Programowanie w elektronice: Podstawy C

Projekt „Matematyka dla Ciekawych Świata”,
Robert Ryszard Paciorek
<rrp@opcode.eu.org>

2019-07-22

C / C++ są najpopularniejszymi językami kompilowanymi do kodu maszynowego (a jeżeli traktować je łącznie to najpopularniejszymi językami w ogóle), pozwalają na stosowanie niskopoziomowych mechanizmów (łącznie z wstawkami asemblerowymi), są użyteczne do bezpośredniego programowania sprzętu (bez warstwy systemu operacyjnego) czy też tworzenia systemów operacyjnych.

Język C jest językiem kompilowalnym to znaczy (po zmodyfikowaniu źródeł) przed uruchomieniem programu konieczne jest dokonanie tłumaczenia kodu źródłowego na kod maszynowy przy pomocy odpowiedniego programu (np. clang lub gcc). Kompilacja przebiega kilku etapowo. W pierwszej kolejności wywoływany jest preprocesor, który jest odpowiedzialny za włączanie plików określonych poprzez `#include` (jest to literalne włączenie zawartości wskazanego pliku w danym miejscu, obsługę rozwijania stałych makr preprocesora (definiowanych z użyciem `#define`) oraz kompilację warunkową z wykorzystaniem poleceń takich jak `#ifdef` czy `#if`. Kompilatory pozwalają na uzyskanie nie tylko wynikowego pliku binarnego, ale także plików po przetworzeniu przez preprocesor czy też po konwersji na assembler.

1 Zmienne

Język C wymaga określania typu zmiennej w momencie jej definiowania.

```
// liczba całkowita ze znakiem
int     liczbaA = -34;
// liczba rzeczywista (pojedynczej precyzji)
float   liczbaB = 673.1;
// 8 bitowa liczba całkowita bez znaku, wymaga pliku nagłówkowego inttypes.h
uint8_t liczbaC = 0xf3;

// zmienna napisowa "C NULL-end string"
char* napisA = "q we";
```

2 Podstawowe operacje

Dodawanie, mnożenie, odejmowanie zapisuje się i działają one tak jak w normalnej matematyce, dzielenie zapisuje się przy pomocy ukośnika i zależnie od typów na których operuje jest ono dzieleniem całkowitym lub zmiennoprzecinkowym.

```
double a = 12.7, b = 3, c, d, e;
int x = 5, y = 6, z;

// dodawanie, mnożenie, odejmowanie zapisuje się
// i działają one tak jak w normalnej matematyce:
e = (a + b) * 4 - y;
```

```
// dzielenie zależy od typów argumentów
d = a / b; // będzie dzieleniem zmiennoprzecinkowym bo a i b są typu float
c = x / y; // będzie dzieleniem całkowitym bo z i y są zmiennymi typu int
b = (int)a / (int)b; // będzie dzieleniem całkowitym
a = (double)x / (double)y; // będzie dzieleniem zmiennoprzecinkowym

// reszta z dzielenia (tylko dla argumentów całkowitych)
z = x % y;

// wypisanie wyników
printf("%d %f %f %f %f %f\n", z, e, d, c, b, a);

// operacje logiczne:
// ((a większe równe od 0) AND (b mniejsze od 2)) OR (z równe 5)
z = (a>=0 && b<2) || z == 5;
// negacja logiczna z
x = !z;

printf("%d %d\n", z, x);

// operacje binarne:
// bitowy OR 0x0f z 0x11 i przesunięcie wyniku o 1 w lewo
x = (0x0f | 0x11) << 1;
// bitowy XOR 0x0f z 0x11
y = (0x0f ^ 0x11);
// negacja bitowa wyniku bitowego AND 0xfff i 0x0f0
z = ~(0xfff & 0x0f0);

printf("%x %x %x\n", x, y, z);
```

3 Przepływ sterowania w programie - skoki, warunki, pętle, funkcje

Licznik programu (program counter, instruction pointer lub instruction address register) jest rejestrem procesora który określa adres następnej (w niektórych architekturach aktualnej) instrukcji która ma zostać przetworzona procesor.

Skoki bezwarunkowe, instrukcje warunkowe, pętle, wywołania funkcji są realizowane poprzez modyfikację licznika programu. W przypadku wywołań funkcji dodatkowo wykonywane są operacje związane z obsługą stosu (zachowywaniem stanu rejestrów, umieszczaniem argumentów na stosie, ...). Instrukcja goto (realizująca skok bezwarunkowy) jest pełnoprawną instrukcją skoku, jedyną wadą jej stosowania jest to że przy niewłaściwym / zbyt częstym wykorzystywaniu (zamiast wywołań funkcji, warunków i pętli) kod programu staje się mniej czytelny.

W większości przypadków pętle realizowane są na poziomie kodu maszynowego jako zestaw instrukcji (np. inkrementacji zmiennej, sprawdzania warunku, skoku), jednak w niektórych rozwiązaniach pętle (np. typu ”powtórz n razy”) mogą być realizowane sprzętowo przy pomocy pojedynczej instrukcji.

3.1 Punkt startu

Jako że program komputerowy jest sekwencją wykonywanych instrukcji musi rozpoczynać się od określonego miejsca. W przypadku kodu C/C++ punktem startu jest funkcja `main()`. Zakończenie tej funkcji

oznacza zakończenie programu, a wartość przez nią zwracana odpowiedzialna jest za tzw. kod powrotu przekazany procesowi wywołującemu program.

```c
#include <stdio.h> // włączenie pliku nagłówkowego

int main() {
  int i, j, k;

  // instrukcja waunkowa if - else
  if (i<j) {
    puts("i<j");
  } else if (j<k) {
    puts("i>=j AND j<k");
  } else {
    puts("i>=j AND j>=k");
  }

  // podstawowe operatory logiczne
  if (i<j || j<k)
    puts("i<j OR j<k");
  // innymi operatorami logicznymi są && (AND), ! (NOT)

  // pętla for
  for (i=2; i<=9; ++i) {
    if (i==3)
      continue; // pominięcie tego kroku pętli
    if (i==7)
      break; // wyjście z pętli
    printf(" a: %d\n", i);
  }

  // pętla while
  while (i>0) {
    printf(" b: %d\n", --i);
  }

  // pętla do - while
  do {
    printf(" c: %d\n", ++i);
  } while (i<2);

  // instrukcja wyboru switch
  switch(i) {
    case 1:
      puts("i==1");
      break;
    default:
      puts("i!=1");
      break;
  }
}
```

3.2 Własne funkcje

```c
#include <stdio.h>

// funkcja bezargumentowa niezwracająca wartości
void f1() {
  puts("ABC");
}

// funkcja dwuargumentowa zwracająca wartość
int f2(int a, int b) {
  puts("F2");
  return a*2.5 + b;
}

int main() {
  f1();

  int a = f2(3, 6);
  // zwracaną wartość można wykorzystać (jak wyżej) lub zignorować

  printf("%d\n", a);
}
```

4 Złożone typy danych

4.1 Struktury

Struktura jest złożonym typem danych służącym do grupowania powiązanych ze sobą logicznie zmiennych. Zmienne wchodzące w skład struktury (pola) identyfikowane są nazwami i mogą być różnych typów. Struktura zajmuje ciągły obszar pamięci, w którym umieszczane są wartości kolejnych pól.

```c
#include <stdio.h>

struct Struktura {
  int a, b;
  double c;
};

int main() {
  struct Struktura s;
  s.a = 13;
  s.c = 17.3;
  printf("%f\n", s.a + s.c);
}
```

4.2 Tablice

Tablica jest strukturą danych w której elementy (takiego samego typu) są ułożone w porządku liniowym i są dostępne za pomocą indeksów (kluczy). Typowo tablica indeksowana jest liczbami całkowitymi nie ujemnymi oraz zajmuje ciągły obszar pamięci.

```c
#include <stdio.h>

int main() {
  int t[4] = {1, 8, 3, 2};
  printf("%d -> \n", t[2]);
  t[2] = 55;
  printf("  %d \n", t[2]);
}
```

4.3 Napisy

Napisy w C są w istocie tablicami elementów typu **char**. Pojedynczy znak reprezentowany jest poprzez jeden element tablicy (dla znaków kodowanych jednobajtowo) lub grupę takich elementów (dla znaków kodowanych wielobajtowo, np. polskich znaczków w UTF8). Koniec napisu oznaczany jest przez element o wartości zero (NULL).

5 Zmienna i jej adres

Wszelkie dane na których operuje program komputerowy przechowywane są w jakimś rodzaju pamięci - najczęściej jest to pamięć operacyjna. W pewnych sytuacjach niektóre dane mogą być przechowywane np. tylko w rejestrach procesora lub rejestrach urządzeń wejścia-wyjścia.

W programowaniu na poziomie wyższym od kodu maszynowego i asemblera używa się pojęcia zmiennej i (niemal zawsze) pozostawia kompilatorowi/interpretatorowi decyzję o tym gdzie ona jest przechowywana. Oczywistym wyjątkiem są grupy zmiennych, czy też bufory alokowane w sposób jawny w pamięci. Ze względu na ograniczoną liczbę rejestrów procesora większość zmiennych (w szczególności tych dłużej istniejących i większych) będzie znajdowała się w pamięci i będą przenoszone do rejestrów celem wykonania jakiś operacji na nich po czym wynik będzie przenoszony do pamięci.

Z każdą zmienną przechowywaną w pamięci związany jest *adres pamięci* pod którym się ona znajduje. Niektóre z języków programowania pozwalają na odwoływanie się do niego poprzez wskaźnik na zmienną lub referencję do zmiennej (odwołania do adresu zmiennej mogą wymusić umieszczenie jej w pamięci nawet gdyby normalnie znajdowała się tylko w rejestrze procesora).

Wszystkie dane są zapisywane w postaci liczb lub ciągów liczb. Typ zmiennej (jawny lub nie) informuje o tym jakiej długości jest dana liczba i jak należy ją interpretować (jak należy interpretować ciąg liczb).

5.1 Zasięg zmiennej

Zasięg zmiennych (widoczność i istnienie) jest limitowany do bloku (wydzielanego nawiasami klamrowymi) w którym zostały zadeklarowane, zmienne z bloków wewnętrznych mogą przesłaniać zmienne zadeklarowane wcześniej.

Wywołanie funkcji powoduje rozpoczęcie nowego kontekstu w którym zmienne z bloku wywołującego funkcję nie są widoczne (ale nadal istnieją). Argumenty do funkcji przekazywane są przez kopiowanie, więc funkcja nie ma możliwości modyfikacji zmiennych z bloku ją wywołującego nawet do niej przekazanych (wyjątkiem jest przekazanie przez referencję lub wskaźnik).

W przypadku manualnej alokacji pamięci (z użyciem malloc) limitowana jest widoczność i istnienie otrzymanego wskaźnika, ale nie zaalokowanego bloku pamięci. Zatem ograniczona jest widoczność takich zmiennych ale nie czas ich istnienia, dlatego też przed utratą wskaźnika na nie należy je usunąć (zwolnić zaalokowaną pamięć).

5.2 Wskaźniki

Wskaźnik jest zmienną, która przechowuje adres pamięci, pod którym znajdują się jakieś dane (inna zmienna). Jako że wskaźnik jest zmienną która też jest umieszczona gdzieś w pamięci można utworzyć wskaźnik do wskaźnika itd. Na wskaźnikach można wykonywać operacje arytmetyczne (najczęściej jest to dodawanie offsetu). Na wskaźniku można wykonać operację wyłuskania czyli odwołania się do wartości zmiennej pod adresem na który wskazuje, a nie do zmiennej wskaźnikowej (zawierającej adres).

Wskaźniki pozwalają na operowanie dużymi zbiorami danych (duże struktury, napisy, etc) bez konieczności ich kopiowania przy przekazywaniu do funkcji, umieszczaniu w różnych strukturach danych, sortowaniu, itd (kopiowaniu ulega jedynie wskaźnik czyli adres) oraz na współdzielenie tych samych danych pomiędzy różnymi obiektami.

Wskaźnik może wskazywać na niewłaściwy adres w pamięci (np. na skutek zwolnienia tego fragmentu lub błędu w operacjach matematycznych na wskaźnikach - wyjściu poza dozwolony zakres), typowo wskaźnikowi który nic nie wskazuje przypisuje się wartość `NULL` (zero). Wyłuskania wskaźników o wartości `NULL` lub wskazujących niewłaściwy obszar pamięci prowadzą do błędów programu, często do zakończenia programu z powodu naruszenia ochrony pamięci ("Segmentation fault").

```c
#include <stdio.h>

int main() {
  int zm = 13;
  int *wsk = NULL; // zmienna wskaźnikowa (na typ int)

  // przypisanie do zmiennej wskaźnikowej adresu zmiennej zm
  // pobranie adresu zmiennej przy pomocy operatora &
  wsk = &zm;
  printf("%p %p\n", &zm, wsk);

  // odwołanie do zmiennej wskazywanej przez wskaźnik (wyłuskanie wartości)
  // przy pomocy operatora *
  printf("%d %d\n", zm, *wsk);
  *wsk = 17;
  printf("%d %d\n", zm, *wsk);
}
```

5.3 Wskaźniki a tablice

Zmienna tablicowa w C to w istocie wskaźnik na pierwszy element tablicy. Dostęp do elementów tablicy odbywa się w oparciu o obliczanie ich adresu na podstawie zależności: AdresElementu = AdresPoczatkuTablicy + IndexElementu * RozmiarElementu.

```c
#include <stdio.h>

int main() {
  int t[4] = {1, 8, 3, 2};
  int *tt = t; // zauważ brak operatora pobrania adresu

  printf("%d %d\n", t[2], tt[2]);
  printf("%d %d\n", *(t + 2), *(tt + 2));
}
```

Zauważ że operator `t[x]` działa tak samo dla tablicy jak i dla wskaźnika i jest w istocie ładniejszym zapisem operacji `*(t+x)` na samym wskaźniku.

5.4 Arytmetyka wskaźnikowa

Jak już zauważyliśmy na wskaźnikach można wykonywać (niektóre) operacje arytmetyczne. Ich działanie jest zależne od typu wskaźnika, tj. zwiększenie wskaźnika o 1 zwiększa adres na który on wskazuje o tyle bajtów ile zajmuje zmienna której typu jest wskaźnik.

```c
#include <stdio.h>

int main() {
  char a;  int  b;
  char *wsk_a = &a;
  int  *wsk_b = &b;

  printf("char: %p %p\n", wsk_a, wsk_a+1);
  printf("int:  %p %p\n", wsk_b, wsk_b+1);
}
```

5.5 Kolejność bajtów

Wskaźniki i rzutowanie typów pozwala patrzeć na dane w postaci poszczególnych bajtów.

```c
#include <inttypes.h>
#include <stdio.h>
int main() {
        // dane jako tablica liczb 16 bitowych
        uint16_t aa[4] = {0x1234, 0x5678, 0x9abc, 0xdeff};

        // wypisujemy ją
        printf("A0: %x %x %x %x\n", aa[0], aa[1], aa[2], aa[3]);
        // chyba nikogo nie zaskoczy wynik powyższego printf:
        //   A0: 1234 5678 9abc deff

        // wypisujemy dwie pierwsze liczby rozłożone na części 8 bitowe
        // (poszczególne bajty)
        printf(
                "A1: %x %x %x %x\n",
                (aa[0] >> 8) & 0xff, aa[0] & 0xff,
                (aa[0] >> 8) & 0xff, aa[0] & 0xff
        );
        // efekt też jest oczywisty:  A1: 12 34 12 34

        // każemy na te same dane patrzeć jako na liczby 8 bitowe
        // (poszczególne bajty)
        uint8_t* bb = (uint8_t*) aa;

        printf("B0: %x %x %x %x\n", bb[0], bb[1], bb[2], bb[3]);
        // czego się teraz spodziewamy?
        //  - wypisze nam tylko połowę oryginalnej tablicy
        //  - ale dokładny wynik zależy od architektury na której uruchamiamy
        //    program (big endian vs little endian)
}
```

Fakt, że różne komputery ten sam ciąg zero-jedynkowy mogą interpretować jako różne liczby (w zależności od architektury „big endian” vs „little endian”), powoduje że przy wymianie danych między systemami konieczne jest ustalenie sposobu tej interpretacji (np. protokoły sieciowe takie jak IP używają „big endian”) lub zawarcie tej informacji w wymienianych danych (kodowania Unicode UTF-16 i UTF-32 zawierają na początku danych znacznik BOM).

6 Zadania

Zadanie 6.0.1

Napisz program wypisujący ”Hello World”.

Zadanie 6.0.2

Zmodyfikuj program z zadania 6.0.1 tak aby z użyciem pętli wypisywał ten napis 10 razy.

Zadanie 6.0.3

Napisz funkcję `wypisz`, która z użyciem pętli będzie wypisywała poszczególne znaki podanego napisu od wskazanej do wskazanej pozycji.
Wywołanie `wypisz("! Hello World !", 3, 6);` powinno spowodować wypisanie `ello`.

Zadanie 6.0.4

Zmodyfikuj program z zadania 6.0.3 tak aby funkcja korzystała z arytmetyki wskaźnikowej, zamiast iteracji po numerze znaku w napisie.