Punteros

- Una variable de tipo puntero almacena una dirección de memoria.
- Dicha dirección de memoria puede ser la que almacena una variable o bien donde se encuentra el código de una función.
- Al declarar el puntero se indica el tipo de aquello a lo que apunta, de modo que pueda "interpretarse" lo que se encuentra en esa dirección.

Punteros

- Para declarar una variable de tipo puntero lo hacemos agregando el operador * entre el nombre de la variable y el tipo al que apunta

```
long l = 475; /* variable tipo long a la que
                 asignamos el valor 475 */
long k;   // otra variable long
long *pl;  // pl es de tipo "puntero a long"
```

- Para que una variable de tipo puntero “apunte” a una variable del tipo a la que apunta, usamos el operador de dirección & (también llamado operador de referencia)

```
pl = &l; /* ahora pl apunta a l, con &l
            obtenemos la dirección de l */
```

Punteros

- Para obtener el valor de la variable a la cuál apunta una variable de tipo puntero usamos el operador de desreferenciación *

```
k = *pl; /* ahora k vale 475 ya que *pl (lo
            apuntado por pl) equivale a l, es
            decir, esta última línea
            equivale a k = l */
```

Arreglos y punteros

- Por razones históricas al usar el identificador de un arreglo en una expresión, se convierte (decae) a un puntero al tipo de dato contenido en el vector
  - Si declaré `int vec[5];` entonces `vec` como parámetro es de tipo `int*`
  - Por eso el vector pasa “como si fuese por referencia” y nos obliga a pasar la dimensión como parámetro
  - También es la razón por la que no puedo devolver un vector.
- La notación `v[i]` sirve tanto para un vector como para un puntero

Asignación de memoria

- Al definir un puntero, el modo de decir que no apunta a ningún lado es asignarle el puntero nulo **`nullptr`** (que en general es el valor cero). **OJO**: intentar acceder a lo apuntado por un puntero con valor **`nullptr`** hace que se cuelgue el programa
- Asignación dinámica de memoria

```
float *pf = new float;
/* new aloja memoria del tamaño del tipo que le
piden y devuelve un puntero a ese tipo de dato */
```

- Una vez que la memoria asignada con new no es más necesaria, se libera con delete

```
delete pf;
/* libera la memoria asignada previamente para que
pueda ser reutilizada */
```

Asignación de memoria - arreglos

- Similar al caso anterior, pero indicando el tamaño del arreglo

```
double *vector = new double[5];
/* Aloja el espacio necesario para todo el
   vector */
```

- Para liberar la memoria con delete agregamos corchetes para indicar que estamos liberando un arreglo

```
delete[] vector;
// libera la memoria asigada previamente
```

Ejemplo vectores dinámicos

```
void mostrar_punt(int *vec, int dim)
//     idéntico a int vec[] pero menos claro
{
        for (int i = 0; i < dim; ++i)
                cout << i << ": " << vec[i] << endl;
}

int main()
{
        int dim;
        cout << "Dimensión del vectores de pares: ";
        cin >> dim;
        int *vec = new int[dim];
        for (int i = 0; i < dim; ++i)
                vec[i] = 2*i;
        mostrar_punt(vec, dim);
        delete[] vec;
        return 0;
}
```

Salida

```
Dimensión del vectores de pares: 10
0: 0
1: 2
2: 4
3: 6
4: 8
5: 10
6: 12
7: 14
8: 16
9: 18
```

Ejemplo Función Vector Dinámico

```
int *crear_vec10(void)
//Notar que devuelvo int* y no int[]
{
        int *vec = new int[10];
        for (int i = 0; i < 10; ++i)
                vec[i] = i;
        return vec;
}

int main()
{
        int *arr = crear_vec10();
        mostrar(arr, 10);
        delete[] arr;
        return 0;
}
```

Salida

```
0: 0
1: 1
2: 2
3: 3
4: 4
5: 5
6: 6
7: 7
8: 8
9: 9
```

Ejemplo INCORRECTO

```
int *crear_vec10_ERR(void)
{
        int vec[10];
        for (int i = 0; i < 10; ++i)
                vec[i] = i;
        return vec; /*devuelvo memoria que se liberar al salir
         de la función por eso el compilador da
        warning: address of local variable ‘vec’ returned */
}

int main()
{
        int *arr = crear_vec10_ERR();
        mostrar(arr, 10);
        delete[] arr; //También incorrecto
        return 0;
}

Salida: Segmentation fault (core dumped)
```

Punteros a Funciones

- Si tengo funciones similares en cuanto a parámetros y tipo que devuelve:

```
void ordenar_a(int[], int);
void ordenar_b(int[], int);
```

- Puedo declarar un puntero a ese tipo de función:

```
void (*pfo)(int[], int);
```

- Asigno con el nombre de la función (sin &):

```
pfo = ordenar_a;
```

- El puntero lo puedo usar igual que el identificador de la función:

```
pfo(vector, dimvec);
```

Ejemplo Puntero a Función

```
void buendia(void)
{
        cout << "Buen día" << endl;
}
void buenasnoches(void)
{
        cout << "Buenas noches" << endl;
}
int main()
{
        bool esdia = es_de_dia(); //suponemos ya programada es_de_dia
        void (*psaludo)(void);
        if (esdia)
                psaludo = buendia;
        else
                psaludo = buenasnoches;

        psaludo();
        return 0;
}
```

Licencia