Podstawy programowania komputerów

Wykład 11: „Pola bitowe. Zaawansowana reprezentacja danych”

swalover@ie.tu.koszalin.pl

Zakres wartości zmiennych

Często zmienna ma zawężony zakres wartości. Dla przykładu zmienne logiczne (tzw. flagi) to zawsze tylko 0 lub 1.

Przykładowe dane osobowe mogą być przechowywane w dwóch bajtach:
płeć (0 - mężczyzna, 1 - kobieta, 1 bit);
wiek (0 - 255 lat, 8 bitów);
ilość dzieci (0 - 15, 4 bity);
kolejny numer małżeństwa (0 - 7, 3 bity);

Sterowanie bitami

Bit jest podstawową jednostką informacji w dwójkowym systemie liczenia.

Standardowo dane są przechowywane w komputerze za pomocą słów (2, 4 bajty).

W jednym słowie można przechowywać szereg informacji na poszczególnych bitach. Jednak bity pamięci operacyjnej nie mają adresów.

Dane typu „pola bitowe”

Jest to zbiór przylegających do siebie bitów, znajdujących się w jednej jednostce pamięci zwanej słowem:

```
unsigned int odczyt :2;
unsigned int zapis :5;
```

Pole bitowe musi to być typu całkowitego int, signed int lub unsigned int.

Polu jest przydzielona liczba bitów wynikająca z deklaracji (nie więcej niż 16 w Borland).

Technika stosowania

Pola bitowe mogą się znaleźć tylko w strukturach, uniach i klasach:

```
struct dostep  {
        unsigned int odczyt     :1;
        unsigned int zapis      :1;
        unsigned int kod        :12;
        unsigned int                    :2;
};
struct dostep flag= {1,0,4095,0};
```

wyrównanie pól

Ograniczenia pól bitowych

- Nie można stosować 'sizeof' i pobierać adresu pola.
- Nie można definiować tablic na polach bitowych.
- Nie można stosować makra 'offsetof' które podaje położenie pola w strukturze.
- Nie można deklarować referencji do pól bitowych.

Operatory bitowe w C

Służą do pracy na poszczególnych bitach !!!

Operatory przesunięcia bitowego:

<< - przesunięcie w lewo,

>> - przesunięcie w prawo

Operatory logiki bitowej:

& - bitowy iloczyn logiczny

| - bitowa suma logiczna

^ - bitowa różnica symetryczna

~ - bitowa negacja

Operatory przesunięcia

Przesuwają bity operandu lewostronnego o liczbę pozycji określoną przez operand prawostronny. Zwalniane bity zostają uzupełnione zerami:

```
int wynik;
int  AA = 0x1010
wynik  = x << 2;
```

Zawartość komórek w kodzie binarnym:

AA = 0001 0000 0001 0000

wynik = 0100 0000 0100 0000

>> na liczbach ze znakiem

```
signed int f = 0xff00;
signed int wynik;
wynik = x>>2;
```

x	1111 1111
wynik	0011 1111 1100 0000
wynik	1111 1111 1100 0000

Wynik zależy od kompilatora

BorlandC

Operatory logiki bitowej

```
int x1 = 0x0f0f;     int x2 = 0x0ff0;
```

x1 0000 1111 0000 1111

x2 0000 1111 1111 0000

```
x1 & x2;  //koniunkcja
```

0000 1111 0000 0000

```
x1  |  x2;  //alternatywa
```

0000 1111 1111 1111

```
x1 ^ x2;  //różnica symetryczna
```

0000 0000 1111 1111

Przykład

```
int dostep; // zmienna globalna
modul_ustaw_bity ( ) {
      int czytaj, pisz;
      czytaj=1; pisz=1<<1;
      dostep=czytaj | pisz;  // 0001 | 0010 = 0011 }
      . . .
int akcja, maska;
      maska=1;     // zapisz
      akcja = dostep & maska; // 0010
```

Istotne możliwości bitowej reprezentacji danych

- bardziej ekonomiczne wykorzystanie pamięci;
- łatwe dodatkowe zaszyfrowanie danych;
- pełna kontrola nad danymi.

Zaawansowana reprezentacja danych

Abstrakcyjne typy danych

swalover@ie.tu.koszalin.pl

Wybór reprezentacji danych

- Przykłady programów z nie określoną ilością danych
- Efektywność wykorzystywania pamięci
- Typy danych udostępniane bezpośrednio przez język C: proste, tablicy, wskaźniki, struktury i unie.
- Kluczowy aspekt budowy programu: sposób reprezentacji danych. Określenie właściwej reprezentacji nie ogranicza się do wyboru typu. Należy zdecydować o operacjach, jakie można będzie na nich wykonywać.

Tablica struktur

- Program może tworzyć tablicę struktur, a następnie wypełniać ją danymi:

```
struct moje_dane {  char nazwa [rozmiar];
                    int numer;
. . . }  . . .
int main (void) {
                    struct moje_dane dane[max_ile_danych];
                    while(gets(dane[i].nazwa)!=NULL) {. . .}}
```

- Nie elastyczność reprezentacji danych. W czasie pisania kodu jesteś my zmuszeni do podjęcia decyzji, które powinny zostać podjęte w trakcie pracy programu.

Dynamiczna rezerwacja pamięci

- **malloc( )** pozwala odłożyć określenie liczby elementów do czasu uruchomienia programu:

```
struct moje_dane * dane;  . . .
puts(“Ile danych?”);
scanf(“%d”, &ile_danych);
dane = (struct moje_dane *)
malloc(ile_danych*sizeof(structmoje_dane));
```

- Nie ma gwarancji, że kolejne wywołania funkcji **malloc ( )** będą przydzielały sąsiadujące ze sobą bloki pamięci. Rośnie ilość wskaźników do obsługi zbioru danych.

Tablica wskaźników

- Możliwe jest tworzenie dużej tablicy wskaźników i przypisywanie im wartości w miarę tworzenia kolejnych struktur:

```
struct moje_dane * dane[max_ile_danych];
int i, n;
. . .
dane[i] = (struct moje_dane *)
malloc(n*sizeof(struct moje_dane));
```

- Tablica wskaźników pochłania dużo mniej miejsca niż tablica struktur, jednak obszar nieużywanych wskaźników jest marnowany i zostaje się ograniczenie **'max_ile_danych'**.

Listy łączone

- Przydzielając miejsce dla nowej struktury, równocześnie przydzielić miejsce dla wskaźnika do następnej struktury tegoż typu:

```
struct moje_dane { char nazwa [rozmiar];
                   int numer;
                   struct moje_dane * next_struct                    } . . .
```

Do ostatniej struktury wpisać wskaźnik **NULL**.

- Przypisać adres pierwszej struktury osobnemu wskaźniku (head pointer).

truct moje_dane *

Dane: interfejs programistyczny

- Jedna z podstawowych czynności przy pisaniu programu do sterowania dużymi obszary danych – dopasowanie typu danych do rozwiązywanego problemu. Typ zawiera dwa rodzaje informacji: zbiór własności oraz zbiór działań.

- Żeby zdefiniować nowy typ danych musimy udostępnić sposób przechowywania danych, na przykład projektując odpowiednią strukturę. Następnie musimy umożliwić operowanie danymi za pomocą prototypów funkcji.

Abstrakcyjne typy danych

- Przed opracowaniem interfejsu programistycznego należy przygotować abstrakcyjny opis własności typu i operacji, jakie można na nim wykonywać.
- Opis ten nie powinien być związany z żadną konkretną implementacją, a nawet z żadnym konkretnym językiem programowania. Formalna, wyabstrahowana charakterystyka typu nosi nazwę **Abstrakcyjnego Typu Danych**.
- Trzeci etap pracy – pisanie kodu implementującego interfejs. Napisanie programu powinno być możliwe bez orientacji w szczegółach implementacji.

Kolejka jako abstrakcyjny typ danych

Kolejka (**queue**) jest to lista łączona o dwóch szczególnych własnościach:

- nowe pozycje mogą być dodawane tylko na końcu listy;
- pozycje mogą być usuwane tylko z początku kolejki.

Kolejka jest formą danych typu FIFO:

Nazwa typu:	**kolejka**
Własności:	potrafi przechować uporządkowany ciąg danych.
Dostępne działania:	inicjalizacja, dodanie danych, pobranie danych, określenie pustej i pełnej kolejki, oraz określenie liczby danych w kolejce

Interfejs kolejki

- Definicje interfejsu można umieścić w pliku nagłówkowym **kolejka.h**
- Nowe typy tworzymy za pomocą mechanizmu C:

```
typedef struct kolejka {char Nazwa[ ]}Kolejka;
```

- Inicjalizacja kolejki:

```
void InicjujKolejke (kolejka * wsk_kol);
```

- Przykłady innych funkcji:

```
BOOLEAN PustaKolejka (const kolejka * wsk_kol);
int LiczbaPozycji (const kolejka * wsk_kol);
```

Czas w programowaniu

Przykład interfejsu programistycznego

swalover@ie.tu.koszalin.pl

Funkcje czasu w języku C

- Zegar czasu rzeczywistego i kalendarz w architekturze PC pracują niezależnie od innych elementów systemu i są one zaszyte w strukturze układu CMOS razem z pamięcią i akumulatorem podtrzymującym te dane.
- Przeniesienie bieżącej daty i czasu do systemu operacyjnego następuje podczas procedury diagnostycznej POST w czasie startu systemu.
- Dostęp do funkcji czasu w C możliwy jest poprzez podłączanie pliku nagłówkowego **time.h**

Typy zdefiniowane w pliku time.h

- **size_t** – typ całkowity, zwracany przez sizeof;
- **clock_t** – typ arytmetyczny, reprezentujący czas;
- **time_t** – typ arytmetyczny, reprezentujący czas;
- **struct** tm – typ strukturalny, przechowujący czas kalendarzowy;

Zdefiniowane są makra: **CLOK_PER_SEC** i **NULL**;

clock()/CLOK_PER_SEC – czas w sekundach.

Struktura, reprezentująca czas w C

```
struct tm { int tm_sec; // sekundy, 0-59 */
            int tm_min;   // minuty, 0-59
            int tm_hour;  // godziny, 0-23
            int tm_day;   // dzień miesiąca, 1-31
            int tm_mon;   // miesiące od stycznia, 0-11
            int tm_year;  // lata od 1900
            int tm_wday;  // dni od niedzieli, 0-6
            int tm_yday;  // dni od 1 stycznia, 0-365
            int tm_istst; // wskaźnik czasu          letniego };
```

Funkcje czasu w języku C

- Char *asctime(const struct tm *tmptr) – zwraca wskaźnik do łańcucha czasu uniwersalnego
- Clock_t clock(void) – zwraca ilość czasu procesora jaki upłynął od uruchomienia programu
- Char *ctime(const time_t *ptm) – zwraca wskaźnik do łańcucha zawierającego informacje, które odpowiadają czasowi kalendarzowemu
- Double difftime(time_t t1, time_t t2) – zwraca liczbę sekund dzielącą t2 od t1

Funkcje czasu c.d.

- **ftime( )** – zwraca czas jaki upłynął od 1 stycznia 1970 roku, godz. 0:00 czasu uniwersalnego (GMT) oraz czy aktualnie obowiązuje czas letni
- **gmtime( )** – zwraca wskaźnik do czasu uniwersalnego wieloczłonowego (tm)
- **localtime( )** – zwraca wskaźnik do czasu lokalnego wieloczłonowego (tm)

Funkcje czasu c.d.

- **mktime( )** – dostosowuje zawartość pól struktury tm, (zwraca ona czas kalendarzowy odpowiadający pobranemu czasowi wieloczłonowemu)
- **time( )** – zwraca aktualny czas kalendarzowy systemu
- **strftime( )** – przekształca czas wieloczłonowy zapisany w strukturze wskazywanej przez time* i zapisuje go w tablicy znaków wskazywanej przez str.

Specyfikatory formatu strftime( )

%a, A – skrót i pełna nazwa dnia tygodnia

%b, B – miesiąc

%c – data i czas

%d – dzień miesiąca: dwie cyfry

%H, I – godzina w skali 24 lub 12

%m, M, S – miesiąc, minuta i sekunda cyfrą

%x, X – data i czas

%y, Y – rok dwucyfrowy lub z wiekiem

Część specyfikatorów zalezy od lokalizacji!!!