UTN FRBA – AED – Examen Final – 2023-02-13

Apellido, Nombre:		Legajo:		Nota:	

- Resuelva el examen en en tinta.
- Durante el examen no se responden consultas; si lo necesita, escriba hipótesis de trabajo, las cuales también se evalúan.

El *problema* es simple: se necesita que sumar una secuencia de números enteros en el rango [0, 127].

1. Se pide codificar tres versiones distintas (*sobrecargas*) de la función `Sumar` que resuelven el problema. Declare todo lo que use:
   a. (*1 punto*) Una que suma los números que están en un *flujo de datos* (archivo abierto) que es parámetro de la función.
   b. (*1 punto*) Otra que suma los que están en un *arreglo* que es parámetro de la función.
   c. (*1 punto*) Y otra los que están en una *lista enlazada* que es parámetro de la función.
2. (*2 puntos*) Codifique un programa que *ejemplifique el uso de las tres versiones*.
3. (*1 punto*) Compare la eficiencia (*complejidad espacio/temporal*) de las versiones de arreglo y de lista enlazada. ¿Cuál ocupa más memoria? ¿Cuál es más lenta? ¿Por qué?
4. Intentemos una versión más. La versión *contigua* (arreglo) permite acceso directo pero tiene una cota en su capacidad; la enlazada no tiene esa cota pero requiere el doble de memoria para cada ítem para poder tener la referencia al siguiente, además, solicitar memoria del *heap* (con el operador `new`) para cada ítem es costoso a nivel tiempo, porque implica una llamada al sistema operativo.
   a. (*3 puntos*) Codifique una cuarta versión de `Sumar` que sea un *híbrido* entre las versiones de arreglo y lista enlazada, donde los nodos tienen arreglos de 1024 ítems para que reduzca espacio y tiempo por tener menos saltos y por solicitar memoria menos veces.
   b. (*1 punto*) Indique una desventaja de esta versión híbrida comparada con las dos originales.
5. (*Punto extra, extra difícil*) Implemente una versión de `Push` para la estructura "híbrida" anterior.

1. Una Resolución

```
//Una resolución del examen final de UTN FRBA AED 2023-02-13
//@josemariasola
#include<iostream>
#include<array>

// Todas las sobre cargas retoran un double !? #
// No debería ser un int? O un long int? o un long long int?
// Pista: Integer overflow versus floating overlow

double Sumar(std::istream&);

// ¿Por qué un template? ¿De qué otra forma se puede lograr el mismo efecto?
// ¿Por qué char?
template<auto N>
double Sumar(const std::array<char,N>& a){ //const char *a, unsigned n
 double s{0};       //double s=0;
 for(auto v : a)       //for(i{0};i<n;++i)
  s+=v;        // s+=a[i]
 return s;
}

struct Node{
 char v;      // char de nuevo. Por qué?
 Node* next{nullptr}; // por defecto, no tiene siguiente.
};

// Abstrae el concepto de lista enlazada del de puntero al primer nodo
struct LinkedList{
 Node* first{nullptr};
};

double Sumar(const LinkedList&);

struct PageNode{
 static const unsigned N{1024}; // Dónde se almacenan los miembros static?
 unsigned n{0};
 std::array<char,N> v;   // Otra vez char!?
 PageNode* next{nullptr};
};

// Lo mismo, pero para nodos de páginas
struct LinkedPageList{
 PageNode* first{nullptr};
};
```

```
double Sumar(const LinkedPageList&);

LinkedList CrearListaEnlazadaEjemplo();
LinkedPageList CrearListaDePáginasEjemplo();

void Push(LinkedPageList&, char); // El punto extra y ...

void TestPush();     // ... su prueba.

int main(){
 std::cout
  << "Sumatoria: " << Sumar(std::cin)                    << '\n'
  << "Sumatoria: " << Sumar(std::array<char,4>{1,2,3,4})  << '\n'
  << "Sumatoria: " << Sumar(CrearListaEnlazadaEjemplo())  << '\n'
        // al finalizar se libera
  << "Sumatoria: " << Sumar(CrearListaDePáginasEjemplo()) << '\n'
        // al finalizar se libera
  ;
 TestPush();
}

/* El rango es de 0 a 127, conincide con el ASCII (el original), y los char
 tienen capacidad para guardar cualquier valor ASCII. También hubiese funcionado
 unsigend char y signed char. Las tres versiones de char ocupan un byte.
Entonces, los char son para almacenar caracteres ó bytes ó números ó números en
 un rango?*/
double Sumar(std::istream& in){
 double s{0};
 for(char v; in.get(v);) // obtiene bytes,  la versión
       // formateada, in >> v, no funciona. Por qué?
  s+=v;
 return s;
}

double Sumar(const LinkedList& l){
 double s{0};
 for(auto p{l.first}; p; p = p->next) // Miren! Sin while ni llaves!!
  s+=p->v;
 return s;
}

/* Más allá de que la versión enlazada requiera más operaciones para pasar
 de nodo y controlar de finalización, comparada con la única operación de
 subindicación ambos recorridos son O(n).

La última página desperdicia espacio. */
```

```
double Sumar(const LinkedPageList& l){
 double s{0};
 for(auto p{l.first}; p; p = p->next)
  for(unsigned i{0}; i<p->n; ++i)
   s+=p->v.at(i);
 return s;
}

LinkedList CrearListaEnlazadaEjemplo(){
 return
  LinkedList{ // Las listas son estructuras recursivas
   new Node{
    1,
    new Node{
     2,
     new Node{
      3,
      nullptr
     }
    }
   }
  };
}

LinkedPageList CrearListaDePáginasEjemplo(){
 return
  LinkedPageList{ // Mamushka!
   new PageNode{
    2,
    {1,2},
    new PageNode{
     1,
     {3},
     new PageNode{
      3,
      {4,5,6},
      nullptr
     }
    }
   }
  };
}

void Push(LinkedPageList& l, char val){
 if(nullptr == l.first){      // Si está vacía entonces
  l.first = new PageNode{1,{val},nullptr};// creaamos la primera página.
  return;
```

```
 }
 if(l.first->n == l.first->N){ // Si no, si la primer página está completa,
  l.first = new PageNode{1,{val},l.first}; // creamos otra.
  return;
 }
 l.first->v.at(l.first->n++)=val; // Si no, hay lugar en la primera página.
}

void TestPush(){
 LinkedPageList l;
 assert(nullptr == l.first); // Se supone que las listas se crean vacías.
 for(unsigned i{0}; i<PageNode::N*10; ++i)// Pusheamos tantos números como
  Push(l,char(i%128)); // entran en 10 páginas y aseguramos esté en rango
 assert(nullptr != l.first); // Se supone que ahora no está vacía.

 for(auto p{l.first}; p; p = p->next)
  for(unsigned i{0}; i<p->n; ++i)
   assert( i%128 == p->v.at(i)); // verificamos lo que guardamos
   //std::cout << int(p->v.at(i)) << ' ';
   // Si no confiamos en el debugger o assert

 Push(l,'J'); // Nos quedamos sin espacio en la primera página, entonces se
 assert(1 == l.first->n); // crea una nueva página con solo lo que pusheamos,
 assert('J' == l.first->v.at(0));// que era una jota.
}
```

v1.1.0 2023-02-19