Podstawy programowania komputerów

Wykład 13: „Typ wyliczeniowy – enumeration”

swalover@ie.tu.koszalin.pl

Wyliczenia

- Wyliczenie (ang. enumeration), zwane także typem wyliczeniowym, jest zbiorem symbolicznych stałych całkowitych określających wszystkie dopuszczalne wartości, jakie może przyjąć dana zmienna.
- Wyliczenia są częstym elementem programów. Na przykład wyliczenie wszystkich monet używanych w Stanach Zjednoczonych jest następujące:

  penny, nickel, dime, quarter, half-dollar, dollar

- Typ wyliczeniowy definiuje się za pomocą słowa kluczowego enum, umieszczanego na początku instrukcji. Ogólną postać:

  enum <nazwa_wyliczenia> {lista_wyliczenia} <lista_zmiennych>;

Nazwa wyliczenia i zmienne typu wyliczeniowego

- Zarówno **nazwa_wyliczenia**, jak i **lista_zmiennych** są opcjonalne, ale nie można ich pominąć jednocześnie.
- Podobnie jak w przypadku struktur, nazwa wyliczenia umożliwia deklarowanie zmiennych tego typu.

Podany niżej kod definiuje wyliczenie o nazwie „coin” i deklaruje zmienną „money”:

```
enum coin {penny, nickel, dime,
           quarter, half_dollar,dollar};
enum coin money;
```

Instrukcje na zmiennych enum

Zakładając istnienie definicji i deklaracji, można napisać następujące instrukcje

```
money = dime;
if (money == quarter)
printf ("Moneta  jest   quarter\n");
```

Ważne jest, aby pamiętać, że każdy z symboli wyliczenia oznacza określoną liczbę całkowitą, w związku z czym może być użyty w dowolnym wyrażeniu, np. instrukcja:

```
printf ("Wartość quarter jest %d",quarter);
```

jest w pełni poprawna.

Określenie wartości stałych wyliczenia

- Jeżeli wartości stałych wyliczenia nie zostały określone jawnie, to pierwsza z nich jest równa 0, druga 1 itd. Dlatego instrukcja `printf ("%d  %d", penny, dime);` wyświetla na ekranie liczby 0 2
- Istnieje możliwość zmiany wartości domyślnej jednego lub większej liczby stałych wyliczenia przez zastosowanie inicjatorów. Za nazwą stałej umieszcza się wówczas znak równości i odpowiednią liczbę całkowitą.
- Jeżeli w definicji typu wyliczeniowego zastosowano inicjator, to występującym za nim stałym przypisywane są wartości większe od wartości zainicjowanej.

Przykład zastosowania inicjatorów

- Wykonanie instrukcji:

```
enum coin {penny, nickel, dime,
       quarter=100,half_dollar, dollar};
```

spowoduje, że stałe typu coin będą mieć następujące wartości:

- `penny` 0
- `nickel` 1
- `dime` 2
- `quarter` 100

- Stosując inicjatory, można przypisać różnym elementom wyliczenia takie same wartości.

Pomyłki w operacjach wejściowych

- Podane niżej instrukcje nie są zbudowane poprawnie:

```
money = dollar;
printf("%s",  money);
```

- Należy pamiętać, że nazwa `dollar` oznacza jedynie stałą całkowitą, a nie łańcuch. Dlatego nie można przekazać funkcji `printf()` zmiennej `money` w celu wyświetlenia łańcucha `"dollar"`.
- Niedozwolony jest także taki fragment programu:

```
money = "penny";
```

Wprowadzenie i wyświetlenie łańcuchów

Aby wyświetlić nazwy monet związanych ze zmienną money, należałoby napisać następującą instrukcję złożoną:

```
switch (money){
 case penny: printf("penny");
  break;
 case nickel: printf("nickel");
  break;
 case dime: printf("dime");
  break;
 case quarter: printf("quarter") ;
  break;
 case half_dollar: printf("half_dollar");
  break;
 case dollar: printf("dollar");}
```

Wprowadzenie i wyświetlenie łańcuchów, cd.

- Można zadeklarować tablicę łańcuchów i używać wartości stałych wyliczenia jako jej indeksu w celu przekształcenia tych stałych do ich nazw.
- Podany niżej fragment programu wykonuje to samo zadanie co poprzedni, czyli wyświetla łańcuch odpowiadający aktualnej wartości zmiennej money:

```
char nazwa[ ][12] = {"penny", "nickel",
     "dime", "quarter", "half_dollar", "dollar"};

     printf("%s", nazwa[money]);
```

Wprowadzenie i wyświetlenie łańcuchów, cd.

- Instrukcje na łańcuchach działają prawidłowo tylko wtedy, gdy definicja typu coin nie zawiera inicjatorów, gdyż indeksowanie tablicy łańcuchów musi rozpoczynać się od zera.
- Instrukcje te nie są wygodne w przypadku komunikacji z konsolą w oparciu o zmienne wyliczeniowe ponieważ wartości stałych wyliczeniowych trzeba przekształcać „ręcznie" do postaci zrozumiałej przez użytkownika.
- Operacje na łańcuchach z enum są najbardziej użyteczne w sytuacjach „technicznych”, np. typy wyliczeniowe stosuje się powszechnie w kompilatorach w celu zdefiniowania tabeli symboli danego języka.

Programowanie niestrukturalne

Instrukcje skoku

swalover@ie.tu.koszalin.pl

Na czym polega programowanie strukturalne?

- Zakłada ono, że sterowanie nie może być przekazywane przy pomocy żadnych instrukcji skoku z wyjątkiem tych "zaszytych" w instrukcjach sterowania.
- Przyjęcie takiej zasady zmusza do bardziej porządnego konstruowania programu (w wielu przypadkach zapisanie algorytmu w sposób strukturalny jest trudniejsze).
- Czasami jednak podporządkowanie się tej zasadzie powoduje zbytnie zagnieżdżenie struktur programu i czyni go mniej czytelnym lub stwarza sytuację zbędnego powtarzania jakiejś sekwencji instrukcji.

Instrukcje break i continue

- W języku C najczęściej używane instrukcje niestrukturalne to break i continue. Sprowadzają się one do wykonania skoku, ale nie jest to skok do dowolnie umieszczonej etykiety lecz w konkretne miejsce, związane z pętlą, w obrębie której instrukcje te występują.
- Czytanie pliku aż do znalezienia danego znaku:

```
while((c=getc(file)!=EOF)
if(c=='$')break;
```

- Drukujemy tylko litery:

```
do{    if((c=getch())<'A')continue;
       putchar(c);
       x++;   }
while(x<10);
```

Instrukcja goto

- Można wykonać skok do miejsca oznaczonego etykietą w obrębie funkcji. Można „wskoczyć” do pętli i można z niej „wyskoczyć”.
- Wielka dowolność użycia instrukcji goto może powodować przy niefrasobliwym jej używaniu, straszliwe zagmatwanie programu. Nie jest to jednak powód, żeby całkowicie rezygnować np. z łatwego wyjścia z zagnieżdżonych pętli:

```
        for(...;...;...)
        for(...;...;...)
                {...
                if(warunek) goto end;
                }
        end: ;
/* jest to bezpośredni skok poza wszystkie pętle,
break powoduje wyjście tylko z wewnętrznej pętli */
```

Instrukcja return

- Należy zaliczyć do konstrukcji niestrukturalnych możliwość umieszczania instrukcji return w kilku różnych miejscach funkcji.
- Możliwe jest wykorzystanie instrukcji return z różnymi wartościami do zwrócenia.
- Rozwiązania te są naturalne dla funkcji, która ma zwracać różne wartości lub/i w różnych miejscach kodu, pozwala to uniknąć „sztucznego ustrukturalniania” funkcji poprzez zwracanie wartości jednej zmiennej pomocniczej.

Instrukcja switch

- Frazy case w instrukcji switch zachowują się jak etykiety skoków. Wykonanie instrukcji switch po wartościowaniu wyrażenia jest równoważne wykonaniu skoku do niejawnej etykiety związanej z jedną z fraz case (po wykonaniu skoku wszystkie inne etykiety case i default są ignorowane).
- Ponieważ frazy case można uważać za etykiety mogą one być umieszczane w dowolnym miejscu między instrukcjami.
- Jeżeli w obrębie ciała instrukcji switch zostanie napotkana instrukcja break, nie zagnieżdżona w żadnej pętli, to spowoduje ona wyjście z instrukcji switch.

Instrukcja switch, cd.

Po skoku wykonywane są kolejne instrukcje:

```
switch(getch())
        {
        case '\r' : read();             //Enter
        case '\x1b' : close_file();     //Esc
        return;
        }
```

Obsługa przypadków jest rozdzielona:

```
switch(menu())
        {
        case 1 : list(); break;
        case 2 : print(); break;
        case 3 : return;
        }
```

Instrukcja switch, cd.

Druga fraza case jest umieszczona wewnątrz instrukcji if :

```
switch(getch())
  { case '\x1b' : printf("Zapisać? ");    // Esc
                  if(getch()=='t')
                  case '\r' : save();     // Enter
    return;  }
```

Wywołanie funkcji save nastąpi, gdy użytkownik naciśnie klawisz Enter (oznaczający tu wyjście z zapisem) lub klawisz Esc (wyjście), i na pytanie "Zapisać? " odpowie twierdząco.

Instrukcja switch, cd.

Frazy case można także umieścić wewnątrz pętli:

```
switch(x)
{    default : for(x=4;x;x--)
        {       case 3 :
                case 2 :
                case 1 : /* ciało pętli */
        }
     case  0 :
     case -1 :    }
```

Jeżeli x = 1...3, nastąpi wejście do pętli for "od środka". W takim wypadku nigdy nie wykona się wyrażenie inicjujące (x=4). W efekcie, pętla wykona się x razy lub cztery razy jeżeli x jest większe niż 4. Pętla nie wykona się ani razu dla x równego -1 lub 0.

Opcja „jump optimization” kompilatora

- Kompilator (np. Borland Turbo C) może umieszczać w kodzie niejawne skoki w celu uniknięcia powtarzania jakiegoś fragmentu czyli tłumaczyć strukturalny algorytm do zwartej, ale niestrukturalnej postaci.
- Zastosowanie opcji jump optimization ma dokładnie taki sam skutek jak jawne dopisanie skoku.
- Niejawny skok dodany przez kompilator można zauważyć śledząc krokowo fragment kodu skompilowany z ustawioną opcją jump optimization.
- Dzięki zastosowaniu optymalizacji wykonywanej przez kompilator możemy mieć więc jednocześnie przejrzysty, strukturalny kod źródłowy i efektywny kod wynikowy odpowiadający konstrukcji z bezpośrednim skokiem.