

Introducción al Cómputo

Recursión
Strings nativos
`std::string`
File I/O

Recursión

- Especificación de un proceso basado en su propia definición. Involucra:
 - Una o más soluciones triviales (directas)
 - Una o más soluciones que recurren en el mismo problema, directa o indirectamente

Recursión

Factorial de un número

$$n! = \begin{cases} \Rightarrow 1 & \text{si } n == 0 \\ \Rightarrow n(n-1)! & \text{si } n > 0 \end{cases}$$

Recursión

Secuencia de Fibonacci

$$Fb(n) = \begin{cases} \Rightarrow 0 & \text{si } n == 0 \\ \Rightarrow 1 & \text{si } n == 1 \\ \Rightarrow Fb(n - 1) + Fb(n - 2) & \text{si } n > 1 \end{cases}$$

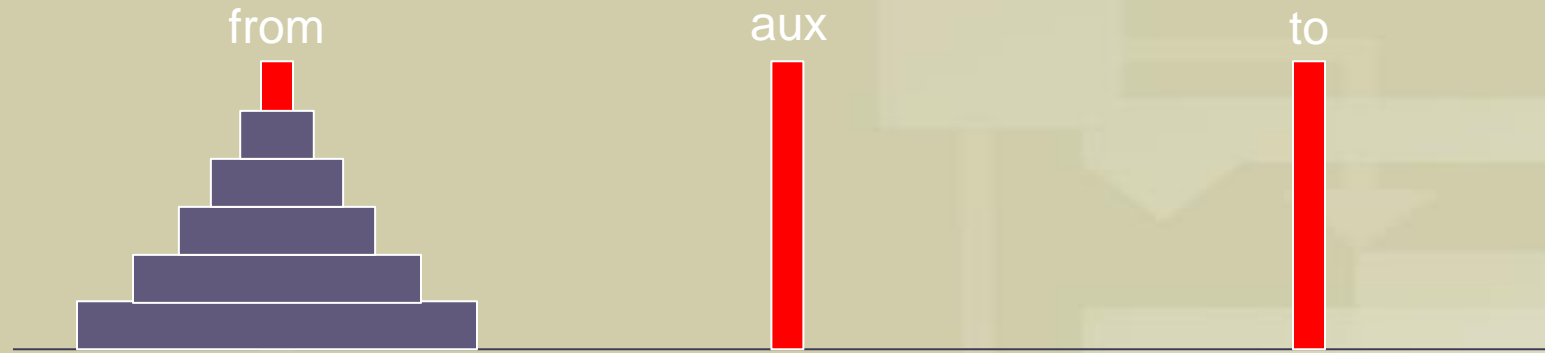
Recursión

```
int factorial(int n) {  
    if( n == 0 )  
        return 1;  
    return n * factorial(n - 1);  
}  
  
int fb(int n) {  
    if( n == 0 || n == 1)  
        return n;  
    return fb(n - 1) + fb(n - 2);  
}  
  
int main()  
{  
    cout << "7! = " << factorial(7) << endl;  
    cout << "fb(7) = " << fb(7) << endl;  
    return 0;  
}
```

$$n! = \begin{cases} \Rightarrow 1 & \text{si } n == 0 \\ \Rightarrow n(n-1)! & \text{si } n > 0 \end{cases}$$

$$Fb(n) = \begin{cases} \Rightarrow 0 & \text{si } n == 0 \\ \Rightarrow 1 & \text{si } n == 1 \\ \Rightarrow Fb(n-1) + Fb(n-2) & \text{si } n > 1 \end{cases}$$

Hanoi



El objeto es mover los n discos desde el poste 'from' al poste 'to' utilizando el poste auxiliar 'aux'.

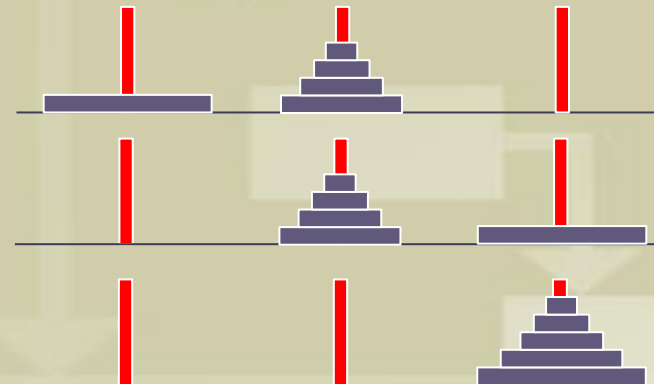
Los discos deben moverse de a uno por vez.

Nunca se debe pasar por la condición de que un disco esté arriba de otro de menor diámetro.

Hanoi



```
void hanoi(int n, int from, int to, int aux) {  
    if( n == 1 ) {  
        cout << "Mover de " << from << " a " << to << endl;  
        return;  
    }  
    hanoi( n-1, from, aux, to);  
    hanoi( 1, from, to, aux);  
    hanoi( n-1, aux, to, from);  
}  
  
int main() {  
    hanoi(3, 1, 3, 2);  
    return 0;  
}
```



Cadenas alfanuméricas como tipo nativo

```
void concat(char result[], const char str1[], int n1,  
            const char str2[], int n2);  
  
int main(void)  
{  
    char s1[5] = { 'T', 'e', 's', 't', ' ' };  
    char s2[6] = { 'w', 'o', 'r', 'k', 's', '.' };  
    char s3[20];  
    int i;  
  
    concat(s3, s1, 5, s2, 6);  
  
    for( i = 0; i < 11; ++i )  
        cout << s3[i];  
    cout << '\n';  
  
    return 0;  
}
```


Solución: arreglo de char + convención

```
cout << "Hello world!\n";
```

- "Hello world!\n" es un string nativo
- Un string nativo es un array nativo de **char** terminado con un 0.

```
char s1[] = { 'T', 'e', 's', 't', 0 };
```

```
char s2[] = "Test";
```

- No es necesario pasar la longitud de los strings nativos:

```
void concat(char result[], const char str1[],  
            const char str2[]);
```

Cadenas alfanuméricas como tipo nativo?

```
void concat(char result[], const char str1[],
            const char str2[])
{
    int idx1 = 0, idx2 = 0;
    char c;

    while( (c = str1[idx1]) != 0 )
        result[idx1++] = c;

    while( (c = str2[idx2]) != 0 )
        result[idx1 + idx2++] = c;

    result[idx1 + idx2] = 0;
}
```

Cadenas alfanuméricas como tipo nativo?

Supongamos que queremos comparar dos strings (nativos) y queremos para eso hacer una función que termine retornando un valor <0 , $=0$ o >0 indicando que $s1$ representa un string menor, igual o mayor que $s2$ (alfabéticamente). Como lo que se pasa a la función son solo las direcciones de memoria del primer `char` de cada string nativo, no queda otra que recorrer los caracteres:

```
int res = compara("ALABAMA", "ALBA");

int compara(const char s1[], const char s2[]){
    int i = 0;

    while( s1[i] && s1[i] == s2[i])
        i++;

    return s1[i] - s2[i];
}

int largo(const char s[]) {
    int i = 0;
    for( ; s[i] != 0; i++) ;
    return i;
}
```

std::string

- C++ provee una manera alternativa de manejar las cadenas de caracteres: `std::string`.
- Múltiples maneras de inicialización
- Concatenación
- Comparación
- Métodos de manipulación
- Acceso a caracteres individuales
- Búsqueda y substrings
- Etc.
- Etc.

Ejemplos con std::string

ejem7_1.cpp

```
#include "icom_helpers.h"
int main(void)
{
    string s1;
    string s2("AAA");    // con inicialización
    string s3 = "BBB";   // ídem anterior
    string s4{"CCC"};    // ídem anterior
    string s5{s2};       // ídem anterior

    s1 = s2 + s3 + s4;   // concatenación

    if(s2 > s3)           // comparación
        cout << "estamos en problemas\n";
    if(s2 == s5)
        cout << "Menos mal!\n";

    cout << "s1: " << s1 << " largo: " << s1.length() << '\n';
    return 0;
}
```

Ejemplos con std::string

ejem7_2.cpp

```
#include "icom_helpers.h"
// acceso a caracteres individuales
int main(void)
{
    string s("Hola Pirulo");

    for(int i = 0; i < s.length(); ++i) // recorrido indexado
        cout << s[i];
    cout << '\n';

    for(auto c : s) // recorrido con range-based for
        cout << c;
    cout << '\n';

    for(auto it = s.begin(); it != s.end(); ++it) // idem a través de un iterador
        cout << *it;
    cout << '\n';

    return 0;
}
```

Ejemplos con std::string

ejem7_3.cpp

```
#include "icom_helpers.h"
// manipulacion
int main(void)
{
    string s("Sean laureles los laureles");    // con inicialización
    string s2("laureles");

    size_t pos = s.find(s2);
    if(pos != string::npos)
        cout << s2 << " esta en " << pos << '\n';

    pos = s.find(s2, pos+1);
    if(pos != string::npos)
        cout << s2 << " esta nuevamente en " << pos << '\n';

    s.replace(s.find(s2), s2.length(), "eternos");
    cout << s << '\n';

    return 0;
}
```

Manipulación de archivos

- C/C++ tienen una gran cantidad de maneras de manipular archivos/dispositivos, siempre a través de bibliotecas, no como parte del lenguaje.
- Para un manejo básico de archivos, el uso simple de los tipos `std::ifstream` y `std::ofstream` resulta suficiente.
- Desde el primer ejemplo de “`hola mundo`” estuvimos utilizando cosas muy parecidas a través de `cin` y `cout`:
 - `cin` es una instancia (objeto) del tipo (clase) `std::istream` asociado al dispositivo estándar de entrada (redireccionable)
 - `cout` es una instancia (objeto) del tipo (clase) `std::ostream` asociado al dispositivo estándar de salida (redireccionable)
- El protocolo simplificado de uso es:
 - Declaración -> apertura -> manipulación (lecturas o escrituras) -> cierre

Ejemplos con archivos

```
#include "icom_helpers.h"
```

ejem7_4.cpp

```
int main(void)
```

```
{
```

```
    ofstream oFile;           // crea la instancia de ofstream
```

```
    oFile.open("kk.dat");     // abre un archivo y lo asocia a oFile
```

```
                                // podria haber declarado ofstream oFile("kk.dat")
```

```
    if(oFile.is_open()) {     // equivalente a if(oFile) { ...
```

```
        oFile << "Esto va a parar al file\n";
```

```
        oFile << "y esto\n" << "y esto tambien...\n";
```

```
        oFile.close();
```

```
    }
```

```
    ifstream iFile("kk.dat");
```

```
    if(iFile) {
```

```
        string line;
```

```
        while ( getline (iFile, line) )
```

```
            cout << line << '\n';
```

```
    }
```

```
    return 0;
```

```
}
```

Ejemplos con archivos

```
#include "icom_helpers.h"
// genera un archivo con NUM_DATOS números aleatorios entre 0 y 999

const int NUM_DATOS = 500;

int main(void)
{
    ofstream oFile("datos.dat");
    if(oFile.is_open()) {

        srand(time(0));

        for(int i = 0; i < NUM_DATOS; ++i)
            oFile << (rand()%1000) << '\n';

        oFile.close();
    }
    return 0;
}
```

ejem7_5.cpp

Ejemplos con archivos

ejem7_6.cpp

```
#include "icom_helpers.h"
// Lee datos desde "datos.dat", los ordena y los escribe en "datos_ord.dat"
int main(void)
{
    ifstream iFile("datos.dat");
    if(iFile.is_open()) {
        vector<int> datos;
        int d;
        while(iFile >> d)
            datos.push_back(d);
        iFile.close();

        sort(datos.begin(), datos.end());

        ofstream oFile("datos_ord.dat");
        for(int d: datos)
            oFile << d << '\n';
    }
    return 0;
}
```