UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y CC. Computación
E.T.S.I. Telecomunicación

GUÍA DE REFERENCIA DE C++

Programa C++

```
// Inclusión de bibliotecas
// Utilización de espacios de nombre
// Definición de Constantes y Tipos
// Definición de subprogramas
int main()
{
    // cuerpo del programa principal
}


// g++ -ansi -Wall -Wextra -Werror -o main main.cpp
```

Tipos Simples

```
bool char short int unsigned float double
enum Color { ROJO, VERDE, AZUL };
```

Constantes, Variables, Asignación

```
const double PI = 3.1415;
int main()
{
    char a, b;   // sin inicializar
    Color c;     // sin inicializar
    int x = 1;   // inicialización
    x = x * 2;   // asignación
    a = 'z';     // asignación
    c = ROJO;    // asignación
    x = int(PI); // asig, conversión
}
```

Operadores: Precedencia, Asociatividad

Operador	Tipo de Operador	Asociatividad
`[] -> .`	Binarios	Izq. a Dch.
`! - *`	Unarios	Dch. a Izq.
`* / %`	Binarios	Izq. a Dch.
`+ -`	Binarios	Izq. a Dch.
`< <= > >=`	Binarios	Izq. a Dch.
`== !=`	Binarios	Izq. a Dch.
`&&`	Binario	Izq. a Dch.
`||`	Binario	Izq. a Dch.
`?:`	Ternario	Dch. a Izq.

Asignación e Incrementos

Sentencia	Equivalencia
`++vble;`	`vble = vble + 1;`
`--vble;`	`vble = vble - 1;`
`vble++;`	`vble = vble + 1;`
`vble--;`	`vble = vble - 1;`
`vble += expr;`	`vble = vble + (expr);`
`vble -= expr;`	`vble = vble - (expr);`
`vble *= expr;`	`vble = vble * (expr);`
`vble /= expr;`	`vble = vble / (expr);`
`vble %= expr;`	`vble = vble % (expr);`

Entrada / Salida Básica

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    int x, y;
    cin >> x >> y;
    cout << "Resultado: " << x << " " << y << endl;
    cin.ignore(1000, '\n');
    cin >> ws;
    char a;
    cin.get(a);
    string nombre;
    getline(cin, nombre);
}
```

Salida de Datos Formateada(*)

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    cout << dec << 27 ;
    cout << hex << 27 ;
    cout << oct << 27 ;
    cout << setprecision(2) << 4.567 ;
    cout << setw(5) << 234 ;
    cout << setfill('#') << setw(5) << 234 ;
}
```

Estructuras de Control

```
if ( expr_lógica ) {
    ...
} else if ( expr_lógica ) {
    ...
} else {
    ...
}

switch ( expr_ordinal ) {
case VALOR:
    ...
    break;
default:
    ...
    break;
}
```

```
for ( unsigned i = 0; i < NELMS; ++i ) {
    ...
}

while ( expr_lógica ) {
    ...
}

do {
    ...
} while ( expr_lógica );
```

Subprogramas

```
int funcion(int p_valor_1, const Fecha& p_ref_cte_2)
{
    ...
    return valor;
}

void procedimiento(int& p_ref_1, Fecha& p_ref_2)
{
    ...
}

int main()
{
    Fecha f = { 29, 6, 2011 };
    int x = funcion(3, f);
    procedimiento(x, f);
}
```

Tipo String

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string nm = "pepe luis";
    if (nm <= "lucas") {
        nm += " lopez";
    }
    nm = nm.substr(5, 4) + "miguel";
    for (int i = 0; i < nm.size(); ++i) {
        nm[i] = char(nm[i] + 1);
    }
    cin >> nm; // getline(cin , nm);
    cout << nm << endl;
}
```

Tipo Registro

```
struct Fecha {
    unsigned dia;
    unsigned mes;
    unsigned anyo;
};
int main()
{
    Fecha f = { 29, 6, 2011 };
    Fecha hoy = f;
    hoy = f;
    hoy.dia = f.dia;
    hoy.mes = f.mes;
    hoy.anyo = f.anyo;
}
```

Tipo Array (std)

```
#include <array>
using namespace std;
const unsigned NELMS = 20;
typedef array<int, NELMS> Dat;
int main()
{
    Dat d = {{ 1, 2, 3 }};
    Dat aux = d;
    if (d <= aux) {
        aux = d;
    }
    for (int i = 0; i < d.size(); ++i) {
        d[i] = d[i] * 2;
    }
}
```

Tipo Array 2 Dimensiones

```
#include <array>
using namespace std;
const unsigned NFILS = 2;
const unsigned NCOLS = 3;
typedef array<int, NCOLS> Fila;
typedef array<Fila, NFILS> Matriz;
int main()
{
    Matriz m = {{
        {{ 1, 2, 3 }},
        {{ 4, 5, 6 }}
    }};
    Matriz aux = m;
    if (m <= aux) {
        aux = m;
    }
    for (int f = 0; f < m.size(); ++f) {
        for (int c = 0; c < m[f].size(); ++c) {
            m[f][c] = m[f][c] * 2;
        }
    }
}
```

Ficheros: Salida de Datos

```
#include <string>
#include <fstream>
using namespace std;
int main()
{
    string nombre = "datos.txt";
    ofstream f_out;
    f_out.open(nombre.c_str());
    if ( ! f_out.fail() ) {
        f_out << "...datos..." << endl;
        bool ok = ! f_out.fail();
        f_out.close();
    }
}
```

Ficheros: Entrada de Datos

```
#include <string>
#include <fstream>
using namespace std;
int main()
{
    string nombre = "datos.txt";
    ifstream f_in;
    f_in.open(nombre.c_str());
    if ( ! f_in.fail() ) {
        string datos;
        f_in >> datos;
        bool ok = ! f_in.fail() || f_in.eof();
        f_in.close();
    }
}
```

Módulos

```
//--- modulo.hpp ----------------------
#ifndef modulo_hpp_
#define modulo_hpp_
#include <string>
namespace umalcc {
    struct Dato {
        std::string xx;
    };
    void put(Dato& d, const std::string& x);
    std::string get(const Dato& d);
}
#endif
//--- modulo.cpp ----------------------
#include "modulo.hpp"
using namespace std;
namespace umalcc {
    void put(Dato& d, const string& x)
    {
        d.xx = x;
    }
    string get(const Dato& d)
    {
        return d.xx;
    }
}
```

```
//--- main.cpp -------------------------
#include <iostream>
#include "modulo.hpp"
using namespace std;
using namespace umalcc;
int main()
{
    Dato d;
    put(d, "pepe");
    cout << get(d) << endl;
}
// g++ -ansi -Wall -Wextra -Werror -c modulo.cpp
// g++ -ansi -Wall -Wextra -Werror -c main.cpp
// g++ -o main main.o modulo.o
```

TAD: Tipo Abstracto de Datos (I)

```
//--- modulo.hpp ----------------------
#ifndef modulo_hpp_
#define modulo_hpp_
#include <string>
namespace umalcc {
    class Dato {
    public:
        ~Dato() ;
        Dato() ;
        Dato(const Dato& o) ;
        Dato(const std::string& x) ;
        Dato& operator=(const Dato& o) ;
        void put(const std::string& x) ;
        std::string get() const ;
    private:
        std::string xx;
    };
}
#endif
```

```
//--- modulo.cpp ----------------------
#include "modulo.hpp"
using namespace std;
namespace umalcc {
    Dato::~Dato() {}
    Dato::Dato() : xx() {}
    Dato::Dato(const Dato& o) : xx(o.xx) {}
    Dato::Dato(const string& x) : xx(x) {}
    Dato& Dato::operator=(const Dato& o)
    {
        if (this != &o) {
            xx = o.xx;
        }
        return *this;
    }
    void Dato::put(const string& x)
    {
        xx = x;
    }
    string Dato::get() const
    {
        return xx;
    }
}
```

TAD: Tipo Abstracto de Datos (II)

```
//--- main.cpp -------------------------
#include <iostream>
#include "modulo.hpp"
using namespace std;
using namespace umalcc;
int main()
{
    Dato d;
    d.put("pepe");
    cout << d.get() << endl;
}
// g++ -ansi -Wall -Wextra -Werror -c modulo.cpp
// g++ -ansi -Wall -Wextra -Werror -c main.cpp
// g++ -o main main.o modulo.o
```

Subprogramas Genéricos

```
#include <iostream>
#include <array>

template <typename Tipo>
inline void intercambio(Tipo& x, Tipo& y)
{
    Tipo aux = x;
    x = y;
    y = aux;
}
template <typename Tipo, unsigned SZ>
void print(const std::array<Tipo, SZ>& a)
{
    for (int i = 0; i < a.size(); ++i) {
        std::cout << a[i] ;
    }
}
const unsigned NELMS = 5;
typedef std::array<int, NELMS> Dat;
int main()
{
    int a = 3, y = 5;
    intercambio(a, b);
    Dat d = {{ 1, 2, 3, 4, 5 }};
    print(d);
}
```

TAD Genérico: Definición En-Línea

```
//--- modulo.hpp ----------------------
#ifndef modulo_hpp_
#define modulo_hpp_
namespace umalcc {
    template <typename Tipo>
    class Dato {
    public:
        ~Dato() {}
        Dato() : xx() {}
        Dato(const Dato& o) : xx(o.xx) {}
        Dato(const Tipo& x) : xx(x) {}
        Dato& operator=(const Dato& o)
        {
            if (this != &o) {
                xx = o.xx;
            }
            return *this;
        }
        void put(const Tipo& x)
        {
            xx = x;
        }
        Tipo get() const
        {
            return xx;
        }
    private:
        Tipo xx;
    };
}
#endif
```

```
//--- main.cpp ------------------------
#include <iostream>
#include <string>
#include "modulo.hpp"
using namespace std;
using namespace umalcc;
int main()
{
    Dato<string> d;
    d.put("pepe");
    cout << d.get() << endl;
}
// g++ -ansi -Wall -Wextra -Werror -o main main.cpp
```

Memoria Dinámica

```
struct Nodo;
typedef Nodo* PNodo;
struct Nodo {
    PNodo sig;
    int dato;
};
int main()
{
    PNodo ptr = new Nodo;
    PNodo ptr2 = ptr;
    ptr->sig = NULL;
    ptr->dato = 3;
    Nodo n = *ptr;
    if ((ptr != NULL)&&(ptr == ptr2)) {
        ptr = ptr->sig;
    }
    delete ptr;
}
```

Contenedores STL: Vector

```
#include <vector>
using namespace std;
typedef vector<int> VectorInt;
int main()
{
    VectorInt v1;
    VectorInt v2(10);
    VectorInt v3(10, 0);
    VectorInt v4 = v3;
    if (v4 <= v3) {
        v4 = VectorInt(10);
        v4.swap(v3);
    }
    for (int i = 0; i < v4.size(); ++i) {
        v4[i] = v4[i] * 2;
    }
    v4.clear();
    v4.push_back(3);
    v4.resize(10, 5);
    v4.resize(7);
    if ( ! v4.empty() ) {
        v4.pop_back();
    }
}
```

Contenedores STL: Vector 2D

```
#include <vector>
using namespace std;
typedef vector<int> FilaInt;
typedef vector<FilaInt> MatrizInt;
int main()
{
    MatrizInt m1;
    MatrizInt m2(2, FilaInt(3));
    MatrizInt m3(2, FilaInt(3, 0));
    MatrizInt m4 = m3;
    if (m4 <= m3) {
        m4 = MatrizInt(5, FilaInt(7));
        m4.swap(m3);
    }
    for (int f = 0; f < m4.size(); ++f) {
        for (int c = 0; c < m4[f].size(); ++c) {
            m4[f][c] = m4[f][c] * 2;
        }
    }
    m4.clear();
    for (int f = 0; f < 3; ++f) {
        m4.push_back(FilaInt());
        for (int c = 0; c < 5; ++c) {
            m4[m4.size()-1].push_back(0);
        }
    }
    m4.resize(10, FilaInt(5, 2));
    m4.resize(7);
    if ( ! m4.empty() ) {
        m4.pop_back();
    }
}
```

Contenedores STL: Deque

```
#include <deque>
using namespace std;
typedef deque<int> DequeInt;
int main()
{
    DequeInt v1;
    DequeInt v2(10);
    DequeInt v3(10, 0);
    DequeInt v4 = v3;
    if (v4 <= v3) {
        v4 = DequeInt(10);
        v4.swap(v3);
    }
    for (int i = 0; i < v4.size(); ++i) {
        v4[i] = v4[i] * 2;
    }
    v4.clear();
    v4.push_back(3);
    v4.resize(10, 5);
    v4.resize(7);
    if ( ! v4.empty() ) {
        v4.pop_back();
    }
    v4.push_front(7); // no en vector
    v4.pop_front();   // no en vector
}
```

Contenedores STL: Stack

```
#include <stack>
using namespace std;
typedef stack<int> StackInt;
int main()
{
    StackInt s1;
    StackInt s2 = s1;
    if ((s2 <= s1)||(s2.size() > 3)) {
        s2 = StackInt();
    }
    s1.push(3);
    if ( ! s1.empty() ) {
        int cima = s1.top();
        s1.pop();
    }
}
```

Contenedores STL: Queue

```
#include <queue>
using namespace std;
typedef queue<int> QueueInt;
int main()
{
    QueueInt q1;
    QueueInt q2 = q1;
    if ((q2 <= q1)||(q2.size() > 3)) {
        q2 = QueueInt();
    }
    q1.push(3);
    if ( ! q1.empty() ) {
        int primero = q1.front();
        int ultimo = q1.back();
        q1.pop();
    }
}
```

Asertos(*)

```
#include <cassert>
int main()
{
    ...
    assert((x > 3)&&(y < 6));
    ...
}
```