Instrukcja do zajęć laboratoryjnych

Język ANSI C (w systemie LINUX)

wersja: 1.21

Nr ćwiczenia:	6, 7	
Temat:	**Odwzorowanie bitowe zbioru liczb**	
Cel ćwiczenia:	Celem ćwiczenia jest napisanie programu umożliwiającego zapis i odczyt danych całkowitych do / z tablicy w postaci tzw. odwzorowania bitowego.	
Wymagane przygotowanie teoretyczne:	Informacje podane na wykładzie. Rozdział 2 w książce K&R „Język ANSI C" (operatory bitowe)	
Sposób zaliczenia:	Sprawozdanie w formie pisemnej. Pozytywna ocena ćwiczenia przez prowadzącego pod koniec zajęć.	[**X**] []

1. Opis zadania

Zadanie polega na napisaniu programu umożliwiającego wprowadzanie i odczytywaniu do / z tablicy jednowymiarowej podanego zbioru liczb całkowitych. Dla ułatwienia na początek zakładamy, że dane podawane są z klawiatury. Po uruchomieniu i przetestowaniu takiej uproszczonej wersji programu, powinien zostać on rozbudowany o możliwość zapisywania oraz odczytywania danych z plików dyskowych.

Zapis i odczyt danych zorganizowany jest w postaci tzw. odwzorowania bitowego. Zakładamy również, że zbiór liczb jest unikalny (dla urealnienia zadania niech będą to np. numery telefonów w jednej strefie numeracyjnej).

2. Wskazówki i uwagi

- w zadaniu chodzi o przechowywanie zbioru np. 10.000 liczb całkowitych z zakresu `<1, 10000>` a nie 10.000 dowolnych (dowolnie dużych) liczb całkowitych. Zakładamy również, że nie wszystkie liczby z podanego zakresu występują w zbiorze. Bez ostatniego założenia zadanie nie miałoby oczywiście sensu.

- chcąc przechować 10.000 kolejnych liczb całkowitych (zakładamy, że pracujemy w środowisku **32. bitowym** gdzie liczby typu `int` są **4. bajtowe**), potrzebować będziemy 40.000 bajtów

pamięci.
Możemy jednak odwzorować nasz zbiór liczb jako ciąg 10.000 **bitów**, gdzie *i*-ty bit jest jedynką wtedy i tylko wtedy, gdy liczba *i* znajduje się w zbiorze. Przykładowo układ bitów w tablicy:

```
-- środowisko jest 32. bitowe, więc: (10000/32=312.5)
-- po zaokrągleniu w górę daje 313
int Tab[313];
```

do której „zapisaliśmy" następujący zbiór liczb:

```
{2, 5, 12, 9999, 10000}
```

wygląda następująco:

```
[01001000 00010000 00000000 00000000 ... 00000011 00000000 00000000]
```

Powyżej pokazano tylko część z dokładnie 10.016 **bitów** tablicy `int Tab[313]`. Zauważmy, że z powodu wprowadzenia niezbędnych zaokrągleń (liczba 312.5 została zaokrąglona do 313) nasz system będzie służył do przechowywania 10.016 liczb zamiast zakładanych 10.000. Oczywiście gdyby zaokrąglić w dół, do liczby 312, można by zapamiętać tylko 9984 liczb. Zakładamy ponadto, że bity liczymy od jeden (nie od zera).

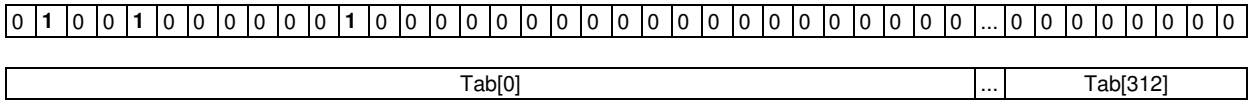
Uwaga: rozważ również inne sposoby deklaracji odpowiedniej tablicy np:

```
// ta wersja jest bardziej ogólna, gdyż będzie zadziała
// w środowisku zarówno 32. bitowym jak i 16. bitowym
#define X 10000 / sizeof(int)*8
int Tab[X];
// lub
char Tab[10000 / 8]; // typ char jest zawsze 8. bitowy
// lub
#include <limits.h>
char Tab[10000 / CHAR_BIT];
```

Swój wybór uzasadnij. W programie staraj się tam gdzie jest to tylko możliwe pisać kod, który nie będzie zależny od platformy systemowej.

- w programie należy zaimplementować zarówno operacje zapisu jak i odczytu liczb. Przykładowo zapis do tablicy liczby 9999 ustawia 9999-ty bit w tablicy `Tab[]`, odczyt liczby 9999 polega na sprawdzeniu, czy 9999-ty bit tablicy `Tab[]` jest ustawiony.

- powyższe rozważania ilustruje też następujący rysunek, pokazujący zawartość bitową pierwszego elementu tablicy czyli element `Tab[0]` oraz ostatnie 8 bitów ostatniego elementu tablicy, czyli elementu `Tab[312]`.



- w rzeczywistości tak duży zbiór danych byłby z pewnością odczytywany z pliku dyskowego. Opracuj również wersję „plikową" programu. W pierwszej wersji dane do pobrania niech znajdują się w pliku tekstowym (chodzi o format tekstowy a nie tekstowy tryb otwarcia pliku – to są dwa zupełnie różne zagadnienia!). Po zakończeniu pracy programu, dane z tablicy (znajdującej się oczywiście w pamięci operacyjnej komputera) powinny również zostać zapisane do pliku tekstowego. Zastanów się nad rozmiarem pliku przeznaczonym do przechowywania zbioru 10.000 całkowitych liczb 4. bajtowych w postaci tekstowej. Przedyskutuj ew. korzyści wyboru formatu binarnego zapisu danych w pliku (chodzi tu o binarny format zapisu danych na pliku) dla przechowywania zbioru 1000 liczb, 10.000 liczb,

1.000.000 liczb. Dyskusję zamieść w sprawozdaniu. We własnym programie zastosuj najlepszy według ciebie sposób zapisu / odczytu na / z dysku.

- zastanów się również nad problemem zapamiętywania liczb powtarzających się. Możesz założyć, że liczby mogą powtarzać się tylko określoną liczbę razy np. 7 razy. Zastanów się, czy dozwolona liczba powtórzeń podlega jakimś ograniczeniom. Odpowiedź uzasadnij na podstawie własnej propozycji rozwiązania problemu zapamiętywania powtarzających się liczb. Dyskusję zamieść w sprawozdaniu.

- od początku dziel program na mniejsze części (funkcje). Program zawierający jedynie funkcję main() nie będzie oceniany.

3. Przykład działania

Przykładowe uruchomienie programu (w wersji „bezplikowej" nie będą występować dwie pierwsze opcje). W przykładzie nie obsługuje się występowania powtarzających się liczb.

```
Wybierz tryb pracy:
[L] - ładowanie danych z pliku
[Z] - zapis danych na plik
[W] - wstawianie liczby do tablicy
[O] - odczytywanie liczby z tablicy
[U] - usuwanie liczby z tablicy
[A] - automatyczne odczytanie zawartości tablicy
[Q] - wyjście z programu

Twój wybór:  A
  W bazie nie ma żadnej liczby

Twój wybór:  W
Podaj liczbę do wstawienia: 5
  OK.

Twój wybór:  W
Podaj liczbę do wstawienia: 5
  BŁĄD: liczba 5  jest już zarejestrowana

Twój wybór:  W
Podaj liczbę do wstawienia: 2
  OK.

Twój wybór:  K
  BŁĄD: nieznane polecenie

Twój wybór:  O
Jakiej liczby szukasz: 4
  Brak szukanej liczby

Twój wybór:  O
Jakiej liczby szukasz: 5
   Liczba jest zarejestrowana

Twój wybór:  A
  W bazie znajdują się następujące liczby: 2, 5

Twój wybór:  W
Podaj liczbę do wstawienia: 10
  OK.
```

```
Twój wybór:  Z
  Zapisano liczby do pliku.
  W pliku znajdują się liczby: 2, 5, 10

Twój wybór:  L
  Załadowano liczby z pliku.
  W pliku znajdują się liczby: 2, 5, 10

Twój wybór:  U
Jaką liczbę usunąć: 2
  OK.

Twój wybór:  U
Jaką liczbę usunąć: 7
  BŁĄD: brak szukanej liczby 7

Twój wybór:  Z
  Zapisano liczby do pliku.
  W pliku znajdują się liczby: 5, 10

Twój wybór:  Q
  Do zobaczenia!
```

4. Sprawozdanie

Sprawozdanie powinno zawierać następujące elementy:

- wydrukowany kod źródłowy programu (do wydruku użyć koniecznie takiej czcionki: `Courier New 8pt`). Plik źródłowy musi być **rozsądnie skomentowany** oraz **czytelnie sformatowany** (możesz użyć jakiegoś programu narzędziowego, który pomoże ładnie sformatować tekst źródłowy). Nieczytelny kod nie będzie sprawdzany !
- plik *makefile* do kompilacji programu,
- dyskusję rozwiązania i rzeczowe uwagi autora programu.

5. Dodatek 1

W trakcie pisania programu może przydać się analiza poniższych trzech, krótkich programów służących do wyświetlania podanej liczby typu `char` lub `int` w formacie dwójkowym. Pierwszy z nich był omawiany na wykładzie, drugi i trzeci jest ulepszoną wersją pierwszego.

```
// num2bin.c
// ver. 1
// converts char number to its binary representation
#include <stdio.h>
#include <limits.h>               // for CHAR_BIT

void main(void)
  {
  unsigned char j, bit;
  char num;
  unsigned char mask;             // musk MUST be unsigned

  while(num != 0)                 // terminate on 0
```

```
    {
    mask = 0x80;                    // 1000 0000 binary
    printf("\nEnter number: ");
    scanf("%d", &num);
    printf("Binary of %d is:  ", num);
    for(j=0; j < CHAR_BIT; j++)    // for each bit
      {
      bit = (mask & num) ? 1 : 0; // bit is 1 or 0
      printf("%d ", bit);          // print bit
      if(j==3)                     // print dash between
      printf("-- ");               // each four bits
      mask >>= 1;                  // shift mask to right
      }  // end for
    }    // end while
  }      // end main()
```

```
// num2bin.c
// ver. 2
// converts int number to its binary representation
#include <stdio.h>
#define INT_BITS sizeof(int)*8
void main(void)
  {
  unsigned char j, bit;
  int num;
  unsigned int mask;                // musk MUST be unsigned

  printf("\n%o \n%x \n%o \n%x \n%x \n%x \n%x \n",
  ~077, ~077, ~0, ~0, 0, ~0<<31, ~(~0<<31));
  // koniecznie przeanalizuj !
  // gdy wynik działania powyższej funkcji printf będzie dla
  // ciebie niezrozumiały, dalsza analiza tego
  // programu (i następnego) nie ma sensu !

  while(num != 0)
    {
    //mask = 0x80000000;           // gdy int 32. bitowy
    //mask = 0x80000;              // gdy int 16. bitowy
    mask = ~0 << INT_BITS-1;       // lepiej, gdyż ~0 NIE zależy
                                   // od długości typu!
  printf("\nEnter number: ");
    scanf("%d", &num);
    printf("Binary of %d is\n:  ", num);
    for(j=0; j < INT_BITS; j++)
      {
      bit = (num & mask) ? 1 : 0;  // bit is 1 or 0
      printf("%d ", bit);          // print bit
      if( (j+1)%8 == 0)            // print dash between
      printf("-- ");               // bytes
      mask >>= 1;
      }
    }
  }
```

```
// num2bin.c
// ver. 3
// converts int number to its binary representation
#include <stdio.h>
#define INT_BITS sizeof(int)*8
void main(void)
  {
  unsigned char j, bit;
  int num;
  // w tej wersji jawnie zadeklarowana maska i (jawne)
  // przesuwanie w niej bitów nie jest już potrzebne

  while(num != 0)
    {
    printf("\nEnter number: ");
    scanf("%d", &num);
    printf("Binary of %d is\n:  ", num);
    for(j=0; j < INT_BITS; j++)
      {
      bit = (num & (unsigned int)(~0 << INT_BITS - 1) >> j) ? 1 : 0;
      // dość złożone wyrażenie, ale poprawne
      // zwróć uwagę, że jakiekolwiek dodatkowe nawiasy są tu zbędne !
      // (przeanalizuj priorytet i łączność użytych operatorów)
      printf("%d ", bit);
      if((j+1)%8 == 0)
      printf("-- ");
      }
    }
  }
```

W ostatniej wersji programu `num2bin.c` zastosowano dosyć złożone wyrażenie (jest tam aż osiem operatorów !). Tak złożonych konstrukcji należy raczej unikać, gdyż są one bardzo podatne na błędy, a podany przykład traktować jako ciekawostkę programistyczną.

Zauważmy, że bez rzutowania `(unsigned int)`, program będzie działał, jednak wyniki pojawiające się na ekranie będą błędne. Rzutowanie to „przygotowuje" poprawny argument dla operatora przesunięcia `>>`. Operator ten będzie prawidłowo przesuwał bity maski w prawo tylko dla argumentów BEZ znaku. Zgodnie z dokumentacją języka C, działanie tego operatora dla argumentów ZE znakiem jest nieokreślone i zależy od implementacji (kompilatora). Poniższy program pozwala zaobserwować błąd w procesie przesuwania bitów maski. Bez trudu zauważamy, że przesuwając bitowo liczbę ZE znakiem w prawo, następuje (tu zbędne) powielanie bitu znaku.

```
// num2bin.c
// ver. 4
#include <stdio.h>
#define INT_BITS sizeof(int)*8

void main(void)
{
unsigned char j;
  for(j=0; j < INT_BITS; j++)
    {
    printf("j: %2d %x ", j, ~0 << INT_BITS - 1 >> j);
```

```
    printf("%8x\n", (unsigned int)(~0 << INT_BITS - 1) >> j);
    }
}

// wynik działania programu
j:  0 80000000 80000000
j:  1 c0000000 40000000
j:  2 e0000000 20000000
j:  3 f0000000 10000000
j:  4 f8000000  8000000
j:  5 fc000000  4000000
j:  6 fe000000  2000000
j:  7 ff000000  1000000
j:  8 ff800000   800000
j:  9 ffc00000   400000
j: 10 ffe00000   200000
j: 11 fff00000   100000
j: 12 fff80000    80000
j: 13 fffc0000    40000
j: 14 fffe0000    20000
j: 15 ffff0000    10000
j: 16 ffff8000     8000
j: 17 ffffc000     4000
j: 18 ffffe000     2000
j: 19 fffff000     1000
j: 20 fffff800      800
j: 21 fffffc00      400
j: 22 fffffe00      200
j: 23 ffffff00      100
j: 24 ffffff80       80
j: 25 ffffffc0       40
j: 26 ffffffe0       20
j: 27 fffffff0       10
j: 28 fffffff8        8
j: 29 fffffffc        4
j: 30 fffffffe        2
j: 31 ffffffff        1
```

6. Dodatek 2

Kody źródłowe powinny być zawsze jednolicie sformatowane, gdyż przede wszystkim poprawia to ich czytelność. Ręczne dbanie o właściwe formatowanie może być jednak dość uciążliwe. Dlatego też stworzono program o nazwie `indent`, który szybko i elegancko wykonuje za nas całą „czarną robotę". Ma on bardzo wiele opcji (patrz dokumentacja man), jednak w praktyce zwykle wystarcza nam tylko kilka z nich. Zachęcamy do zapoznania się z dokumentacją.

Poniżej zamieszczono plik źródłowy w języku C przed formatowaniem oraz wyniki jakie uzyskano po „przepuszczeniu" tego pliku przez program `indent`. Wersję pierwotną celowo bardzo „rozformatowano", aby pokazać możliwości omawianego programu. Postaraj się samodzielnie zorientować jaki efekt końcowy dają poszczególne przełączniki.

```
/* indent_bad.c             */
/* Pierwotny plik źródłowy */
/* Całkowity chaos!         */

#include    <stdio.h>
#include<stdio.h>
int main   (void)
{
int i, sum=0;
int count= 0;
    int total  =    0;
char ch;

for (i=0;i<3;i++)  {
printf ("Hello world\n"       );
      sum+=i;}
while (count<10     ){
total +=     count ;
printf(    "\ncount=%d,total=%d",count++,total);
}

printf  ("\nType a character: ");
ch=getchar();
if ( ch ==        'y'    )
{
printf   ("\nYou typed y.");printf("\nNot some other character.\n");
}     return 0;   }
```

```
/* indent -kr -i2 -ts2 indent_bad.c -o indent_kr.c */

#include <stdio.h>
#include <stdio.h>
int main(void)
{
   int i, sum = 0;
   int count = 0;
   int total = 0;
   char ch;

   for (i = 0; i < 3; i++) {
      printf("Hello world\n");
      sum += i;
   }


   while (count < 10) {
      total += count;
      printf("\ncount=%d,total=%d", count++, total);
   }

   printf("\nType a character: ");
   ch = getchar();
   if (ch == 'y') {
      printf("\nYou typed y.");
      printf("\nNot some other character.\n");
   }

   return 0;
}
```

```
/* indent -gnu -i2 -ts2 indent_bad.c -o indent_gnu.c */

#include <stdio.h>
#include <stdio.h>
int
main (void)
{
   int i, sum = 0;
   int count = 0;
   int total = 0;
   char ch;

   for (i = 0; i < 3; i++)
      {
         printf ("Hello world\n");
         sum += i;
      }


   while (count < 10)
      {
         total += count;
         printf ("\ncount=%d,total=%d", count++, total);
      }

   printf ("\nType a character: ");
   ch = getchar ();
   if (ch == 'y')
      {
         printf ("\nYou typed y.");
         printf ("\nNot some other character.\n");
      }

   return 0;
}
```

```
/* indent -gnu -i0 -ts2 indent_bad.c -o indent_gnu.c */

#include <stdio.h>
#include <stdio.h>
int
main (void)
{
int i, sum = 0;
int count = 0;
int total = 0;
char ch;

for (i = 0; i < 3; i++)
   {
   printf ("Hello world\n");
   sum += i;
   }


while (count < 10)
   {
   total += count;
   printf ("\ncount=%d,total=%d", count++, total);
   }

printf ("\nType a character: ");
ch = getchar ();
if (ch == 'y')
   {
   printf ("\nYou typed y.");
   printf ("\nNot some other character.\n");
   }

return 0;
}
```