KRZYSZTOF JASSEM ■ ANDRZEJ ZIEMKIEWICZ

SZTUKA **DOBREGO** PROGRAMOWANIA

PWN

KRZYSZTOF JASSEM ■ ANDRZEJ ZIEMKIEWICZ

SZTUKA **DOBREGO** PROGRAMOWANIA

PWN

Projekt okładki **Hubert Zacharski**

Ilustracja na okładce **shutterstock/polygraphus**

Wydawca **Łukasz Łopuszański**

Redaktor prowadzący **Iwona Lewandowska**

Redaktor **Irena Puchalska**

Koordynator produkcji **Anna Bączkowska**

Skład i łamanie **Dariusz Ziach**

ISBN 978-83-01-19029-3

Wydanie I
Warszawa 2016

Wydawnictwo Naukowe PWN SA
02-460 Warszawa, ul. Gottlieba Daimlera 2
tel. 22 69 54 321, faks 22 69 54 288
infolinia 801 33 33 88
e-mail: pwn@pwn.com.pl
www.pwn.pl

Druk i oprawa: OSDW Azymut Sp. z o.o.

Spis treści

Wstęp 13

Przygotowanie środowiska pracy 14
Instalacja systemu Linux 15
Pobranie OpenSUSE 15
Przygotowanie instalacji 15
Instalacja 15
Uruchamianie programów 16
Edycja kodu źródłowego 16
Przejście do trybu tekstowego 16
Skompilowanie programu 17
Uruchomienie programu 17
Polecane źródła w Internecie 17

Część I

Lekcja 1. Optymalizowanie funkcji 21

Elementy matematyki 21
Podstawy języka C 22
Program, instrukcja 22
Komentarze 23
Identyfikatory 23
Słowa kluczowe 23
Białe znaki 23
Podstawowe typy numeryczne 24
Zmienne i operator przypisania 25
Operatory 26

Instrukcja warunkowa 27
Instrukcje pętli 28
Pętla while 28
Pętla do while 29
Pętla for 29
Funkcje 30
Funkcje rekurencyjne 31
Rozwiązanie zadania 32
Krok 1. Pierwsze zadanie pomocnicze 32
Krok 2. Rachunki matematyczne 32
Krok 3. Drugie zadanie pomocnicze 33
Krok 4. Uogólnienie rachunków 34
Krok 5. Pierwsza wersja kodu funkcji rozwiązującej zadanie 34
Krok 6. Dodanie warunku zakończenia rekurencji 35
Krok 7. Optymalizacja liczby operacji 35
Krok 8. Optymalizacja liczby zmiennych 36
Krok 9. Usunięcie rekurencji 36
Krok 10. Sprawdzenie poprawności argumentów 38
Wnioski 39
Polecane źródła w Internecie 39

Lekcja 2. Działania na bitach 41

Sito Eratostenesa 41
Podstawy Języka C 41
Operacje na bitach 41
Operacja and 42
Operacja or 42
Operacja xor 42
Operacja not 42
Przesunięcie bitowe w lewo 42
Przesunięcie bitowe w prawo 43
Rzutowanie 43
Dyrektywy preprocesora 44
Tablice 45
Adresy zmiennych 46
Wskaźniki 47
Wskaźniki a tablice 48
Alokowanie pamięci dla tablicy 49
Rozwiązanie zadania 50
Krok 1. Podejście standardowe 50
Krok 2. Zmniejszenie liczby operacji 51

Krok 3. Oszczędność pamięci 52
Krok 4. Poprawa efektywności algorytmu 57
Wnioski 58
Polecane źródła w Internecie 59
Podstawy informatyki 61

Lekcja 3. Alokowanie pamięci 61

Podstawy języka C 62
Funkcja malloc() 62
Funkcja calloc() 63
Funkcja realloc() 64
Funkcja free() 64
Przykład zarządzania pamięcią 65
Zmienne automatyczne i statyczne 66
Rozwiązanie zadania 67
Wnioski 72
Polecane źródła w Internecie 72

Część II

Lekcja 4. Program główny 75

Podstawy języka C 75
Funkcja main() 75
Funkcja exit () 75
Standardowe strumienie wejścia i wyjścia 76
Napisy 76
Konwersja napisu do liczby 77
Rozwiązanie zadania 78
Krok 1. Zadanie pomocnicze 78
Krok 2. Unikanie prostych błędów 80
Krok 3. Zastosowanie nazwy programu 81
Krok 4. Obsługa opcji 83
Krok 5. Wyświetlenie komunikatu o błędzie 85
Krok 6. Zastosowanie zmiennych środowiskowych 85
Wnioski 86
Polecane źródła w Internecie 86

Lekcja 5. Przetwarzanie opcji 87

Podstawy języka C 87
Przekazywanie argumentów funkcji. 87
Instrukcja wielokrotnego wyboru: switch 89
Standardowe opcje programu 90
Rozwiązanie zadania. 90
Wnioski 94
Polecane źródła w Internecie 94

Lekcja 6. Przetwarzanie parametrów wejściowych – plików 95

Podstawy języka C 95
Odczyt i zapis do pliku 95
Trójargumentowy operator warunkowy 98
Sterowanie preprocesorem. 98
Prototyp funkcji 98
Atrybut extern 100
Kompilacja programu 102
Rozwiązanie zadania. 103
Krok 1. Rozdzielenie kompetencji między funkcje. 103
Krok 2. Rozdzielenie kompetencji między pliki 106
Krok 3. Utworzenie plików nagłówkowych 108
Krok 4. Przetwarzanie danych wejściowych. 109
Krok 5. Kompilacja 110
Aneks 111
Polecane źródła w Internecie 116

Część III

Lekcja 7. Debugowanie programu 119

Na czym polega debugowanie 119
Podstawy języka C 120
Buforowanie. 120
Wyrażenie przecinkowe 121
Funkcja char() 121
Rozwiązanie zadania. 122
Krok 1. Wydruki kontrolne w kodzie programu 122
Krok 2. Zastosowanie makra. 124
Krok 3. Makro ze zmienną liczbą parametrów 126
Krok 4. Debugowanie wybierane dynamicznie 128

Krok 5. Obsługa parametru wywołania +d 130
Wnioski 131
Aneks 132
Modyfikacje w funkcji set_opt(). 132
Modyfikacje w funkcji main() 132
Modyfikacja wypisywania pomocy rozszerzonej. 133
Polecane źródła w Internecie 133

Lekcja 8. Budowanie złożonych programów 135

Podstawy teoretyczne 135
Struktura pliku Makefile 135
Rozwiązanie zadania. 136
Krok 1. Tworzenie pliku Makefile 136
Deklaracje zmiennych. 136
Reguły kompilacji 137
Krok 2. Optymalizacja pliku Makefile 138
Krok 3. Realizowanie celów specjalnych 139
make clean 139
make distclean 140
make dist 140
make install 142
Krok 4. Sprawdzenie poprawności listy zależności 142
Wnioski 143
Polecane źródła w Internecie 143

Lekcja 9. Udostępnianie programu. 145

Narzędzia do udostępniania programów 145
Krok 1. Skanowanie katalogu. 145
Krok 2. Edytowanie pliku configure.ac 146
Krok 3. Edytowanie pliku config.h.in 149
Krok 4. Edytowanie pliku makefile.in 149
Krok 5. Wywołanie programu autoconf 149
Krok 6. Uruchomienie programu configure 150
Krok 7. Korzystanie z pliku config.h. 152
Krok 8. Budowanie programu. 152
Krok 9. Dystrybucja 153
Wnioski 153
Polecane źródła w Internecie 153

Część IV

Lekcja 10. Dynamiczne struktury danych 157

Funkcje rekurencyjne 157
Rekurencyjne struktury danych 159
Dynamiczne struktury danych 160
Listy 160
Drzewa binarne 162
Porównanie przetwarzania list i drzew 163
Realizacja dynamicznych struktur danych w języku C 164
Struktury 164
Rozwiązanie zadania 165
Reprezentacja węzła drzewa 165
Krok 1. Tworzenie drzewa 166
Krok 2. Wstawianie elementu na początek i na koniec listy 168
Krok 3. Zamiana drzewa na listę 170
Krok 4. Zamiana listy na drzewo 175
Krok 5. Wypisywanie zawartości drzewa 180
Krok 6. Wyszukiwanie informacji w drzewie 181
Krok 7. Optymalizacja wyszukiwania 181
Krok 8. Baza z wieloma kluczami 182
Wnioski 183
Polecane źródła w Internecie 183

Lekcja 11. Wyrażenia regularne 185

Automaty skończone 185
Wyrażenia regularne 187
Metaznaki 188
Rozpoznawanie wyrażeń regularnych 189
Rozwiązanie zadania 189
Krok 1. Sformułowanie zadania dla konkretnego zastosowania 189
Krok 2. Budowanie tablicy sterującej 190
Wnioski 197
Polecane źródła w Internecie 198

Lekcja 12. Interpreter poleceń 199

Gramatyki formalne 200
Narzędzia informatyczne 201
Generator parsera yacc 202

Deklaracje języka C. 203
Deklaracje parsera 203
Reguły gramatyki. 203
Kod dodatkowy 204
Generator leksera lex 204
Reguły leksykalne 205
Rozwiązanie zadania. 205
Krok 1. Tworzenie pliku *mag.y* *205*
Krok 2. Generowanie parsera. 207
Krok 3. Tworzenie pliku mag.l 208
Krok 4. Generowanie analizatora leksykalnego 209
Krok 5. Tworzenie prostego programu głównego 210
Krok 6. Modyfikacja pliku Makefile. 212
Krok 7. Wywołanie funkcji yyparse() w pętli 213
Krok 8. Sterowanie reakcjami systemu 215
Modyfikacja leksera 217
Modyfikacja funkcji main() 217
Modyfikacja parsera 218
Krok 9. Rozwijanie interpretera 222
Krok 10. Rozwiązywanie konfliktów. 225
Wnioski 226
Polecane źródła w Internecie 227

Pliki towarzyszące książce 229

Skorowidz 233

Wstęp

Pierwsze prawo „programisty” brzmi: „U mnie działa”. Niezły programista nieco je przeformułuje: „Mój program działa wszędzie”. Dobry programista doda: „Mój program działa wszędzie i dobrze”.

Dobremu programiście nie wystarczy fakt, że program „chodzi”. Dobrze napisany program jest jak dzieło sztuki: piękny i elegancki. Podświadomość programisty zapala czerwone światełko, gdy widzi rozwiązanie nieeleganckie. Zazwyczaj później okazuje się, że w programie jest jakaś „rysa na szkle” – mały defekt, który ujawni się kiedyś zupełnie niespodziewanie.

Dobry program można łatwo odróżnić od „działającego”:

1) szybciej zwraca wyniki;
2) zużywa mniej pamięci;
3) działa dla szerszego zakresu danych.

W tej książce chcemy nauczyć tworzenia **dobrych** programów. Na kilkunastu przykładach pokażemy krok po kroku, jak optymalizować programy – czynić je bardziej efektywnymi.

Język C, wynaleziony w roku 1972, do dziś nie znalazł sobie równych – wraz z językami pochodnymi, takimi jak C++, C# lub Objective-C, tworzy najbardziej popularną rodzinę języków programowania na świecie. Przykłady zawarte w tej książce są przedstawione w „czystym” języku C, ponieważ wszystko, co można w nim zapisać, bez trudu można wyrazić w językach pochodnych, co nie musi zachodzić w przeciwnym kierunku.

Programować można w językach różnego poziomu. Najniższy poziom to język „asemblerowy”, w którym kodujemy bezpośrednio rozkazy maszynowe (a nie instrukcje będące złożeniami rozkazów). C jest językiem średniego poziomu. Zawiera niektóre konstrukcje wysokiego poziomu (unie, tablice, struktury), ale jest jeszcze na tyle „blisko maszyny”, że

można w nim np. zdefiniować fragmenty będące wstawkami w assemblerze. Ponadto, język C umożliwia działania na strukturach dynamicznych, których nie można definiować w tych językach wyższego poziomu, które nie zawierają zmiennych wskaźnikowych.

Z niniejszej książki dowiesz się, jak napisać funkcję, za pomocą której można wyznaczyć liczby pierwsze 56 razy większe niż przykłady podawane w standardowych kursach programowania, i to 7 razy szybciej. Nauczysz się, jak tworzyć programy, którego działanie zależy od argumentów wywołania. Dowiesz się, jak efektywnie napisać interpreter poleceń podanych w języku naturalnym oraz zrozumiesz, jak działa analizator wyrażeń regularnych.

Treść książki ułożono w ciąg lekcji, podzielonych na cztery części. Część I (lekcje 1–3) to nauka pisania funkcji, które oszczędnie i efektywnie gospodarują pamięcią operacyjną. Część II (lekcje 4–6) jest poświęcona strukturze programu głównego. W części III (lekcje 7–9) omówiono proces tworzenia programów złożonych, składających się z kilku modułów, np. napisanych przez różnych autorów. Ostatnia część (lekcje 10–12) zawiera przykłady zastosowania pozyskanych umiejętności. Pokazujemy w niej, jak w sposób przejrzysty i krótki można napisać programy wykonujące niebanalne zadania.

Każda lekcja ma na celu wyrobienie określonej umiejętności programistycznej. Cel ten realizowany jest poprzez rozwiązywanie, krok po kroku, zadania podanego na początku lekcji. Lekcję kończy podsumowanie – lista porad przydatnych każdemu programiście.

Jeśli dopiero zaczynasz przygodę z programowaniem, to ten podręcznik jest dla Ciebie – znajdziesz w nim wyjaśnienia poszczególnych elementów języka C, a już od początku swojej programistycznej edukacji nauczysz się dążyć do perfekcji.

Jeśli programujesz od wielu lat, to zajrzyj do tej książki – zapewne utwierdzisz się w przekonaniu, że potrafisz programować całkiem nieźle, ale być może przekonamy cię, że możesz jeszcze coś udoskonalić w swoim warsztacie.

Przygotowanie środowiska pracy

Aby móc wypróbować, jak działają funkcje i programy opisane w tej książce, musisz, drogi Czytelniku, przygotować odpowiednie środowisko pracy. W tym celu trzeba zainstalować jedną z wersji systemu Linux.

Istnieją dziesiątki tzw. dystrybucji (odmian) systemu Linux, a jej wybór jest sprawą w dużej mierze osobistą. Największą popularnością cieszy się obecnie dystrybucja *Ubuntu* oraz oparta na niej wersja *Linux Mint*. Dla osób rozpoczynających swoją przygodę z Linuksem rekomendujemy stosowanie dystrybucji *OpenSuse*. Jest ona rozwijana przez SUSE – najstarszą, wciąż działającą, organizację linuksową. Jest to według nas wersja najłatwiejsza do przyjęcia dla użytkowników systemu Windows.

Instalacja systemu Linux

Podamy tutaj kilka wskazówek, jak zainstalować dystrybucję OpenSUSE.

Pobranie OpenSUSE

Pakiet można pobrać pod adresem: //software.opensuse.org/421/en. Przed pobraniem warto przeczytać instrukcje podane na tej stronie internetowej.

Przygotowanie instalacji

Najlepszym rozwiązaniem jest nagranie („wypalenie") płyty DVD z pobranym pakietem instalacyjnym. Alternatywnie, pakiet ten można umieścić na dysku USB (wtedy do poprawnej instalacji potrzebny jest program pomocniczy, np. *ImageWriter*). Użytkownikom systemu Windows polecamy dokonanie defragmentacji dysku bezpośrednio przed instalacją (w nowszych wersjach Windows defragmentacja odbywa się samoczynnie co pewien czas). Chodzi o to, aby wszystkie pliki windowsowe zgromadzić na początku dysku, zostawiając jego resztę wolną do użytku przez Linux.

Instalacja

Należy uruchomić komputer z nośnika, na którym nagrany jest program instalacyjny i wybrać opcję *Install*. Procedura instalacyjna zadaje szereg pytań (np. o strefę czasową i język, w jakim ma być zainstalowany system). Na ogół nietrudno na nie odpowiedzieć.

Następnie procedura instalacyjna przystępuje do partycjonowania dysku. Polega to na tym, że w wolnej przestrzeni na dysku system wydziela nowe części (partycje) dla systemu Linux. Cała procedura instalacji w niczym nie zagraża danym przechowywanym w partycji Windows.

Ostatnią czynnością wykonywaną przez procedurę instalacyjną jest instalacja nowego programu rozruchowego. Działa on w trybie dualnym, co oznacza, że po uruchomieniu komputera możemy wybrać system Windows lub Linux.

W ramach instalacji utworzone zostają konta dwóch użytkowników: konto administratora o nazwie *root* oraz konto (zwykłego) użytkownika. Szczególnie istotne jest zapamiętanie hasła administratora.

Uruchamianie programów

W pierwszej części książki omawiamy funkcję, a dopiero w drugiej uczymy, jak napisać działający program. Niecierpliwy czytelnik może jednak chcieć sprawdzić, jak działają funkcje już od pierwszej lekcji. Podajemy teraz krótką instrukcję, jak to zrobić.

Edycja kodu źródłowego

Kod źródłowy programu można utworzyć jest w dowolnym edytorze tekstowym. Polecamy korzystanie z edytorów: *Vim*, *Gvim* lub *Emacs*. Aby sprawdzić działanie funkcji, trzeba napisać choćby najprostszy program z wykorzystaniem tej funkcji. W przypadku funkcji opisanej w lekcji 1 kod źródłowy całego programu może mieć następującą postać:

```
uint
wykladnik(uint n, uint k)
{
  uint w = 0;
    while (n/=k) w += n;
    return (w);
}

void
main()
{
    printf("Liczba 5 występuje w rozkładzie liczby 2000!
           z wykładnikiem %u\n", wykladnik(2000,5));
}
```

Tak utworzony kod źródłowy zapisz pod wybraną przed siebie nazwą (np. *wykladnik.c*) w wybranym przed siebie katalogu.

Przejście do trybu tekstowego

Jeśli system OpenSUSE uruchomił się w trybie graficznym, to należy otworzyć okno, w którym polecenia wydaje się w trybie tekstowym (tzw. emulator konsoli). W tym celu wystarczy uruchomić program o nazwie *Konsole*. Inną metodą jest zastosowanie skrótu klawiaturowego *Ctrl+Alt+Fn*, gdzie *n* jest liczbą od 1 do 6, ale powoduje to opuszczenie trybu graficznego, więc ta metoda powinna być stosowana tylko jako środek ratunkowy, gdy w trybie graficznym coś się zepsuje.

Skompilowanie programu

Przejdź do katalogu (służy do tego polecenie *cd*), w którym został zapisany kod źródłowy programu i skompiluj go za pomocą programu *gcc*. W tym celu wydaj polecenie:

```
$ gcc -o wykladnik wykladnik.c
```

wykladnik to wybrana przez ciebie docelowa nazwa programu. Jeśli w kodzie programu są błędy, zostaniesz o nich poinformowany.

Uruchomienie programu

Jeśli kompilacja przebiegnie bez zakłóceń, to w tym samym katalogu powstanie nowy plik o nazwie *wykladnik*, który możesz uruchomić poprzez podanie polecenia:

```
$ ./wykladnik
```

Polecane źródła w Internecie

The Top 11 Best Linux Distros for 2015
https://www.linux.com/news/top-11-best-linux-distros-2015

The Best Linux Distros of 2016
https://www.linux.com/news/best-linux-distros-2016

openSUSE
https://pl.opensuse.org/

Compiling C and C++ Programs
http://pages.cs.wisc.edu/~beechung/ref/gcc-intro.html

Część I

Optymalizowanie funkcji

Zadanie: napisać funkcję, która, dla danego czynnika pierwszego *k* i liczby naturalnej *n*, obliczy potęgę, w jakiej czynnik *k* występuje w rozkładzie liczby *n!* na czynniki pierwsze.

Elementy matematyki

Rozkład liczby na czynniki pierwsze oznacza zapisanie jej jako iloczynu liczb pierwszych.

Przykładowo, rozkład liczby 60 na czynniki pierwsze zapisujemy w następujący sposób:

$60 = 2 \cdot 2 \cdot 3 \cdot 5$

Kolejność czynników w rozkładzie można zmienić, np.:

$60 = 2 \cdot 3 \cdot 2 \cdot 5$

Powyższe dwa rozkłady liczby 60 będziemy uważać za równoznaczne. Przy założeniu, że kolejność czynników rozkładu jest nieistotna, prawdziwe jest twierdzenie:

Każdą liczbę naturalną można rozłożyć na czynniki pierwsze tylko w jeden sposób.

W dalszym ciągu stosować będziemy tylko taki zapis, w którym czynniki pierwsze podane są w kolejności od najmniejszego do największego. Możemy wtedy rozkład liczby zapisać za pomocą potęg czynników pierwszych, np.

$60 = 2^2 \cdot 3^1 \cdot 5^1$

Powiemy wtedy, że w rozkładzie liczby 60 czynnik pierwszy 2 występuję w potędze drugiej (ma wykładnik 2), a czynniki pierwsze 3 oraz 5 występują w potędze pierwszej (mają wykładnik 1).

Silnia liczby naturalnej **n,** oznaczana jako **n!**, to iloczyn wszystkich liczb naturalnych nie większych niż **n**.

Przykładowo:

$$6! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = 720$$

Silnię liczby naturalnej **n** można zdefiniować również **rekurencyjnie**, czyli za pomocą definicji, która odwołuje się do samej siebie:

$$n! = \begin{cases} 1 & \text{dla } n = 1 \\ n \cdot (n-1)! & \text{dla } n > 1 \end{cases}$$

Aby obliczyć wartość **6!** korzystając z definicji rekurencyjnej, dokonujemy następujących wyliczeń cząstkowych:

$$\begin{aligned} 6! &= \\ &= 6 \cdot 5! = \\ &= 6 \cdot 5 \cdot 4! = \\ &= 6 \cdot 5 \cdot 4 \cdot 3! = \\ &= 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2! = \\ &= 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1! = \\ &= 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 720 \end{aligned}$$

Podstawy języka C

W tej części przybliżymy podstawowe elementy języka programowania C.

Program, instrukcja

Program to ciąg instrukcji wykonywanych zgodnie z kolejnością ich zapisania. W języku C każdą instrukcję kończy znak średnika. Instrukcja w języku C ma zatem postać:

nazwa_instrukcji (*lista parametrów***);**

Przykładem instrukcji w języku C jest instrukcja **printf**, która wypisuje komunikat na urządzenie domyślne (np. monitor komputera). Komunikat kończony jest zwykle komendą przejścia do nowego wiersza **\n**, np.

```
printf("Witaj\n");        // wypisanie na ekran napisu 'Witaj'
```

Składnia języka C dopuszcza zastąpienie jednej instrukcji blokiem instrukcji. Instrukcje łączymy w bloki za pomocą nawiasów klamrowych { }, np.:

```
{
    printf("Witaj\n");
    printf("Jak się nazywasz?");
}
```

Komentarze

Komentarze to teksty pomocnicze, które nie są wykonywane przez program. Zaczynają się znakami ***/**** i kończą znakami ****/***.

Komentarz mieszczący się w jednym wierszu można również zapisać za znakami ***//***.

```
/* To moj pierwszy program */
printf("Witaj");          // Wypisanie komunikatu powitalnego
```

Identyfikatory

Identyfikator to nazwa stworzona przez programistę. Identyfikator musi zaczynać się od małej lub dużej litery alfabetu łacińskiego albo znaku podkreślenia; może zawierać ponadto litery, znaki podkreślenia lub cyfry.

Przykładowe identyfikatory: *wykladnik, _silnia, n, k1.*

Słowa kluczowe

Słowa kluczowe to wyrazy zarezerwowane, których nie można stosować jako identyfikatorów, np. *char, switch, return*.

Białe znaki

Białe znaki oznaczają spacje, tabulatory, znaki przejścia do następnego wiersza, a także komentarze. Służą one do oddzielania poszczególnych elementów języka, np.

```
m = n/*ujemna*/ - k/*ujemnik*/;
```

W powyższym przykładzie spacje oraz komentarze oddzielają identyfikatory oraz operatory (przypisania i odejmowania).

Podstawowe typy numeryczne

Typ *zmiennej* określa, jakie wartości może przyjmować *zmienna*, a co za tym idzie, ile miejsca w pamięci powinno zostać dla niej zarezerwowane.

Typ numeryczny *zmiennej* oznacza, że może ona przyjmować wyłącznie wartości będące liczbami. W języku C typy numeryczne dzieli się na całkowite oraz zmiennoprzecinkowe.

Przykładowe typy całkowite:

Typ	Zajmowana pamięć	Zakres wartości
char	1 bajt	od (-128) do 127 lub od 0 do 255
unsigned char	1 bajt	od 0 do 255
signed char	1 bajt	od -128 do 127
int	4 bajty	od -2 147 483 648 do 2 147 483 647
unsigned int	4 bajty	od 0 do 4 294 967 295

Przykładowe typy rzeczywiste:

Typ	Zajmowana pamięć	Zakres wartości	Dokładność
float	4 bajty	od 1.2E-38 do 3.4E+38	6 miejsc znaczących
double	8 bajtów	od 2.3E-308 do 1.7E+308	15 miejsc znaczących
long double	16 bajtów	od 3.4E-4932 do 1.1E+4932	19 miejsc znaczących

Nazwy typów programista może przemianować za pomocą makra *#define* (więcej informacji o makrach można znaleźć w lekcji 2, w części *Dyrektywy preprocesora*), np.:

```
#define  uint  unsigned int
```

Od tego momentu typ *uint* będzie traktowany jak *unsigned int*. Podobny efekt można uzyskać za pomocą słowa kluczowego **typedef**:

```
typedef  unsigned int uint;
```