C

TABLICE INFORMATYCZNE • Radosław Sokół

Helion

Szkielet programu

Biblioteka standardowa języka C rozpoczyna realizację programu użytkownika od podprogramu nazwanego `main` o jednym z następujących prototypów:

`int main();`	Wariant bezargumentowy
`int main(char *argumenty[]);`	Wariant przyjmujący argumenty wiersza polecenia

Szkielet programu nie wykonującego żadnej czynności może wyglądać następująco:

```
int main() {
    return 0;
}
```

```
int main(char *argumenty[]) {
    return 0;
}
```

Wartość zwracana przez instrukcję `return` zazwyczaj stanowi kod błędu i jest przekazywana do systemu operacyjnego i dostępna dla innych programów. Typowo wartość 0 oznacza poprawne wykonanie operacji, a wyższe wartości — błędy o rosnącym poziomie istotności.

Instrukcje

Każda instrukcja jest kończona średnikiem:

```
instrukcja1;
instrukcja2;
....
instrukcjaN;
```

Za każdym razem, gdy istnieje potrzeba potraktowania kilku instrukcji jak jednej, sekwencję instrukcji należy zawrzeć w nawiasach klamrowych:

```
{
    instrukcja1;
    ....
    instrukcjaN;
}
```

Ponadto blok ograniczony klamrami stanowi odrębną jednostkę programu w zakresie przechowywania danych.

Zmienne zadeklarowane w bloku są dostępne wyłącznie w czasie realizowania instrukcji tego bloku, a zajmowana przez nie pamięć jest zwalniana przy opuszczaniu bloku.

Komentarze

Fragmenty tekstu źródłowego zawarte pomiędzy symbolami `/*` oraz `*/` są traktowane jako komentarz, to znaczy ignorowane podczas translacji programu na postać wynikową. Komentarz może mieć dowolną objętość, od fragmentu jednego wiersza aż po wiele wierszy.

Od wersji C99 język C dopuszcza stosowanie jednowierszowych komentarzy w stylu języka C++, rozpoczynających się symbolami `//` i obowiązujących do końca wiersza:

```
/* Komentarz
   wielowierszowy */
// Komentarz jednowierszowy
```

Wskazówka Komentarze mogą być wykorzystywane do wyłączania chwilowo niepotrzebnych fragmentów tekstu źródłowego programu z kompilacji, na przykład w celu sprawdzenia działania innego wariantu tego samego fragmentu.

Kompilowanie programu

Sposób kompilowania programu zależy od zastosowanego kompilatora lub środowiska programistycznego. W poniższych przykładach został uwzględniony kompilator `gcc`, stanowiący standard w środowisku otwartego oprogramowania.

`gcc –c plik.c –o plik.o`	Kompilacja pojedynczego pliku źródłowego (`plik.c`) do postaci wynikowej pośredniej (`plik.o`).
`gcc plik1.c plik2.o ... –o program`	Kompilacja i konsolidacja wielu plików źródłowych i pośrednich do postaci pliku wynikowego (`program`). W przypadku systemu Windows należy do nazwy programu wynikowego dodać rozszerzenie *.exe*.

Najczęściej używane opcje `gcc` stosowane na etapie kompilacji i/lub konsolidacji:

`-I`*`ścieżka`*	Wskazanie katalogu dostępnego pod ścieżką *`ścieżka`* jako źródła dodatkowych plików nagłówkowych (*.h*) zawierających deklaracje niestandardowych modułów bibliotecznych.
`-D`*`symbol`*	Zdefiniowanie symbolu o nazwie *`symbol`* w celu obsłużenia go na etapie przetwarzania wstępnego.
`-D`*`symbol=wartość`*	Zdefiniowanie symbolu o nazwie *`symbol`* i nadanie mu wartości *`wartość`* w celu wykorzystania jej na etapie przetwarzania wstępnego.
`-Wall`	Włączenie najwyższego poziomu wykrywania podejrzanych konstrukcji w tekście źródłowym.

`-O0`
`-O1`
`-O2`
`-O3`
`-Os`
	Wybór poziomu optymalizacji kodu wynikowego. Poziom 0 wyłącza wszystkie optymalizacje, poziomy 1, 2, 3 powodują włączenie kolejnych zestawów optymalizacji po stronie kompilatora języka C. Poziom specjalny `s` powoduje optymalizację pod kątem uzyskania możliwie najmniej obszernego, lecz wciąż wydajnego kodu wynikowego.
	Wskazówka Ze względu na specyfikę obecnych mikroprocesorów optymalizacja pod kątem minimalnego rozmiaru kodu wynikowego (`-Os`) często przynosi najlepsze rezultaty.

`-s`	Obowiązuje tylko na etapie konsolidacji. Powoduje wyłączenie umieszczania w pliku wynikowym informacji pozwalających częściowo odtworzyć strukturę tekstu źródłowego, które usprawniają proces wyszukiwania błędów w programie. Należy stosować w ostatecznych wersjach programów trafiających do klientów końcowych.
`-g`	Dołącza do pliku pośredniego lub wynikowego dodatkowe informacje pozwalające prowadzić śledzenie i analizę działania programu. Opcja powoduje zwiększenie objętości generowanych plików i ułatwia odkrycie budowy programu. Powinna być włączana tylko przy generowaniu wersji do użytku wewnętrznego.

PRZECHOWYWANIE I PRZETWARZANIE DANYCH

Typy zmiennych

Typ zmiennej określa ilość pamięci poświęcanej na przechowywanie porcji informacji oraz sposób, w jaki kompilator będzie traktował tę informację.

Zapis	Alternatywny zapis	Znaczenie
`void`		Brak wartości
`char`		Liczba 8-bitowa, naturalna lub całkowita, albo znak alfanumeryczny
`short`	`short int`	Liczba 8- lub 16-bitowa, naturalna lub całkowita
`int`		Liczba 16- lub 32-bitowa, naturalna lub całkowita
`long`	`long int`	Liczba 32- lub 64-bitowa, naturalna lub całkowita
`long long`	`long long int`	Liczba 64-bitowa, naturalna lub całkowita
`float`		Liczba zmiennoprzecinkowa, typowo 32-bitowa
`double`		Liczba zmiennoprzecinkowa, typowo 64-bitowa
`long double`		Liczba zmiennoprzecinkowa, typowo 80-bitowa
`bool`		Wartość logiczna (`true`/`false`)

Obsługa typu `bool` wymaga dołączenia pliku nagłówkowego *stdbool.h*:

```
#include <stdbool.h>
```

Do typów całkowitoliczbowych (`char`, `short`, `int`, `long`, `long long`) mogą być stosowane dodatkowe kwantyfikatory, `signed` i `unsigned`, określające dopuszczalność liczb ujemnych.

`signed`	Liczby całkowite
`unsigned`	Liczby naturalne wraz z liczbą zero

Jeżeli nie zostanie użyty żaden kwantyfikator, domyślnie przyjmowany jest `signed`. Wyjątkiem jest typ `char`, który w zależności od implementacji może być traktowany domyślnie jako `signed` lub `unsigned`. W miejscach, gdzie zakres liczbowy i dopuszczalność liczb ujemnych są istotne, należy wprost wprowadzać odpowiednie kwantyfikatory.

Standard nie definiuje zakresów liczbowych poszczególnych typów, jednak w przypadku programów kompilowanych przez `gcc` i uruchamianych na komputerach 32- i 64-bitowych zakresy są następujące:

Typ	Rozmiar bitowy	Zakres	
		`signed`	`unsigned`
`char`	8	–128 ÷ 127	0 ÷ 255
`short`	16	–32 768 ÷ 32 767	0 ÷ 65 535
`int`	32	–2 147 483 647 ÷ 2 147 483 647	0 ÷ 4 294 967 295
`long`	32	–2 147 483 647 ÷ 2 147 483 647	0 ÷ 4 294 967 295
`long long`	64	–9 223 372 036 854 775 808 ÷ 9 223 372 036 854 775 807	0 ÷ 18 446 744 073 709 551 615

Ponadto wiele funkcji biblioteki standardowej zwraca wartości typu `size_t`. Jest to typ zdefiniowany za pomocą wyrażenia `typedef`, odpowiadający optymalnemu dla danej platformy sprzętowej i programowej typowi przechowującemu rozmiar obszaru pamięci (typowo `unsigned long int` lub `unsigned long long int`).

Definiowanie zmiennych

Definicja zmiennej rezerwuje miejsce na stosie procesora oraz określa typ i nazwę zmiennej. Możliwe jest również nadanie początkowej wartości zmiennej:

- *`typ nazwa;`*
- *`typ nazwa = wartość;`*
- *`atrybut typ nazwa;`*
- *`atrybut typ nazwa = wartość;`*

Nazwa zmiennej może składać się z liter alfabetu łacińskiego, cyfr oraz znaku podkreślenia (`_`). Nazwa nie może zaczynać się od cyfry. Małe i wielkie litery są odróżniane (`liczba` i `Liczba` to dwie różne zmienne). Zmienna nie może być nazwana zarezerwowanym słowem kluczowym języka C.

W przypadku zmiennych o tym samym typie (i opcjonalnym atrybucie) możliwe jest połączenie kilku deklaracji w jednym wyrażeniu:

- *`typ nazwa1, nazwa2, ...;`*
- *`atrybut typ nazwa1, nazwa2, ...;`*
- *`typ nazwa1 = wartość1, nazwa2, nazwa3 = wartość3, ...;`*
- *`atrybut typ nazwa1 = wartość1, nazwa2, nazwa3 = wartość3, ...;`*

Możliwe nazwy atrybutów:

Nazwa	Znaczenie
`auto`	Zmienna lokowana na stosie mikroprocesora. Atrybut domyślny, stosowany przy braku innych.
`register`	Zmienna lokowana w rejestrze mikroprocesora. Podnosi szybkość operowania na zawartości zmiennej. Liczba rejestrów jest ograniczona, należy stosować w bardzo ograniczonym zakresie. Kompilator ma prawo traktować to wyłącznie jako sugestię i pominąć w razie braku możliwości wykorzystania rejestru mikroprocesora.
`volatile`	Wyłączenie możliwości optymalizowania dostępu do zawartości zmiennej. Atrybut wykorzystywany w stosunku do zmiennych reprezentujących rejestry sprzętowe urządzeń lub stosowanych w środowisku współbieżnym.
`const`	Blokada możliwości zmiany zawartości zmiennej po jej początkowym nadaniu.

Przykłady:

```
int licznik;
double mnoznik = 12.45;
```

Literały

Literały to wartości stałe wstawiane bezpośrednio w tekst źródłowy programu. Literałom również mogą być nadawane typy.

Opcjonalne przedrostki literałów liczbowych:

`0x`	Liczba zapisana szesnastkowo, np. 0x21 to wartość 21 szesnastkowo (33 dziesiętnie)
`0`	Liczba zapisana ósemkowo, np. 023 to wartość 23 ósemkowo (19 dziesiętnie)

Opcjonalne przyrostki literałów liczbowych:

`u`	Liczba naturalna (`unsigned`)
`l`	Liczba typu `long`
`ll`	Liczba typu `long long`
`ul`	Liczba naturalna `unsigned long`
`ull`	Liczba naturalna `unsigned long long`
`f`	Liczba typu `float`

Liczby zmiennoprzecinkowe mogą być zapisywane w **formie wykładniczej**:

`1.34e1`	$1{,}34\times10^{1}$, czyli 13,4
`-4.78e-2`	$-4{,}78\times10^{-2}$, czyli –0,0478

W przypadku liczb zmiennoprzecinkowych stosuje się zapis wyłącznie z kropką dziesiętną. Liczby nie mogą być formatowane z wykorzystaniem znaków odstępu.

`1,45`	ŹLE: przecinek użyty zamiast kropki
`1 234 567`	ŹLE: znaki odstępu rozdzielają grupy cyfr
`123.456`	DOBRZE: prawidłowo zapisana liczba

Literały zmiennoprzecinkowe są domyślnie typu `double`. Aby uzyskać typ `float`, należy użyć przyrostka `f`, natomiast aby uzyskać typ `long double`, należy użyć przyrostka `l`.

Wyrażenia algebraiczne

Na podstawie literałów i wartości zmiennych można wyznaczać nowe wartości, używane bezpośrednio lub przypisywane do tej samej lub innej zmiennej.

Wynik wyrażenia algebraicznego ma typ określony przez pierwszy jego element. Pozostałe elementy są konwertowane do tego typu, co może skutkować utratą danych. W przypadkach wątpliwych należy stosować jawne rzutowanie.

Operatory arytmetyczne:

Operator	Znaczenie
`a + b`	Suma dwóch liczb
`a – b`	Różnica dwóch liczb
`a * b`	Iloczyn dwóch liczb
`a / b`	Iloraz dwóch liczb; w przypadku liczb całkowitych jest to część całkowita ilorazu
`a % b`	Dzielenie modulo dwóch liczb całkowitych, wynikiem jest reszta z dzielenia całkowitego *`a`* przez *`b`*
`++a`	Zwiększenie liczby całkowitej o 1, wynikiem wyrażenia jest zmodyfikowana wartość (preinkrementacja)
`--a`	Zmniejszenie liczby całkowitej o 1, wynikiem wyrażenia jest zmodyfikowana wartość (predekrementacja)
`a++`	Zwiększenie liczby całkowitej o 1, jednak wynikiem wyrażenia jest poprzednia wartość *`a`* (postinkrementacja)
`a--`	Zmniejszenie liczby całkowitej o 1, jednak wynikiem wyrażenia jest poprzednia wartość *`a`* (postdekrementacja)