Множество

Множество е група от елементи от тип T, над които се изпълняват операциите:

- **принадлежност към множество;**
- **сечение на множества;**
- **обединение на множества;**
- **разлика на множества.**

Примери:

S = {1, 4, 8, 9}
T = {0, 1}

Приложение

– сканираща програма на компилатор, която преобразува последователността от символи в символи на езика, т.е. идентификатор, число и т.н.;

– разписание.

Логическо описание

Съединение на елементи от тип T, образуващи тяло на множеството.

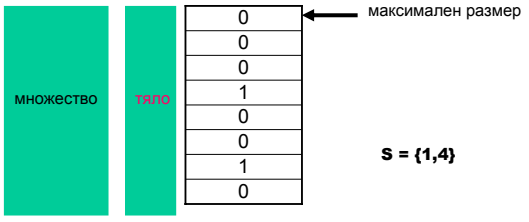
данни: тип T
тип МНОЖЕСТВО$_T$ = (ПРАЗНО | НЕПРАЗНО_МНОЖЕСТВО$_T$)
тип НЕПРАЗНО_МНОЖЕСТВО$_T$ = (тяло: МНОЖЕСТВО$_T$)

Операции

1. **Създаване на множество**
2. **Празно множество**
3. **Конструиране на множество**
4. **Принадлежност към множество**
5. **Обединение на множества**
6. **Сечение на множества**
7. **Разлика на множества**

Физическо представяне

Представяне чрез характеристична функция – масив от логически стойности, чиито – i-ти компоненти определят наличие или отсъствие на i-тата стойност в множеството – низ от битове.



тяло – **фиксирана област от паметта (масив)**

Дефиниране

```
#define SIZE sizeof(int)*8  // максимален размер
struct set                  // множество
{
  int bits;                 // тяло
};
typedef struct set SET;
```

int е последователност от 32 бита, като всеки бит може да има стойност 1 или 0.

1. Създаване на празно множество

Алгоритъм

тяло ← **ПРАЗНО**

```
SET create()
{
  SET s;
  s.bits = 0;          // всички елементи показват
  return s;            // отсъствие на стойност
}                      // в множеството
```

2. Празно множество

Алгоритъм

ако тяло **= ПРАЗНО**
  множеството е празно
в противен случай
  множеството не е празно

```
int empty (SET s)
{
  return !s.bits;
}
```

3. Конструиране на множество – чрез елементи на масив

Алгоритъм

j ← 0
за i←0 до i<SIZE стъпка 1
повтаряй
  ако i = масив[j] и j<размер$_{масив}$
    бит$_{тяло[i]}$ ← 1
    j ← j + 1
в противен случай
  бит$_{тяло[i]}$ ← 0

бит$_{тяло[i]}$ ← 1

```
s->bits |= (1<<i);
```

бит$_{тяло[2]}$ ← 1

1	00000001
1<<2	00000100
s->bits	01000000
s->bits |= (1<<2)	01000100

бит$_{тяло[i]}$ ← 0

```
s->bits &= ~(1<<i);
```

бит$_{тяло[2]}$ ← 0

1	00000001
1<<2	00000100
~(1<<2)	11111011
s->bits	01000100
s->bits &= ~(1<<i);	01000000

Проверка дали бит$_{тяло[i]}$ е установен в 0 или 1

s->bits & (1<<i)) != 0

1	00000001	1	00000001
1<<2	00000100	1<<2	00000100
s->bits	01000100	s->bits	01000000
s->bits & (1<<2)	00000100	s->bits & (1<<2)	00000000

множество — масив за инициализация — брой на елементите в масива

```
void init (SET *s, int a[], int n)
{
  int i, j;
  j = 0;
  for (i=0; i<SIZE; i++)
    if (i == a[j] && j < n)
    {
        s->bits |= (1<<i);
        j++;
    }
    else
        s->bits &= ~(1<<i);
}
SET s1;
int a[4] = {1, 4, 8, 9};
init (&s1, a, sizeof(a));
```

установяване на i-тия бит в 1

установяване на i-тия бит в 0

4. Принадлежност към множество $i \in S$

Алгоритъм

ако бит$_{тяло[i]}$ = 1
  елементът принадлежи към множеството
в противен случай
  елементът не принадлежи към множеството

множество — елемент от базовия тип на множеството

```
int in (SET s, int i)
{
        return (s.bits & (1<<i)) != 0;
}
```

5. Обединение на множества $s_1 \cup s_2$

Алгоритъм

$$s_1 \cup s_2 = \{i \mid i \in s_1 \text{ или } i \in s_2\}$$

множество$_1$ множество$_2$

```
SET uni (SET s1, SET s2)
{
  SET s3;
  s3.bits = s1.bits | s2.bits;
  return s3;
}
```

6. Сечение на множества $s_1 \cap s_2$

Алгоритъм

$$s_1 \cap s_2 = \{i \mid i \in s_1 \ u \ i \in s_2\}$$

множество$_1$ множество$_2$

```
SET intersection (SET s1, SET s2)
{
  SET s3;
  s3.bits = s1.bits & s2.bits;
  return s3;
}
```

7. Разлика на множества $s_1 - s_2$

Алгоритъм

$$s_1 - s_2 = \{i \mid i \in s_1 \ u \ i \notin s_2\}$$

множество$_1$ множество$_2$

```
SET difference (SET s1, SET s2)
{
  SET s3;
  s3.bits = s1.bits & ~s2.bits;
  return s3;
}
```

Задача: Структура от данни МНОЖЕСТВО

Разписание за провеждане на срещи

Във фирма работят 25 служащи, управител и президент. Нека работният ден е от 9 часа. Използвайте множество, което съдържа часовете от работния ден, заети с провеждане на срещи.

1. **Декларирайте множество** full**, което е инициализирано с часове за провеждане на срещи от {1÷9}.**
2. **Декларирайте множество** ceo**, което съдържа назначените срещи на управителя на фирмата съответно {1, 3÷5}.**
3. **Декларирайте множество** vip**, което съдържа назначените срещи на президента на фирмата съответно {3, 6}.**

4. Намерете множество possible**, което съдържа възможните часове за среща между управителя и президента на фирмата.**

Алгоритъм

множество от свободните часове на управителя

$free \leftarrow full - ceo$

множество от възможните часове за среща между управителя и президента

$possible \leftarrow free \cap (full - vip)$

5. Декларирайте масив employee**, на който всеки елемент съдържа множество от заетите със срещи часове на служещите във фирмата. Нека първият служител има уговорена среща {2}, последният – {8}, а всички останали служителите нямат назначени срещи.**

6. В множеството possible **намерете възможните часове за среща, когато президентът, управителят и всички служители са свободни.**

Алгоритъм

за всеки служител

множество от свободните часове за среща между управител, президент и служител

$possible \leftarrow possible \cap (full - employee_i)$

7. Напишете функция print, която отпечатва възможните часове за среща, когато президентът, управителят и всички служители са свободни.

Алгоритъм

```
ако possible = ∅
  отпечатай: Не може да се проведе среща
в противен случай
  отпечатай: Възможни часове за среща
  за час ← 1 до 9 стъпка 1
  повтаряй
    ако час ∈ possible
      отпечатай: час
```

8. Отпечатайте възможните часове за среща, като използвате функцията print.