Estructura de Datos y Algoritmos

UNIONES en C/C++

Una **union** es un tipo de datos derivado, como una estructura, con miembros que comparten el mismo espacio de almacenamiento.

Una variable de tipo union puede contener (en momentos diferentes) objetos de diferentes tipos y tamaños.

Las uniones proporcionan una forma de manipular diferentes clases de datos dentro de una sola área de almacenamiento.

En cualquier momento una union puede contener un máximo de un objeto debido a que los miembros de una union comparten el espacio de almacenamiento.

Una union se declara con el mismo formato de una struct. Primero declaramos el tipo union y luego declaramos variables de ese tipo. Por ejemplo:

```
typedef char str[10];

union tipo_union{
        int ival;
        float fval;
        str sval;
};
```

declara el tipo union y declaramos una variable del tipo union o puntero a union con:

```
tipo_union u, *v;
```

La variable será lo suficientemente grande como para mantener el mayor de los tres tipos. Valores de cualquiera de esos tipos pueden ser asignados a u y despues empleado en expresiones mientras que el uso sea consistente: el tipo recuperado debe ser el tipo que se almacenó más recientemente. Es responsabilidad del programador llevar el registro del tipo que está almacenado actualmente en una union.

El resultado de referenciar un miembro de union diferente al último que se guardó es indefinido, ya que el dato almacenado se trata como de tipo diferente.

Tenemos acceso a los miembros de una union con:

```
nombre_union.miembro
```

por ejemplo

```
u.ival
```

o con

```
apuntador_union->miembro
```

por ejemplo

v->fval

Supongamos tenemos una variable tipo_de_u que lleva el registro del tipo actualmente almacenado en u, entonces podemos definir:

```
if (tipo_de_u==INT) cout << u.ival;
else if (tipo_de_u==FLOAT) cout << u.fval;
     else if (tipo_de_u==STRING) cout << u.sval;
```

Las uniones pueden presentarse dentro de estructuras y arreglos y viceversa.

Las operaciones que se pueden realizar sobre uniones son las siguientes: asignar una union a otra del mismo tipo, tomar la dirección de un tipo union, acceder a los miembros de una union utilizando el operador de miembro de estructura (.) o el operador de apuntador a estructura (&).

No podemos comparar uniones debido a que no se sabe cual miembro de cada una está activo.

Una union puede inicializarse con un valor del mismo tipo que el primer miembro de la union.

A continuación veremos programas que utilizan uniones:

```
union Number {
        int x;
        float y;};


int main()
{
  Number value;

  value.x=100;
  cout << "Imprimo el valor del miembro definido (x)"<< value.x << "\n";
  value.y=10.5;
  cout << "Imprimo el valor del miembro definido (y)"<< value.y << "\n";
  return 0;
}
```

Una función en la cual imprimimos dependiendo de que valor está definido es

```
//tipo=0 indica el elemento definido es el entero
//tipo≠0 indica el elemento definido es el real
void imprimo_union (Number v,int tipo)
{     if (tipo==0)
         cout << "entero "<< v.x << "\n";
      else
         cout << "real "<< v.y << "\n";
}
```

Un ejemplo de como usamos apuntadores a uniones es:

```
int main()
{
    Number i1,*i2;

    i2=&i1;
    i2->x=6;
    cout << i1.x << "\n";
    imprimo_union(*i2,0);
    imprimo_union(i1,0);
    i2->y = 2.5;
    imprimo_union(i1,1);
    system("PAUSE");

    return 0;
}
```

imprime:

```
6
entero 6
entero 6
real 2.5
```

otra posibilidad es usar new:

```
int main()
{
    Number i1,*i2;

    i2=new Number;
    i2->x=6;
    i1=*i2;
    imprimo_union(*i2,0);
    imprimo_union(i1,0);
    i2->y = 2.5;
    i1=*i2;
    imprimo_union(i1,1);
    system("PAUSE");

    return 0;
}
```

imprime:

```
entero 6
entero 6 real 2.5
```