Punteros

Curso de Estructura de datos y algoritmos 1 – Universidad ORT Uruguay

Conceptos generales

1. ¿Que son los punteros?

Un puntero es una variable que contiene una dirección de memoria. El manejo de los punteros depende básicamente de dos operadores denominados operador de referenciación (el signo &) y el operador de desreferenciación (el signo *).

Estos son de gran utilidad al momento de utilizar valores y estructuras dinámicas, por ejemplo para crear un vector dinámico (su tamaño puede ser definido en tiempo de ejecución). En una segunda lectura se profundizara en este tema.

Consideraciones al utilizar punteros:

- El tipo de datos del puntero debe coincidir con el de la variable a la cual apunta.
- El &(ampersand) es un operador de referenciación que nos permite obtener la dirección de memoria de una variable cualquiera.
- El *(asterisco) es un operador de desreferenciación y permite obtener el valor almacenado en la dirección de memoria que apunta el puntero.

2. Ejemplo introductorio

```
1       int x = 25;
2       int y = 30;
3       int* ptr = &x;
4       cout << *ptr;
5       y = *ptr;
6       *ptr = 44;
```

A continuación pasaremos a describir línea a línea las instrucciones anteriores:

1) Esta declara una variable del tipo entero llamada “x”, que contiene el valor 25.
2) Esta declara una variable del tipo entero llamada “y”, que contiene el valor 30.
3) Esta declara una variable del tipo puntero a entero llamada “ptr”, que contiene la dirección de memoria de la variable “x”. Comúnmente se dice que “el puntero ptr apunta a la variable x”.
   Es importante destacar que para obtener la dirección de memoria de la variable x utilizamos el operador de referenciación.
4) Esta instrucción muestra por pantalla el valor de la variable a la cual apunta el puntero “ptr”. En este caso mostraría 25.
5) Asignamos a la variable “y” el valor de la variable a la cual apunta el puntero “ptr”, por lo tanto la variable “y” guardaría el valor 25.
6) Cambiamos el valor de la variable a la cual apunta el puntero “ptr”, es decir la variable “x” ahora tendría el valor 44.

Es importante tener en cuenta a la hora de utilizar los punteros la inicialización de estos. Dependiendo del compilador, la no inicialización de los punteros puede no

compilar, dar una advertencia o contener valores arbitrarios (a esto comúnmente le llamamos “basura”).

```
int* ptr;
cout << *ptr;
```

Por lo cual una buena práctica es inicializar los punteros. Una opción es inicializar los punteros en la constante NULL. Cuando un puntero se inicializa en esta constante se dice que este apunta a “nada”.

```
int* ptr = NULL;
```

3. Los punteros y los parámetros de las funciones

```
void main(){
   int a = 4;
   int b = 2;
   intercambiar(a, b);
   cout << a << endl;
   cout << b << endl;
}

void intercambiar(int x, int y){
   int auxiliar = x;
   x = y;
   y = auxiliar;
}
```

En C/C++, por defecto, todos los parámetros de las funciones se pasan por valor (la función recibe una copia del parámetro, que no se ve modificado por los cambios que al copia sufra dentro de la función).
Por lo cual es importante notar que en la función anterior las variables se pasan por copia por lo tanto se intercambian las variables copias, pero las variables a y b no se ven modificadas.

Por lo cual para poder realmente modificar el parámetro de una función, debemos pasar este por referencia, y en C/C++ esto es posible pasando la dirección de memoria de la variable en lugar de la propia variable.

```
void main(){
   int a = 4;
   int b = 2;
   intercambiar(&a, &b);
   cout << a << endl;
   cout << b << endl;
}

void intercambiar(int* x, int* y){
   int auxiliar = *x;
   *x = *y;
   *y = auxiliar;
}
```

Como se puede notar en el código anterior, en la llamada la función intercambiar estamos pasando la dirección de memoria de la variable “a” y la variable “b”; por lo cual en la función intercambiar estamos recibiendo un puntero a estas dos variables.

Ejercicios

1. Dada la siguiente funcion:

```
void misterio1(int *ptr){
   if (ptr == NULL){
        return;
   }
   cout << *ptr;
}
```

   1) ¿Que realiza la misma?
   2) ¿Por que se verifica si el parametro recibido es distinto de nulo?

2. Dada la siguiente funcion:

```
void misterio2(int* ptr, int* ptr2){
   if (ptr == ptr2){
        cout << *ptr;
   }
}
```

   ¿Cuando y que muestra la funcion?

3. Dada la siguiente funcion:

```
void misterio3(int* ptr, int* ptr2) {
   if (ptr != NULL && ptr2 !=NULL) {
        if (*ptr == *ptr2){
             cout << *ptr2;
        }
   }
}
```

   ¿Cuando y que muestra la funcion?

4. Dada la siguiente funcion:

```
void misterio4(int* ptr) {
   if (ptr != NULL) {
        cout << (*ptr) + 1;
   }
}
```

   ¿Cuando y que muestra la funcion?