Programowe interpretery poleceń w C

Urządzenia wykonawcze podłączane do sterowników stosowane są w technice bardzo szeroko - począwszy od prostych sterowników typu włącz-wyłącz, skończywszy na bardzo skomplikowanych, umożliwiających regulację proporcjonalną. Najczęściej to wymagania aplikacji wyznaczają budowę układu wykonawczego i sposób jego sterowania.

W artykule przedstawię sposób wykorzystania języka C do budowy prostego interpretera poleceń wysyłanych przez interfejs szeregowy, za pomocą których można będzie sterować pracą dowolnego układu wykonawczego.

Na początek konwersja typów

Niektórzy z Czytelników z całą pewnością zetknęli się z językiem programowania Pascal oraz specyficznym typem zmiennych tzw. typem proceduralnym. W dużym uproszczeniu polegał on na tym, że odwołanie do zmiennej miało podobne konsekwencje jak wywołania procedury. Kompilator języka C nie oferuje podobnego typu zmiennych, a momentami byłby on bardzo przydatny. Ot chociażby do konstrukcji opisywanego w poprzedniej części kursu interaktywnego menu - wybranie w nim opcji mogłoby bezpośrednio powodować uruchomienie odpowiadającej jej funkcji. Realizacja tak pojętego typu proceduralnego jest możliwa dzięki wykorzystaniu wskaźników oraz mechanizmów przekształceń typów.

Na **list. 1** umieściłem fragment programu wraz z definicjami odpowiednich typów zmiennych. Jako pierwszy zdefiniowano typ zmiennej będącej wskaźnikiem typu *char*, kwalifikowanym do przestrzeni adresowej zawierającej kod programu mikrokontrolera (*code*). Za wskaźnikiem występuje lista argumentów, która w tym przypadku jest pusta. Następnie ten typ wykorzystywany jest do budowy tablicy zawierającej wykaz funkcji. Tablica również zakwalifikowana została do obszaru *code*, ponieważ zawiera wartości stałe, nieulegające zmianie w czasie wykonywania programu. Rozmiar tablicy - wykazu funkcji - nie jest ustalony. Jej koniec sygnalizuje znak o kodzie „0". Dla przykładu i dla uproszczenia funkcja *command* pokazana na list. 1, zwraca wartość umieszczoną w tablicy pod indeksem 0. Zgodnie z wcześniejszą definicją odpowiada to wskazaniu adresu, pod którym umieszczona jest funkcja o nazwie *test_1*.

List. 1. Fragment programu z definicjami zmiennych - wskaźników do funkcji

```
//struktura na definicje komend
typedef struct
{
char (code *funkcja)(void);
}komendy;

//tablica z wykazem komend
code komendy wykaz[] =
{
   test_1,
   test_2,
   0
};

char command()
{
   return(wykaz[0].funkcja());
}
```

Funkcja zawarta w definicji struktury musi zwracać jakąś wartość. Najłatwiej, gdy jest to wartość typu *char*, którą można później wykorzystać do sygnalizacji np. błędów realizacji poleceń, jednak może to być również inny typ zmiennych. Jest to wymóg konieczny dla późniejszej realizacji polecenia *return(wykaz[indeks].funkcja())*, ponieważ polecenie *return* nie może zwracać wartości typu *void* oraz z powodów, o których będzie mowa dalej. Funkcja *command* powinna być tego samego typu, jak określono to w definicji struktury. Oczywiście powinna, a nie musi. Jeśli typy będą różne, to nastąpi niejawna konwersja typu zwracanej wartości.

Przyjrzyjmy się teraz programowi pokazanemu na **list. 2**. Zawiera on fragment programu odpowiadający funkcji *command* po kompilacji do postaci języka asembler. Fragment ten opatrzyłem komentarzami w taki sposób, aby osoby niemające do czynienia z asemblerem, mogły zrozumieć, jak działa mechanizm wywołujący funkcję znajdującą się pod wskazywanym adresem w pamięci programu. Kompilator wykonując konwersję wskaźnika do postaci *char*, wywołuje ukrywającą się pod wskazaniem funkcję. Jest to konieczne, aby funkcja zwróciła wartość, która będzie mogła ulec zamianie na typ *char*. A to, że funkcja umieszczona jest na liście i możliwe jest odwołanie do konkretnej pozycji tejże listy, tylko ułatwia realizację zadania postawionego jako cel artykułu - realizację implementacji interpretera poleceń.

Dla praktyków - przykład 1

Wykorzystując opisany wyżej mechanizm, wykonałem prosty interpreter poleceń odbieranych przez mikrokontroler z wykorzystaniem sprzętowego interfejsu UART. Jako układ modelowy posłużyła mi płytka prototypowa z mikrokontrolerem AT89S8252 taktowanym zegarem 7,3728 MHz. Na płytce znalazł się również układ dopasowujący stany logiczne na wyprowadzeniach UART mikrokontrolera do portu szeregowego komputera PC - standardowy MAX232. Wykonując programy przykładowe, posługiwałem się kompilatorem RC-51 firmy Raisonance.

List. 2. Polecenie *return(wykaz(0).funkcja())* po kompilacji

```
0000  900000 R  MOV DPTR,#wykaz ;do rejestru DPTR młodszy bajt adresu funkcji z
tablicy wykaz
0003  7400       MOV A,#00       ;akumulator jako wartość indeksu do tablicy - tu 0
0005  93         MOVC A,@A+DPTR  ;załaduj do akumulatora bajt spod adresu wykaz+0
0006  FA         MOV R2,A        ;przechowaj pobraną wartość w rejestrze R2
0007  900000 R  MOV DPTR,#wykaz ;ponownie do DPTR adres tablicy wykaz
000A  7401       MOV A,#1        ;ale indeks tablicy w tym przypadku to 1
000C  93         MOVC A,@A+DPTR  ;do akumulatora starszy bajt adresu funkcji
                                 ;z tablicy wykaz+1
000D  FB         MOV R3,A        ;przechowaj jego wartość w rejestrze R3
000E  8A83       MOV DPH,R2      ;przepisz R2 do rejestru DPH (starszy bajt DPTR)
0010  8B82       MOV DPL,R3      ;przepisz R3 do rejestru DPL (młodszy bajt DPTR)
0012  120000 R   LCALL ?C_INDCALL    ;wywołaj wewnętrzną procedurę RC-51 uruchamiającą
                                 ;funkcję spod adresu wskazywanego przez DPTR
0015 22          RET             ;powrót do programu głównego
```

List. 3. Deklaracje zmiennych oraz *#include* interpretera poleceń z przykładu 1

```
#pragma DEFJ(TIM1_INIT=0xFE)       //timer 1 jako prędkość transmisji (19200bps)
                    //dla rezonatora 11,0592MHz TH1=0xFD; dla 7,3728MHz TH1=0xFE
#pragma SMALL                      //wybór modelu pamięci programu
#include "reg8252.h"               //dołączenie definicji rejestrów
#include "stdio.h"                 //dołączenie funkcji wejścia - wyjścia
#define WORD unsigned int          //definicja typu WORD

//definicje nagłówków funkcji programu
char in(char data *bufor);
char out(char data *bufor);
char status(char data *bufor);
char on(char data *bufor);
char off(char data *bufor);
char help(char data *bufor);

//definicja typu dla tablicy - wykazu poleceń
typedef struct
{
char code *komenda;
char (code *funkcja)(char data *);
}komendy;

//tablica z wykazem poleceń
code komendy wykaz[] =             //wykaz komend i powiązanych z nimi funkcji
{
   "IN", in,
   "OUT", out,
   "STATUS", status,
   "ON", on,
   "OFF", off,
   "HELP", help,
   "?", help,
   "", NULL                        //koniec wykazu
};
```

Interpreter wykonuje następujące polecenia:
- IN <numer portu> np. IN 1 - odczyt portu o podanym numerze
- OUT <numer portu> <wartość> np. OUT 1 0x20 - zapisuje do portu o podanym numerze liczbę,
- STATUS - podaje informację o statusie urządzenia: WYŁĄCZONY/AKTYWNY,
- ON - załączenie operacji na portach, tj. poleceń IN i OUT,
- OFF - wyłączenie operacji na portach, tj. blokowanie funkcjonowania poleceń IN i OUT,
- HELP lub ? - informacja o obsługiwanych poleceniach.

Polecenia mogą być przesyłane przez aplikację sterującą lub podawane ręcznie za pomocą programu typu terminal znakowy. Każde wysyłane polecenie musi kończyć się sekwencją 0x0D-0x0A (CR-LF).

Na **list. 3** pokazano najważniejsze fragmenty programu źródłowego w języku C zawierające deklaracje zmiennych i stałych oraz dyrektywy sterujące kompilacją.

Początek to właściwe dla RC-51 polecenie *#pragma DEFJ(TIM1_INIT=0xFE)* definiujące wartość zapisywaną do rejestru TH1 Timera 1 sterującego pracą UART. Dla rezonatora 7,3728 MHz oraz podwójnej szybkości zegara sterującego pracą interfejsu szeregowego (SMOD = 1) zapis do TH1 wartości 0xFE wymusza transmisję szeregową asynchroniczną z prędkością 19200 bd. W następnej kolejności dołączane są definicje rejestrów mikrokontrolera, biblioteka funkcji wejścia-wyjścia oraz dla czytelności programu zdefiniowany zostaje typ WORD. W kolejnym kroku definiowane są nagłówki funkcji będących odpowiednikami poleceń realizowanych przez interpreter. Zdefiniowanie ich w tym miejscu jest konieczne, ponieważ za moment nazwy funkcji będą użyte do konstrukcji tablicy-wykazu poleceń.

Definicja struktury o nazwie *komendy* zawiera więcej składników niż we wcześniejszym przykładzie. Obok znanego nam już wskaźnika do funkcji pojawił się również wskaźnik do łańcucha znaków umieszczonego w pamięci programu mikrokontrolera. Jest to nazwa, po której rozpoznawane są polecenia i jednocześnie najprostsza z metod powiązania nazwy symbolicznej z odpowiadającą mu funkcją. Dla łatwiejszej analizy przykładu nazwy funkcji są niemal identyczne z odpowiadającymi im poleceniami.

Na **list. 4** znajduje się fragment programu rozpoznający polecenie oraz wywołujący odpowiadającą mu funkcję. Posłużyłem się w nim konstrukcją z pierwszego przykładu, z tym że polecenie *return* zawiera dynamicznie wyznaczany indeks do funkcji. Dwie zmienne *j* oraz *i* służą (odpowiednio) jako indeksy do poszczególnych liter przekazywanego jako argument funkcji ciągu znaków oraz poszczególnych linii tablicy - wykazu poleceń. Funkcja XOR (^) służy do sprawdzenia warunku równości znaków, a bitowe AND (&) maskuje bity odpowiadające małym literom alfabetu. Dzięki temu wielkość znaków (liter) odebranych z UART nie wpływa na interpretację polecenia.

W przypadku, gdy komenda nie zostanie odnaleziona w wykazie, wartość indeksu *j* będzie równa 0 i posłuży do wysłania komunikatu o błędzie. W innym przypadku zwracany jest wskaźnik do funkcji, którego przekształcenie do typu *char* owocuje wywołaniem funkcji.

Główna pętla programu zawiera tylko kilka poleceń: rezerwuje miejsce w pamięci na bufor odebranych z UART znaków, ustawia bit SMOD w rejestrze PCON mikrokontrolera, wysyła znak zachęty do terminala (znak >), wywołuje funkcję *gets* pobierającą ciąg znaków ze standardowego urządzenia wejścia - wyjścia (dla mikrokontrolera jest to UART), a następnie przekazuje wskaźnik do bufora opisywanej wyżej funkcji *command* rozpoznającej odebrane polecenia. Każdorazowo zakończenie realizacji komendy sygnalizowane jest napisem OK wysłanym przez mikrokontroler do terminala.

Dla praktyków - przykład 2

Kolejny przykład bazuje na poznanych już wcześniej. Ilustruje on jednak jeden z możliwych sposobów dołączenia zdalnego wyświetlacza sterowanego przez mikrokontroler. Dla uproszczenia funkcji sterujących użyłem wyświetlacza LCD 4 linie x 20 znaków, jednak można sobie wyobrazić zastosowanie dowolnego wyświetlacza, na przykład dołączonej do mikrokontrolera tablicy pokazującej wyniki na meczu piłkarskim, dołączonego zdalnie wyświetlacza zegara itp.

Przykładowy interpreter realizuje następujące polecenia:
- CURSOR <kod> np. CURSOR 1 - zmiana wyglądu kursora, wartość <kod> powinna się zawierać w przedziale od 0 do 2 (0 = kursor wyłączony, 1 - kursor włączony, 2 - kursor włączony i migotanie na polu kursora)
- CLR - czyszczenie ekranu LCD,
- GOTOXY <x> <y> np. GOTOXY 1 1 - umieszczenie kursora na pozycji <x> <y> (kolumna, wiersz),
- WRITE <tekst> np. WRITE Driver LCD 4x20 - wyświetlenie tekstu podanego jako parametr wywołania; uwaga: tekst nie może być dłuższy niż rozmiar bufora - 7 (w tym przypadku są to 34 znaki),
- CWRITE <kod> np. CWRITE 0x01 - wyświetlenie znaku o kodzie <kod>,
- FILL <kod> np. FILL 0x01 - wypełnienie LCD znakami o kodzie <kod>,
- DEF <kod> <bajt0>.. <bajt7> - definicja własnego znaku użytkownika np.: DEF 0x00 0xAA 0x55 0xAA 0x55 0xAA 0x55 0xAA 0x55 umieści definicję „kratki" w generatorze znaków LCD na pozycji numer 0,

List. 4. Fragment programu odpowiedzialny za rozpoznawanie odbieranych poleceń

```
//wyszukiwanie komend oraz wywołanie odpowiadających im funkcji
char command(char data *bufor)
{
   char i, j;                  //256 komend o maks. długości 256 znaków

   for (i = 0;;)
       for (j = 0;; )
       {
           if(wykaz[i].komenda[j] != 0)    //jeśli komenda różna od znaku "pustego"
           {                       //do porównania zamiana małych liter na duże
               if(((wykaz[i].komenda[j]^bufor[j]) & 0x5F) == 0)
               {
                   j++;
                   continue;      //następny znak
               }
               i++;
               break;             //następna komenda
           }
           if( j == 0 )
           {
               printf("%s\n","BLAD: Nie rozpoznano komendy!");  //brak komendy
                                  //w wykazie
               return(0);
           } else return (wykaz[i].funkcja(bufor+j));    //wykonanie funkcji spod
                                  //wskazanego adresu
   }
```

- INIT - inicjalizacja wyświetlacza w trybie interfejsu o długości słowa równej 4 bity,
- STATUS - podaje informację o statusie WYŁĄCZONY/AKTYWNY,
- ON - załączenie akceptowania poleceń przez kontroler LCD,
- OFF - wyłączenie akceptowania poleceń przez kontroler LCD,
- HELP lub ? - informacja o realizowanych poleceniach.

Program funkcjonuje identycznie jak poprzedni, różnią się one między sobą tylko liczbą realizowanych funkcji. Ten pierwszy będzie się nadawał szczególnie dobrze do sterowania urządzeń typu włącz-wyłącz, drugi realizuje także nieco bardziej zaawansowane funkcje. Do jego implementacji wykorzystałem opisywaną we wcześniejszych odcinkach kursu bibliotekę LCD4B. Została ona dołączona do pliku projektu, a nagłówki funkcji sterujących wyświetlaczem umieszczane są w programie głównym za pomocą dyrektywy *#include "lcd4b.h"*.

Program główny, oprócz omawianych wcześniej poleceń, zawiera również wywołanie funkcji inicjalizacji oraz czyszczenia ekranu modułu LCD. Użycie wymienionego na liście realizowanych poleceń *INIT* nie jest konieczne. Zostało ono wprowadzone na wypadek sytuacji awaryjnej.

Uwagi końcowe

Jak wspomniałem wcześniej, do wykonania programów demonstracyjnych posłużył kompilator firmy Raisonance RC-51. Niestety powstający w wyniku kompilacji kod przekracza 4 kB i dlatego też nie można posłużyć się wersją demonstracyjną „wprost". Bardzo dużo miejsca w pamięci programu zajmuje zwłaszcza implementacja funkcji *printf* i *scanf* wywoływanych wielokrotnie przez różne funkcje. Ta pierwsza formatuje i przesyła ciąg znaków przez UART, ta druga odbiera, odczytuje i przekształca znaki odebrane z UART na zmienne zgodnie z podanym wzorcem. Zamiast RC-51 można również posłużyć się innym kompilatorem C. W takim przypadku polecenie ustawiające prędkość transmisji UART może wyglądać jak niżej:

```
void main()
{
 SCON = 0x50;
 TMOD = 0x20;
 SMOD = 1;
 TH1 = 0xFE;
 TL1 = -1;
 TR1 = 1;
```

Polecenia wysyłane przez np. program *hyper terminal* są odbierane za pomocą zdefiniowanej przez producenta pakietu funkcji *gets()*. Wymaga ona, aby każdy przesłany ciąg znaków zakończony był przez sekwencję CR-LF. Niestety większość programów typu terminal wysyła na końcu linii CR nie dbając o LF. Tak też jest w przypadku *hyper terminala*, który to musi mieć ustawioną opcję dodawania LF na końcu linii. Należy w nim ustawić parametr *Wyślij końce wierszy ze znakiem wysuwu wiersza* (*Plik>Właściwości*, zakładka *Ustawienia>Ustawienia ASCII*). Oczywiście można również zaimplementować inną, własną funkcję działającą jak *gets()*.

Port UART mikrokontrolera pracuje z szybkością 19200 bd. Tę samą należy wybrać w nastawach portu COM komputera PC o ile to właśnie on używany jest do przesyłania poleceń (19200, n, 8, 1). Funkcja *gets()* odsyła echo wysyłanych poleceń, toteż są one widoczne na ekranie terminala.

Prezentowane wyżej przykłady programów źródłowych to tylko zachęta do samodzielnego eksperymentowania. Oczywiście mają one pewną wartość użytkową, jednak bardziej służą do wskazania możliwości niż do budowy gotowego urządzenia mającego konkretne zastosowanie.

Jacek Bogusz, EP
jacek.bogusz@ep.com.pl