Estructura de Datos y Algoritmos

Modificación de Estructuras

1. Modificando arrays

Hemos visto llamadas de funciones por referencia. En el caso particular de arrays no es necesario llamar por referencia, alcanza con modificar el array dentro de la función, por ejemplo:

```
void cargo_array (int A [], int n)
{
      for (int i=0;i < n; i++)
          A[i]=i;
}
```

em main defino:

```
int A[10];
y
cargo_array(A,10);
```

esto cargará el array con el valor i en A[i] para todo i.

Otro ejemplo:

```
void intercambio_en_array (int A[], int i, int j)
{
      int s;
      s=A[i];
      A[i]=A[j];
      A[j]=s;
}
```

llamo con intercambio_en_array(3,6) por ejemplo. Sin pasar el array por referencia se modifica.

2. Modificando struct

El caso de struct es diferente del de los arrays. Si paso un struct como argumento por defecto es pasado por valor. Si quiero modificar el struct debo utilizar los operadores de dirección y de indirección.

Consideremos por ejemplo la siguiente struct:

```
#define MAX 100
struct Set {
        int e[MAX];
        int tope;};
```

la estructura se modifica en las siguientes operaciones:

```
void makenull(struct Set *A)
{ (*A).tope=0; }

void agrego_elemento(struct Set *A, int a)
{
      (*A).e[(*A).tope]=a;
      (*A).tope++;
}
```

las llamadas deben utilizar el operador de dirección, por ejemplo, podemos tener una función Set_input definida como sigue:

```
struct Set Set_input()
{
        int i=0;
        struct Set s;
        makenull(&s);
        while (i >=0)
        {
          cin >> i;
          cout << "\ n";
          if (i>=0) agrego_elemento(&s,i);
        }
        return s;
}
```

observar el uso de los operadores de dirección.

Otro ejemplo es dada la siguiente struct:

```
struct par {
        int x;
        int y;
        };
```

intercambiar sus elementos como sigue:

```
void intercambio_struct(struct par *p)
{     int aux;
```

```
    aux=(*p).x;
    (*p).x=(*p).y;
    (*p).y = aux;
}
```

llamamos utilizando el operador de dirección: intercambio_struct(&p1).