Taller de repaso de punteros y operaciones con vectores

Introducción a la computación paralela

Brayan Andrés Quintero Pinto

2190083

H1

Escuela de ingeniería de sistemas e informática

Universidad Industrial de Santander

1 de noviembre del 2022

Resultados y conclusiones.

Se hicieron pruebas para n=5,80 y 200 para vectores generados con malloc(), new(), vector<> y vector<vector<>> y se obtuvieron los resultados expuestos en la siguiente tabla:

	Malloc	New	Vector	VectorVector
N=5	0.008s	0.009s	0.010s	0.009s
N=80	2.377s	2.413s	2.367s	2.384s
N=200	14.575s	14.704s	14.784s	14.924s

El mejor rendimiento para n=5 y n=200 se obtuvo con Malloc. Para n=80 el mejor rendimiento fue con la clase Vector.
Grosso modo, el mejor rendimiento se obtuvo con la función malloc().

Responder las siguientes preguntas.

1. **¿Qué es un operador de dereferenciación y qué es un operador de dirección de memoria? ¿Cómo se relacionan?**

El operador de indirección se representa con un asterisco ***** y se utiliza para operar sobre punteros y acceder a los valores a los que estos apuntan.

En cambio, el operador de dirección se representa con una 'y' comercial **&** y se utiliza sobre variables de cualquier tipo para obtener su dirección.

Por ejemplo:

Esto es una variable de tipo entero:

```
int x;
```

Esto es un puntero a un entero:

```
int *px;
```

Se da el valor 5 a x

```
x = 5;
```

Utilizando el operador de dirección se guarda en px la dirección de la variable x

```
px = &x;
```

Aquí a través del operador de indirección se asigna el valor 10 en la dirección a la que apunta px

```
*px = 10;
```

Esta última sería lo mismo que haber asignado directamente el valor 10 a la variable x ya que px apunta a x

```
x = 10;
```

2. **Si se usan estructuras y punteros ¿Cuál es la diferencia entre usar el operador . y el operador →?**

En esta declaración

```
typedef struct _game { int something; } g;
```

g es un *tipo*, no una variable. Como tal, g.something no tiene sentido. Typedef significa "definir tipo". En cambio, tendrías

```
g my_g_instance;
g *my_g_ptr = &my_g_instance;

my_g_instance.something = 2;
my_g_ptr->something = 5;
```

La diferencia entre . y → es si la variable a la izquierda del operador es un puntero o no.