```
módulo conjuntos
exporta

  tipo conjcar {conjunto de caracteres}

  procedimiento vacío(sal A:conjcar)
  {devuelve en A el conjunto vacío}

  función esVacío(A:conjcar) devuelve bool
  {devuelve verdad si y sólo si A es vacío}

  procedimiento poner(ent c:carácter; e/s A:conjcar)
  {Añade el carácter c al conjunto A, si no estaba ya}

  procedimiento quitar(ent c:carácter; e/s A:conjcar)
  {quita el carácter c del conjunto A, si está}

  función pertenece(c:carácter; A:conjcar) devuelve bool
  {devuelve verdad si y sólo si c está en A}

  procedimiento unión(ent A,B:conjcar; sal C:conjcar)
  {devuelve en C la unión de A y B}

  procedimiento intersección(ent A,B:conjcar; sal C:conjcar)
  {devuelve en C la intersección de A y B}

  función cardinal(A:conjcar) devuelve natural
  {devuelve el cardinal del conjunto}

  {Las tres siguientes operaciones implementan un Iterador.
   Hablaremos de ello más adelante}

  procedimiento iniciarIterador(e/s A:conjcar)
  {Prepara el iterador para que el siguiente elemento a visitar sea el primer carácter
   de A por visitar, si existe (situación de no haber visitado ningún carácter)}

  función existeSiguiente(A:conjcar) devuelve booleano
  {Devuelve falso si ya se han visitado todos los caracteres de A.
   Devuelve verdad en caso contrario.}

  procedimiento siguiente(e/s A:conjcar; sal c:carácter; sal error:booleano)
  {Implementa las operaciones "siguiente" y "avanza" de la especificación, es decir:
   Si existeSiguiente(A), error toma el valor falso, c toma el valor del siguiente
   carácter del conjunto, y se avanza el iterador al carácter siguiente del conjunto.
   Si no existeSiguiente(A), error toma el valor verdad, c queda indefinido y A
   queda como estaba.}

implementación
  tipo conjcar = registro
                   elmto:vector[carácter] de bool;
                   card:natural;
                   iterPos:0..256 {para implementar el iterador;
                                   el valor 256 representa que no existeSiguiente}
                 freg

  procedimiento vacío(sal A:conjcar)
  variable n:0..255
  principio
    A.card:=0;
    para n:=0 hasta 255 hacer
      A.elmto[chr(n)]:=falso
    fpara
  fin
```

```
función esVacío(A:conjcar) devuelve bool
principio
  devuelve A.card=0
fin

función pertenece(c:carácter; A:conjcar) devuelve bool
principio
  devuelve A.elmto[c]
fin

procedimiento poner(ent c:carácter; e/s A:conjcar)
principio
  si not pertenece(c,A) entonces
    A.elmto[c]:=verdad;
    A.card:=A.card+1
  fsi
fin

función cardinal(A:conjcar) devuelve natural
principio
  devuelve A.card
fin

procedimiento quitar(ent c:carácter; e/s A:conjcar)
principio
  si pertenece(c,A) entonces
    A.elmto[c]:=falso;
    A.card:=A.card-1
  fsi
fin

procedimiento unión(ent A,B:conjcar; sal C:conjcar)
variable n:0..255
principio
  C.card:=0;
  para n:=0 hasta 255 hacer
    C.elmto[chr(n)]:=A.elmto[chr(n)] or B.elmto[chr(n)];
    si C.elmto[chr(n)] entonces
      C.card:=C.card+1
    fsi
  fpara
fin

procedimiento intersección(ent A,B:conjcar; sal C:conjcar)
variable n:0..255
principio
  C.card:=0;
  para n:=0 hasta 255 hacer
    C.elmto[chr(n)]:=A.elmto[chr(n)] and B.elmto[chr(n)];
    si C.elmto[chr(n)] entonces
      C.card:=C.card+1
    fsi
  fpara
fin

procedimiento iniciarIterador(e/s A:conjcar)
principio
  A.iterPos:=0;
  mientrasQue A.iterPos≤255 andthen not A.elmto[chr(A.iterPos)] hacer
    A.iterPos:=A.iterPos+1
  fmq
fin
```

```
  función existeSiguiente(A:conjcar) devuelve booleano
  principio
    devuelve A.iterPos<256
  fin

  procedimiento siguiente(e/s A:conjcar; sal c:carácter; sal error:booleano)
  principio
    si existeSiguiente(A) entonces
      c:=chr(A.iterPos);
      A.iterPos:=A.iterpos+1;
      mientrasQue A.iterPos≤255 andthen not A.elmto[chr(A.iterPos)] hacer
        A.iterPos:=A.iterPos+1
      fmq;
      error:=falso
    sino
      error:=verdad
    fsi
  fin

fin
```