania do tablic „degenerują się" do wskaźników do pierwszego elementu tablicy, co opisuje pytanie 6.3. Z tego powodu tablice są „obywatelami drugiej kategorii" w C — nigdy nie posługujesz się tablicami jako całymi obiektami (na przykład aby skopiować je albo przekazać do funkcji). Kiedy użyjesz nazwy tablicy, w wyrażeniu pojawi się wskaźnik zamiast całej tablicy. Nawet operator indeksowania tablic `[]` w rzeczywistości operuje na wskaźniku. Wyrażenie `a[i]` jest równoważne wyrażeniu wskaźnikowemu `*((a)+(i))`.

Duża część tego rozdziału (szczególnie pytania „retrospektywne" 6.8 – 6.10) może wydawać się powtarzaniem wciąż tych samych wiadomości. Wielu programistów ma jednak spore kłopoty ze zrozumieniem związków i różnic między wskaźnikami i tablicami, w tym rozdziale staram się wyjaśnić je najlepiej, jak tylko potrafię. Jeżeli nudzą Cię takie powtórki, możesz przeskoczyć do następnego rozdziału. Jeżeli jednak masz kłopoty z tablicami lub wskaźnikami, przeczytaj uważnie odpowiedzi, a poszczególne części układanki na pewno „wskoczą na swoje miejsca".

Podstawowe związki między tablicami i wskaźnikami

6.1

Pytanie: **W jednym z plików źródłowych mam definicję `char a[6]`, a w innym deklarację `extern char *a`. Dlaczego to nie działa?**

Odpowiedź: Zmienna zadeklarowana przy użyciu `extern char *a` nie jest typu tablicowego, nie pasuje więc do rzeczywistej definicji. Typ „wskaźnik do typu T" nie jest tym samym, co typ „tablica elementów typu T". Użyj deklaracji `extern char a[]`.

Referencje: ANSI §3.5.4.2
ISO §6.5.4.2
CT&P §3.3, §4.5

6.2

Pytanie: **Ale przecież słyszałem, że `char a[]` i `char *a` to to samo. Czy to prawda?**

Odpowiedź: Nie, to nie jest prawda (to, co słyszałeś, odnosiło się do parametrów formalnych funkcji, zobacz pytanie 6.4). Tablice nie są wskaźnikami, chociaż są blisko z nimi związane (zobacz pytanie 6.3) i korzysta się z nich podobnie (zobacz pytania 4.1, 6.8, 6.10 i 6.14).

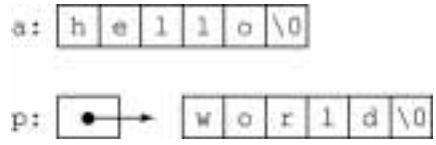
Deklaracja tablicy `char a[6]` powoduje przydzielenie miejsca na sześć znaków i powiązanie go z nazwą `a`. Innymi słowy, `a` stanowi adres obszaru pamięci, w którym zmieści się sześć znaków. Z drugiej strony, deklaracja wskaźnika `char *p` powoduje przydzielenie miejsca na wskaźnik i powiązanie go z nazwą `p`. Wskaźnik ten może wskazywać praktycznie gdziekolwiek: na pojedynczą zmienną typu `char`, na ciągłą tablicę elementów typu `char` albo nigdzie[1] (zobacz też pytania 1.30 i 5.1).

Jak zwykle w takich sytuacjach, obrazek wart jest tysiąca słów. Deklaracje:

```
char a[] = "hello";
char *p = "world";
```

powodują utworzenie i zainicjalizowanie struktur danych, które można zobrazować następująco:

[1] Nie należy jednak interpretować pojęć „gdziekolwiek" i „nigdzie" zbyt szeroko. Aby wskaźnik miał poprawną wartość, musi wskazywać na prawidłowo przydzielony obszar pamięci (zobacz pytania 7.1, 7.2 i 7.3). Aby wskazywać „nigdzie", musi być wskaźnikiem pustym (zobacz pytanie 5.1).



Należy pamiętać, że wyrażenie `x[3]` tłumaczone jest przez kompilator odmiennie, w zależności od tego, czy `x` to wskaźnik, czy tablica. W zasięgu widoczności powyższych deklaracji wyrażenie `a[3]` powoduje wygenerowanie następującego kodu: „weź adres tablicy `a`, dodaj do niego 3 i pobierz znak stamtąd". Z kolei wyrażenie `p[3]` tłumaczone jest jako: „weź adres wskaźnika `p`, pobierz jego wartość, dodaj do niej 3 i pobierz znak stamtąd". Innymi słowy, wyrażenie `a[3]` odnosi się do znaku odległego o 3 miejsca od początku obiektu o *nazwie* `a`, natomiast `p[3]` to znak odległy o 3 miejsca od obiektu *wskazywanego* przez `p`. W naszym przykładzie zarówno `a[3]`, jak i `p[3]` odnoszą się do litery `'l'`, ale nie są to te same litery i ich adresy obliczane są inaczej.

Referencje: K&R2 §5.5
CT&P §4.5

6.3

Pytanie: **Co w takim razie oznacza „odpowiedniość wskaźników i tablic" w C?**

Odpowiedź: Wiele nieporozumień dotyczących tablic i wskaźników w języku C bierze się właśnie z niewłaściwego rozumienia tego pojęcia. „Odpowiedniość" nie oznacza, że są one identyczne, ani nawet, że mogą być używane wymiennie. Definicję owej „odpowiedniości" można przedstawić następująco: referencja do obiektu o typie „tablica elementów typu T", „degeneruje się" w wyrażeniach (z trzema wyjątkami) do wskaźnika do pierwszego elementu tablicy. Jest on typu „wskaźnik do typu T" (Wspomniane wyjątki to: kiedy nazwa tablicy jest argumentem operatora `sizeof`, operatora `&` albo jest literałem napisowym w inicjalizatorze tablicy znaków[2]. Zobacz pytania 6.23, 6.12 i 1.32, odpowiednio).

Z takiej definicji wynika, że zastosowanie operatora `[]` do tablic i wskaźników nie różni się tak bardzo[3], mimo że są to odmienne obiekty. Jeżeli `a` to tablica, a `p` to wskaźnik, wyrażenie w postaci `a[i]` powoduje, że odwołanie do tablicy zamieniane jest niejawnie na wskaźnik do pierwszego elementu, zgodnie z powyższą regułą. Dalsze operacje są identyczne jak w przypadku indeksowania wskaźnika w wyrażeniu `p[i]` (chociaż obliczenie różni się operacjami odczytu pamięci, jak wyjaśniłem w pytaniu 6.2). Jeżeli przypiszesz do wskaźnika adres tablicy:

```
p = a;
```

to wyrażenia `p[3]` i `a[3]` będą się odnosić do tego samego elementu.

[2] Za „literał napisowy w inicjalizatorze tablicy znaków" uważamy także literały inicjalizujące tablice znaków `wchar_t`.

[3] Mówiąc ściśle, operator `[]` zawsze jest stosowany do wskaźników, zobacz pytanie 6.10, punkt 2.

Dzięki temu właśnie wskaźniki można tak łatwo stosować do operacji na tablicach, używać ich zamiast tablic jako argumentów funkcji (zobacz pytanie 6.4) czy symulować tablice dynamiczne (zobacz pytanie 6.14).

Zobacz też pytania 6.8 i 6.10.

Referencje: K&R1 §5.3
K&R2 §5.3
ANSI §3.2.2.1, §3.3.2.1, §3.3.6
ISO §6.2.2.1, §6.3.2.1, §6.3.6
H&S §5.4.1

6.4

Pytanie: **Jeżeli więc tablice i wskaźniki różnią się tak bardzo, dlaczego można używać ich wymiennie jako parametrów formalnych funkcji?**

Odpowiedź: Dla wygody.

Ponieważ w wyrażeniach tablice stają się wskaźnikami, do funkcji przekazywane są zawsze wskaźniki, a nigdy tablice. Możesz „udawać", że funkcja oczekuje jako parametru tablicy i podkreślić to w kodzie źródłowym, deklarując funkcję jako:

```
f(char a[])
{ … }
```

Jednak taka deklaracja, zinterpretowana dosłownie, nie ma zastosowania, gdyż funkcja i tak otrzyma wskaźnik, więc dla kompilatora równoważna jest z deklaracją:

```
f(char *a)
{ … }
```

Nie ma nic niewłaściwego w mówieniu, że funkcja otrzymuje jako parametr tablicę, jeżeli jest on wewnątrz funkcji traktowany jako tablica.

Wymienne stosowanie tablic i wskaźników dopuszczalne jest jedynie w deklaracjach parametrów formalnych funkcji i w żadnym innym przypadku. Jeżeli podmiana tablicy na wskaźnik w deklaracji funkcji przeszkadza Ci, deklaruj parametry jako wskaźniki. Wiele osób uważa, że zamieszanie, jakie powodują takie deklaracje, znacznie przeważa nad korzyściami z faktu, że parametr „wygląda", jakby był tablicą (Zauważ również, że taka konwersja zachodzi tylko raz; parametr typu `char a2[][]` jest nieprawidłowy. Zobacz pytania 6.18 i 6.19).

Zobacz też pytanie 6.21.

Referencje: K&R1 §5.3, §A10.1
K&R2 §5.3, §A8.6.3, §A10.1
ANSI §3.5.4.3, §3.7.1, §3.9.6
ISO §6.5.4.3, §6.7.1, §6.9.6
H&S §9.3
CT&P §3.3
Ritchie, *The Development of the C Language*

Tablicom nie można przypisywać wartości

Jeżeli tablica pojawia się po prawej stronie operatora przypisania, kopiowany jest wskaźnik do pierwszego elementu, a nie cała tablica. Co więcej, tablica nie może się pojawić po lewej stronie operatora przypisania (między innymi dlatego, że i tak nie można jej przypisać całej tablicy, jak wynika z poprzedniego zdania).

6.5

Pytanie: **Czemu tablicom nie można przypisywać wartości? Poniższy kod nie działa:**

```
extern char *getpass();
char str[10];
str = getpass("Wprowadź hasło: ");
```

Odpowiedź: Tablice są „obywatelami drugiej kategorii" w języku C. Nie można im, między innymi, przypisywać wartości (zobacz też pytanie 6.7). Kiedy chcesz skopiować elementy z jednej tablicy do drugiej, musisz to wyrazić jawnie. W przypadku napisów, pamiętanych jako `char[]`, odpowiednia jest najczęściej funkcja `strcpy`:

```
strcpy(str, getpass("Wprowadź hasło: "));
```

Jeżeli chcesz operować na tablicach bez kopiowania ich zawartości, możesz używać wskaźników — wtedy wystarczy zwykłe przypisanie. Zobacz też pytania 4.1 i 8.2.

Referencje: ANSI §3.2.2.1
ISO §6.2.2.1
H&S §7.9.1

6.6

Pytanie: **Jeżeli nie mogę przypisywać wartości tablicom, w jaki sposób działa ten kod?**

```
int f(char str[])
{
      if(str[0] == '\0')
            str = "pusty";
      …
}
```

Odpowiedź: W tym przypadku `str` jest parametrem funkcji. Jego deklaracja jest „podmieniana" przez kompilator, jak opisałem to w pytaniu 6.4. Innymi słowy, `str` *jest* wskaźnikiem (typu `char *`), można więc mu przypisać wartość.

6.7

Pytanie: **W jaki sposób tablica może być l-wartością, jeżeli nie można do niej przypisać?**

Odpowiedź: Termin *l-wartość* nie oznacza w rzeczywistości „czegoś, czemu można przypisać wartość". Lepszą definicją byłoby „coś, co posiada adres w pamięci"[4]. Standard ANSI/ISO języka C definiuje również pojęcie „modyfikowalnej l-wartości". Tablica nią nie jest. Zobacz też pytanie 6.5.

Referencje: ANSI §3.2.2.1
ISO §6.2.2.1
Rationale §3.2.2.1
H&S §7.1

Retrospektywa

Ponieważ związki między tablicami i wskaźnikami powodują tyle nieporozumień, kilka następnych pytań dotyczy właśnie przyczyn tych nieporozumień.

6.8

Pytanie: **Jaka jest praktyczna różnica między tablicami i wskaźnikami?**

Odpowiedź: Tablica jest pojedynczym, przydzielonym wcześniej ciągłym obszarem pamięci, zawierającym elementy tego samego typu. Posiada stały rozmiar i położenie. Wskaźnik to odniesienie do dowolnego elementu (określonego typu) gdziekolwiek. Wskaźnikowi należy przypisać adres przydzielonego w jakiś sposób obszaru pamięci, ale można jego wartość modyfikować (a obszarowi pamięci, jeżeli został przydzielony dynamicznie, można zmienić rozmiar). Wskaźnik może wskazywać na elementy tablicy i można go użyć (wraz z funkcją `malloc`) do symulowania dynamicznych tablic. Wskaźniki są jednak znacznie bardziej ogólną strukturą danych (zobacz również pytanie 4.1).

Z powodu tak zwanej „odpowiedniości wskaźników i tablic" (zobacz pytanie 6.3) może się wydawać, że tablic i wskaźników można używać wymiennie. Wskaźnik do obszaru pamięci przydzielonego przez funkcję `malloc` jest często traktowany jako tablica (może być nawet argumentem operatora `[]`). Zobacz pytania 6.14 i 6.16 (Pamiętaj o zachowaniu ostrożności przy użyciu operatora `sizeof`, zobacz pytanie 7.28).

Zobacz też pytania 1.32, 6.10 i 20.14.

[4] Pierwotna definicja l-wartości rzeczywiście mówiła o lewej stronie operatora przypisania.

6.9

Pytanie: **Ktoś wyjaśnił mi, że tablice to w rzeczywistości stałe wskaźniki. Czy to prawda?**

Odpowiedź: To zbyt uproszczone wyjaśnienie. Nazwa tablicy jest „stałą" w tym sensie, że nie można jej przypisać wartości. Jednak tablica to *nie* wskaźnik, co powinna wyjaśnić odpowiedź na pytanie 6.2. Zobacz też pytania 6.3, 6.8 i 6.10.

6.10

Pytanie: **Ciągle nie do końca rozumiem. Czy wskaźnik jest rodzajem tablicy, czy może tablica jest rodzajem wskaźnika?**

Odpowiedź: Tablica *nie jest* wskaźnikiem, a wskaźnik *nie jest* tablicą. *Referencja* do tablicy (czyli użycie nazwy tablicy w kontekście wyrażenia) jest *zamieniana* na wskaźnik (zobacz pytania 6.2 i 6.3).

Są przynajmniej trzy prawidłowe interpretacje tej sytuacji:

1. Wskaźniki mogą symulować tablice (ale to nie jedyne ich zastosowanie, zobacz pytanie 4.1).
2. W języku C w zasadzie nie ma tablic z prawdziwego zdarzenia (są „obywatelami drugiej kategorii"). Nawet operator `[]` jest w rzeczywistości operatorem działającym na wskaźniku.
3. Na wyższym poziomie abstrakcji wskaźnik do ciągłego obszaru pamięci może być uważany za tablicę (chociaż są też inne zastosowania wskaźników).

Z drugiej strony, *nie należy* myśleć w ten sposób:

4. „Są dokładnie takie same" (nieprawda, zobacz pytanie 6.2).
5. „Tablice to stałe wskaźniki" (nieprawda, zobacz pytanie 6.9).

Zobacz też pytanie 6.8.

6.11

Pytanie: **Spotkałem się z „dowcipnym" kodem, zawierającym wyrażenie `5["abcdef"]`. Dlaczego jest ono poprawne?**

Odpowiedź: Może to zabrzmi niewiarygodnie, ale indeksowanie tablic jest działaniem przemiennym[5] w języku C. Ten ciekawy fakt wynika ze wskaźnikowej definicji

[5] Przemienność dotyczy tylko argumentów operatora `[]`. Wyrażenie `a[i][j]` jest w oczywisty sposób różne od wyrażenia `a[j][i]`.

operatora `[]`. Wyrażenie `a[e]` jest równoważne `*((a)+(e))` dla *dowolnych* dwóch wyrażeń `a` i `e`, jeżeli tylko jedno z nich jest wyrażeniem wskaźnikowym, a drugie całkowitym. „Dowód" przemienności mógłby wyglądać tak:

- `a[e]` jest z definicji równoważne:
- `*((a) + (e))` jest równoważne na mocy przemienności dodawania:
- `*((e) + (a))` jest z definicji równoważne:
- `e[a]`.

Nieoczekiwana przemienność operatora `[]` traktowana jest zazwyczaj w tekstach o języku C jako powód do dumy, chociaż nie ma sensownych zastosowań poza Konkursami Zaciemnionego Kodu w C (zobacz pytanie 20.36).

Ponieważ napisy w języku C są tablicami elementów typu `char`, wyrażenie `"abcdef"[5]` jest całkowicie poprawne. Jego wartością jest litera `'f'`. Możesz uważać to za skróconą postać wyrażenia:

```
char *tmpptr = "abcdef";

... tmpptr[5] ...
```

W pytaniu 20.10 znajdziesz bardziej realistyczny przykład.

Referencje: Rationale §3.3.2.1
H&S §5.4.1, §7.4.1

Wskaźniki do tablic

Ponieważ tablice zwykle zamieniane są na wskaźniki, szczególnie łatwo o nieporozumienia, kiedy operujemy na wskaźnikach do całych tablic (zamiast, jak zwykle, do ich pierwszych elementów).

6.12

Pytanie: **Jeżeli odwołania do tablic przekształcane są na wskaźniki, jaka jest różnica między `array` i `&array` (przy założeniu, że `array` to jakaś tablica)?**

Odpowiedź: Wyrażenia te różnią się typem.

Standard języka C stanowi, że wyrażenie `&array` jest typu „wskaźnik do tablicy elementów typu T" i zwraca wskaźnik do całej tablicy (wcześniejsze kompilatory generalnie ignorowały operator `&` w takim kontekście, czasem tylko zgłaszając ostrzeżenie). We wszystkich kompilatorach języka C odwołanie do nazwy tablicy (bez operatora `&`) zwraca wskaźnik, typu „wskaźnik do typu T", do pierwszego elementu tablicy.

W przypadku tablicy jednowymiarowej, jak na przykład:

```
int a[10];
```

odwołanie do `a` jest typu „wskaźnik do typu `int`", a `&a` — „wskaźnik do tablicy 10 elementów typu `int`". W przypadku tablic dwuwymiarowych:

```
int array[NROWS][NCOLUMNS];
```

odwołanie do `array` jest typu „wskaźnik do tablicy `NCOLUMNS` elementów typu `int`", natomiast `&array` — „wskaźnik do tablicy `NROWS` tablic o `NCOLUMNS` elementów typu `int`".

Zobacz też pytania 6.3, 6.13 i 6.18.

Referencje: ANSI §3.2.2.1, §3.3.3.2
ISO §6.2.2.1, §6.3.3.2
Rationale §3.3.3.2
H&S §7.5.6

6.13

Pytanie: **Jak zadeklarować wskaźnik do tablicy?**

Odpowiedź: Zastanów się, czy rzeczywiście go potrzebujesz. Kiedy ktoś mówi o wskaźniku do tablicy, ma zazwyczaj na myśli wskaźnik do jej pierwszego elementu.

Zamiast wskaźnika do tablicy czasem lepiej użyć wskaźnika do jednego z elementów tablicy. Tablice elementów typu `T` stają się w wyrażeniach wskaźnikami do typu `T` (zobacz pytanie 6.3), co jest bardzo wygodne. Indeksowanie albo zwiększanie powstałego tak wskaźnika pozwala na dostęp do elementów tablicy. Rzeczywiste wskaźniki do tablic, kiedy są indeksowane lub zwiększane, „przechodzą" przez całe tablice i są przydatne tylko, gdy operujemy na tablicach tablic[6] (zobacz pytanie 6.18).

Jeżeli naprawdę potrzebujesz wskaźnika do całej tablicy, zadeklaruj go na przykład tak: `int (*ap)[N]`, gdzie `N` to rozmiar tablicy (zobacz też pytanie 1.21). Jeżeli rozmiar tablicy jest nieznany, `N` można teoretycznie pominąć, ale zadeklarowany w ten sposób „wskaźnik do tablicy nieznanego rozmiaru" jest bezużyteczny.

Poniższe przykłady pokazują różnice między zwykłymi wskaźnikami a wskaźnikami do tablic. Przy założeniu, że obowiązują deklaracje:

```
int a1[3] = {0, 1, 2};
int a2[2][3] = {{3, 4, 5}, {6, 7, 8}};
int *ip;                /* wskaźnik do typu int */
int (*ap)[3];           /* wskaźnik do tablicy 3 elementów typu int */
```

możemy użyć wskaźnika do typu `int`, `ip`, aby operować na jednowymiarowej tablicy `a1`:

[6] Rozumowanie to dotyczy oczywiście również tablic trój- i więcejwymiarowych.

```
ip = a1;
printf("%d ", *ip);
ip++;
printf("%d\n", *ip);
```

Ten fragment kodu wydrukuje:

```
0 1
```

Jednak próba użycia wskaźnika do tablicy, `ap`, na `a1`:

```
ap = &a1;
printf("%d ", **ap);
ap++;                       /* ŹLE */
printf("%d\n", **ap);       /* zachowanie niezdefiniowane */
```

spowodowałaby wypisanie 0 i zachowanie niezdefiniowane (od wypisania przypadkowej wartości do błędu w czasie wykonania) w momencie drugiego wywołania funkcji `printf`. Wskaźnik do tablicy może się przydać, kiedy operujemy na tablicy tablic, na przykład `a2`:

```
ap = a2;
printf("%d %d\n", (*ap)[0], (*ap)[1]);
ap++;                   /* przechodzimy do następnej (pod)tablicy */
printf("%d %d\n", (*ap)[0], (*ap)[1]);
```

Ten fragment kodu drukuje:

```
3 4
6 7
```

Zobacz też pytanie 6.12.

Referencje: ANSI §3.2.2.1
ISO §6.2.2.1

Dynamiczne tworzenie tablic

Bliski związek tablic i wskaźników ułatwia symulowanie tablic o rozmiarze określonym w czasie działania programu za pomocą wskaźników do dynamicznie przydzielonych obszarów pamięci.

6.14

Pytanie: **Jak określić rozmiar tablicy w czasie działania programu? Jak uniknąć tablic o z góry ustalonym rozmiarze?**

Odpowiedź: Odpowiedniość tablic i wskaźników (zobacz pytanie 6.3) pozwala „symulować" tablice dynamiczne za pomocą wskaźnika do obszaru pamięci, dostarczonego przez funkcję `malloc`. Po wykonaniu:

```
#include <stdlib.h>
int *dynarray = (int *)malloc(10 * sizeof(int));
```

(i jeżeli wywołanie funkcji `malloc` się powiedzie) możesz odwoływać się do `dynarray[i]` (dla `i` od 0 do 9) tak, jakby `dynarray` była zwykłą, statycznie utworzoną tablicą (`int a[10]`). Zobacz również pytania 6.16, 7.28 i 7.29.

6.15

Pytanie: **Jak zadeklarować lokalną tablicę o rozmiarze równym tablicy przekazanej jako parametr?**

Odpowiedź: W języku C nie można tego zrobić. Rozmiary tablic muszą być znane w czasie kompilacji (Kompilator GNU C dopuszcza możliwość deklarowania tablic o zmiennym rozmiarze jako rozszerzenie; znalazło się ono również w standardzie C99). Możesz użyć funkcji `malloc`, aby stworzyć „tablicę dynamiczną", ale pamiętaj o zwolnieniu jej funkcją `free`. Zobacz też pytania 6.14, 6.16, 6.19, 7.22 i może również 7.32.

Referencje: ANSI §3.4, §3.5.4.2
ISO §6.4, §6.5.4.2

6.16

Pytanie: **Jak dynamicznie przydzielić pamięć dla tablicy wielowymiarowej?**

Odpowiedź: W większości przypadków najlepiej stworzyć tablicę[7] wskaźników do wskaźników, a następnie każdemu ze wskaźników przypisać adres dynamicznie przydzielonego „wiersza". Oto przykład dla tablicy dwuwymiarowej:

```
#include <stdlib.h>

int **array1 = (int **)malloc(nrows * sizeof(int *));
for(i = 0; i < nrows; i++)
      array1[i] = (int *)malloc(ncolumns * sizeof(int));
```

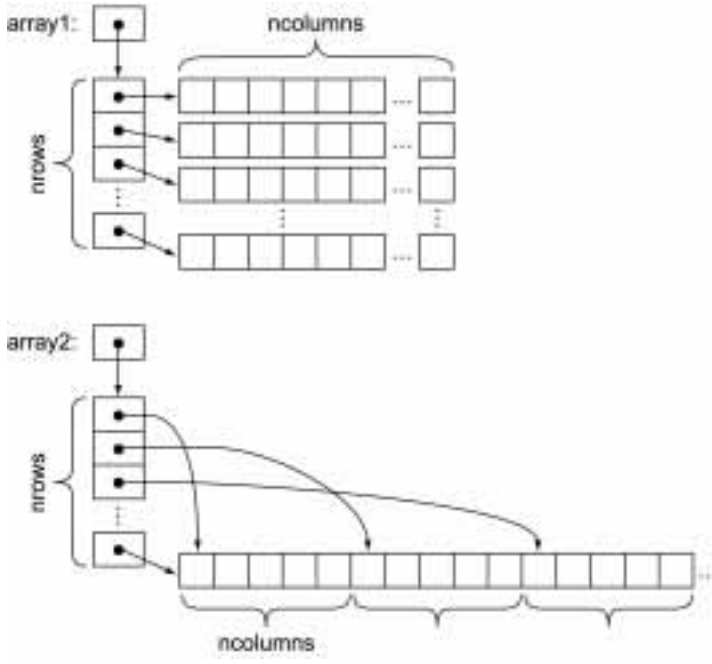
W rzeczywistym kodzie należałoby oczywiście sprawdzić wszystkie wartości zwrócone przez funkcję `malloc`.

Jeżeli nie zamierzasz zmieniać długości wierszy, a za to chciałbyś, aby przydzielony tablicy obszar pamięci był ciągły, wystarczy odrobina arytmetyki na wskaźnikach:

```
int **array2 = (int **)malloc(nrows * sizeof(int *));
array2[0] = (int *)malloc(nrows * ncolumns * sizeof(int));
for(i = 1; i < nrows; i++)
      array2[i] = array2[0] + i * ncolumns;
```

[7] Mówiąc ściśle, nie są to tablice, ale raczej obiekty *używane* jak tablice, zobacz pytanie 6.14.

W obu przypadkach (to znaczy tablic `array1` i `array2`) tablice dynamiczne można indeksować za pomocą normalnych operatorów: `arrayx[i][j]` (dla `0 ≤ i < NROWS` i `0 ≤ j < NCOLUMNS`). Poniższy rysunek przedstawia schematycznie „układ" wierszy w tablicach `array1` i `array2`.



Jeżeli dwa odwołania do pamięci w tym schemacie są z jakichś powodów nie do przyjęcia[8], możesz zasymulować tablicę dwuwymiarową dynamicznie stworzoną tablicą jednowymiarową:

```
int *array3 = (int *)malloc(nrows * ncolumns * sizeof(int));
```

Jednak w takim przypadku obliczanie indeksów musisz wykonywać własnoręcznie. Aby odwołać się do elementu o współrzędnych `i,j`, należałoby użyć wyrażenia `array3[i * ncolumns + j]`[9]. Takiej tablicy nie można jednak przekazać do funkcji, która akceptuje tablicę wielowymiarową. Zobacz też pytanie 6.19.

Możesz też użyć wskaźników do tablic:

```
int (*array4)[NCOLUMNS] =
      (int (*)[NCOLUMNS])malloc(nrows * sizeof(*array4));
```

[8] Zauważ jednak, że dwukrotne odwołanie do tablicy nie musi być wcale mniej efektywne niż jawne mnożenie indeksów.

[9] Jawne obliczanie indeksu można ukryć w makrodefinicji, na przykład: `#define Arrayacces(a, i, j) ((a)[(i) * ncolumns + (j)])`. Jednak wywołanie takiej makrodefinicji, z nawiasami i przecinkami, nie kojarzyłoby się z normalną składnią dostępu do tablic. Makrodefinicja musiałaby mieć również dostęp do jednego z wymiarów tablicy.

albo nawet:

```
int (*array5)[NROWS][NCOLUMNS]) =
      (int (*)[NROWS][NCOLUMNS])malloc(sizeof(*array5));
```

Jednak składnia odwołania do elementów wskazywanych tablic robi się coraz bardziej skomplikowana (w przypadku `array5` trzeba pisać `(*array5)[i][j]`). Najwyżej jeden wymiar tablicy można określić w czasie działania programu.

Używając tych technik, nie można oczywiście zapomnieć o zwolnieniu przydzielonej pamięci, kiedy nie jest już potrzebna. W przypadku tablic `array1` i `array2` wymaga to kilku kroków (zobacz też pytanie 7.23):

```
for(i = 0; i < nrows; i++)
      free((void *)array1[i]);
free((void *)array1);
free((void *)array2[0]);
free((void *)array2);
```

Nie możesz też mieszać tablic przydzielonych dynamicznie ze zwykłymi, utworzonymi statycznie (zobacz pytanie 6.20, a także 6.18).

Powyższe techniki można oczywiście rozszerzyć na trzy i więcej wymiarów. Oto „trójwymiarowa" wersja pierwszego sposobu:

```
int ***a3d = (int ***)malloc(xdim * sizeof(int **));
for(i = 0; i < xdim; i++) {
      a3d[i] = (int **)malloc(ydim * sizeof(int *));
      for(j = 0; j < ydim; j++)
            a3d[i][j] = (int *)malloc(zdim * sizeof(int));
}
```

Zobacz też pytanie 20.2.

6.17

Pytanie: **Wpadłem na fajny pomysł — jeżeli napiszę:**

```
int realarray[10];
int *array = &realarray[-1];
```

mogę traktować tablicę `array` tak, jakby jej indeksy zaczynały się od 1. Czy to poprawne?

Odpowiedź: Chociaż taka technika może wydawać się atrakcyjna (była nawet używana w starszych wydaniach książki *Numerical Recipes in C*), nie jest zgodna ze Standardem języka C. Arytmetyka wskaźników zdefiniowana jest tylko, jeżeli wartość wskaźnika pozostaje w obrębie tego samego przydzielonego bloku pamięci albo umownego „końcowego" elementu tuż za nim. W przeciwnym wypadku zachowanie programu jest niezdefiniowane, *nawet jeżeli nie następuje dereferencja wskaźnika*. Kod w pytaniu oblicza wskaźnik do obszaru przed początkiem tablicy `realarray`.

W momencie odejmowania offsetu może wystąpić błąd wygenerowania nieprawidłowego adresu (na przykład gdyby obliczenie adresu spowodowało „zawinięcie" wokół początku segmentu pamięci).

Referencje: K&R2 §5.3, §5.4, §A7.7
ANSI §3.3.6
ISO §6.3.6
Rationale §3.2.2.3

Funkcje a tablice wielowymiarowe

Trudno jest przekazywać tablice wielowymiarowe do funkcji z zachowaniem pełnej ogólności. Podmiana parametrów tablicowych na wskaźniki (zobacz pytanie 6.4) oznacza, że funkcja, która akceptuje zwykłe tablice, może również przyjmować tablice o dowolnej długości, co jest wygodne. Jednak podmiana dotyczy tylko najbardziej „zewnętrznej" tablicy, pozostałe wymiary nie mogą być zmienne. Problem ten wynika z faktu, że w języku C wymiary tablic muszą być zawsze znane w czasie kompilacji. Nie można ich określić, przekazując na przykład dodatkowy parametr.

6.18

Pytanie: **Mój kompilator zgłasza błędy, kiedy przekazuję tablicę dwuwymiarową do funkcji przyjmującej wskaźnik do wskaźnika. Dlaczego?**

Odpowiedź: Zasada (zobacz pytanie 6.3), na mocy której tablice „degenerują" się do wskaźników, nie działa rekurencyjnie. Tablica dwuwymiarowa (czyli w języku C tablica tablic) staje się wskaźnikiem do tablicy, nie wskaźnikiem do wskaźnika. Wskaźniki do tablic powodują dużo nieporozumień, dlatego należy ich używać ostrożnie. Zobacz pytanie 6.13 (nieporozumienia wzmaga fakt, że istnieją kompilatory, w tym stare wersje `pcc` i oparte na nich wersje programu `lint`, które niepoprawnie pozwalają przypisywać wielowymiarowe tablice do wskaźników na wskaźniki).

Jeżeli przekazujesz dwuwymiarową tablicę do funkcji:

```
int array[NROWS][NCOLUMNS];
f(array);
```

jej deklaracja musi mieć postać:

```
f(int a[][NCOLUMNS])
{ … }
```

albo:

```
f(int (*ap)[NCOLUMNS])   /* ap to wskaźnik na tablicę */
{ … }
```

W przypadku pierwszej deklaracji kompilator wykonuje podmianę typu parametru z „tablicy tablic" na „wskaźnik do tablicy" (zobacz pytania 6.3 i 6.4). W drugiej deklaracji jawnie określamy typ jako wskaźnik do tablicy. Ponieważ wywoływana funkcja nie przydziela pamięci dla tablicy, nie musi znać jej obu wymiarów — liczba wierszy, NROWS, może zostać pominięta w deklaracji. „Kształt" tablicy jest jednak wciąż ważny, więc wymiar NCOLUMNS (i wszystkie kolejne w przypadku tablic wielowymiarowych) musi zostać wyspecyfikowany.

Jeżeli deklaracja funkcji wskazuje, że oczekuje ona wskaźnika do wskaźnika, najprawdopodobniej nie można przekazać jej tablicy dwuwymiarowej bezpośrednio. Może być wtedy potrzebny pomocniczy wskaźnik:

```
extern g(int **ipp);

int *ip = &array[0][0];
g(&ip);          /* BYĆ MOŻE ŹLE */
```

Taki sposób wywołania jest jednak mylący i prawie na pewno nieprawidłowy. Tablica została „spłaszczona" — straciliśmy informację o jej szerokości.

Zobacz też pytania 6.12 i 6.15.

Referencje: K&R1 §5.10
K&R2 §5.9
H&S §5.4.3

6.19

Pytanie: **Jak tworzyć funkcje, które przyjmują tablice dwuwymiarowe, jeżeli nie znam ich „szerokości" w czasie kompilacji?**

Odpowiedź: To nie jest łatwe. Jednym ze sposobów jest przekazanie wskaźnika do elementu o indeksach [0][0] wraz z obydwoma wymiarami i „ręczne" obliczanie indeksów:

```
f2(int *aryp, int nrows, int ncolumns)
{ … aryp[i * ncolumns + j] … /* odwołanie do elementu array[i][j] */ }
```

Aby jawnie obliczyć indeks elementu, potrzebujemy wartości ncolumns („szerokości" każdego wiersza), nie nrows (*liczby* wierszy). Łatwo się tu pomylić.

Funkcji tej można przekazać tablicę array z pytania 6.18 w następujący sposób:

```
f2(&array[0][0], NROWS, NCOLUMNS);
```

Trzeba jednak zauważyć, że program, w którym indeksy w tablicy wielowymiarowej obliczane są „ręcznie", nie jest ściśle zgodny ze Standardem ANSI języka C. Zgodnie z oficjalną interpretacją, dostęp do elementu (&array[0][0])[x] jest niezdefiniowany, jeżeli x >= NCOLUMNS.

Kompilator GNU C pozwala definiować lokalne tablice o rozmiarach ustalonych przez argumenty funkcji, możliwość ta pojawiła się też w Standardzie C99.

Jeżeli chcesz stworzyć funkcję, która przyjmować będzie tablice wielowymiarowe o różnych rozmiarach, jednym z rozwiązań jest dynamiczne symulowanie takich tablic, jak w pytaniu 6.16.

Zobacz też pytania 6.15, 6.18 i 6.20.

Referencje: ANSI §3.3.6
ISO §6.3.6

6.20

Pytanie: **Jak korzystać jednocześnie z wielowymiarowych tablic przydzielanych statycznie i dynamicznie przy przekazywaniu ich do funkcji?**

Odpowiedź: Nie ma jednej, uniwersalnej metody. Mając deklaracje:

```
int array[NROWS][NCOLUMNS];
int **array1;      /* każdy wiersz osobno */
int **array2;      /* ciągły obszar pamięci */
int *array3;       /* tablica "spłaszczona" */
int (*array4)[NCOLUMNS];
int (*array5)[NROWS][NCOLUMNS];
```

gdzie wskaźniki są zainicjalizowane tak, jak w pytaniu 6.16, i funkcje, zadeklarowane jako:

```
f1(int a[][NCOLUMNS], int nrows, int ncolumns);
f2(int *aryp, int nrows, int ncolumns);
f3(int **pp, int nrows, int ncolumns);
```

gdzie funkcja `f1` akceptuje „zwykłą" tablicę dwuwymiarową, funkcja `f2` — „spłaszczoną" tablicę dwuwymiarową, a funkcja `f3` — tablicę symulowaną przez wskaźnik do wskaźnika (zobacz pytania 6.18 i 6.19), poniższe wywołania funkcji działają zgodnie z oczekiwaniami:

```
f1(array, NROWS, NCOLUMNS);
f1(array4, nrows, NCOLUMNS);
f1(*array5, NROWS, NCOLUMNS);
f2(&array[0][0], NROWS, NCOLUMNS);
f2(*array, NROWS, NCOLUMNS);
f2(*array2, nrows, ncolumns);
f2(array3, nrows, ncolumns);
f2(*array4, nrows, NCOLUMNS);
f2(**array5, NROWS, NCOLUMNS);
f3(array1, nrows, ncolumns);
f3(array2, nrows, ncolumns);
```

Poniższe dwa wywołania powinny działać prawidłowo na większości systemów, ale wymagają „podejrzanych" rzutowań i działają tylko, jeżeli dynamiczna wartość `ncolumns` równa jest statycznej `NCOLUMNS`:

```
f1((int (*)[NCOLUMNS])(*array2), nrows, ncolumns);
f1((int (*)[NCOLUMNS])array3, nrows, ncolumns);
```

Z wyżej wymienionych tylko funkcja `f2` może przyjmować tablice statyczne i przydzielone dynamicznie, chociaż *nie* będzie działać dla tradycyjnej implementacji `array1`, czyli „każdy wiersz osobno". Pamiętaj jednak, że przekazywanie `&array[0][0]` (albo `*array`) do funkcji `f2` nie jest ściśle zgodne ze Standardem — zobacz pytanie 6.19.

Jeżeli rozumiesz, dlaczego wszystkie wyżej wymienione sposoby przekazywania tablic wielowymiarowych do funkcji działają oraz dlaczego kombinacje, których tu nie przedstawiono, nie działają, możesz uznać, że *bardzo* dobrze rozumiesz tablice i wskaźniki w języku C.

Zamiast jednak zaprzątać sobie głowę tymi wszystkimi trudnymi regułami, możesz zastosować dużo prostsze rozwiązanie: *wszystkie* tablice wielowymiarowe twórz dynamicznie, jak w pytaniu 6.16. Jeżeli nie będzie statycznych tablic wielowymiarowych — wszystkie będą przydzielane jak `array1` lub `array2` w pytaniu 6.16 — wtedy wszystkie funkcje mogą mieć deklaracje podobne do `f3`.

Rozmiary tablic

Operator `sizeof` zwraca rozmiar tablicy, ale tylko wtedy, gdy jest on znany, a odniesienie do tablicy nie zredukowało się do wskaźnika.

6.21

Pytanie: **Dlaczego operator `sizeof` nie zwraca poprawnego rozmiaru tablicy, która jest parametrem funkcji? Moja testowa funkcja wypisuje 4, zamiast 10:**

```
f(char a[10])
{
      int i = sizeof(a);
      printf("%d\n", i);
}
```

Odpowiedź: Kompilator podmienia typ parametru z tablicy na wskaźnik (w tym przypadku `char *`, zobacz pytanie 6.4). Operator `sizeof` zwraca w tym wypadku rozmiar wskaźnika. Zobacz też pytania 1.24 i 7.28.

Referencje: H&S §7.5.2

6.22

Pytanie: **Jak kod w pliku, w którym tablica zadeklarowana jest jako `extern` (jest zdefiniowana i jej rozmiar jest określony w innym pliku), może określić jej wielkość? Operator `sizeof` nie działa na niej.**

Odpowiedź: Zobacz pytanie 1.24.

6.23

Pytanie: **Jak mogę określić liczbę elementów tablicy, jeżeli operator `sizeof` zwraca rozmiar w bajtach?**

Odpowiedź: Po prostu podziel rozmiar całej tablicy przez rozmiar jednego elementu:

```
int array[] = {1, 2, 3};
int narray = sizeof(array) / sizeof(array[0]);
```

Referencje: ANSI §3.3.3.4
ISO §6.3.3.4