

Metodología de la Programación

Tema 3. Funciones (ampliación)

Andrés Cano Utrera
(acu@decsai.ugr.es)
Departamento de Ciencias de la Computación e I.A.



Curso 2014-15

Contenido del tema

- 1 La función main
- 2 Parámetros con valor por defecto
- 3 Sobrecarga de funciones
- 4 Funciones inline
- 5 Variables locales static

Contenido del tema

- 1 La función main
- 2 Parámetros con valor por defecto
- 3 Sobrecarga de funciones
- 4 Funciones inline
- 5 Variables locales static

La función main I


- Un programa C++ comienza cuando el SO transfiere el control a main y finaliza cuando esta función acaba.
- Hasta ahora, hemos usado la siguiente cabecera simple para **main**:
`int main()`
- C++ permite una versión ampliada de la cabecera de main:
`int main(int argc, char *argv[])`
 - **Valor de retorno:** El int devuelto por main informa al SO sobre el posible código de error del programa.
 - 0: Ok (valor por defecto)
 - Otro valor: Algún tipo de error
 - **Argumentos de main:**
 - int argc: Número de argumentos usados al ejecutar el programa.
 - char *argv[]: Array de cadenas con cada uno de los argumentos.
argv[0]: Nombre del ejecutable
argv[1]: Primer argumento
...

La función main II: Ejemplo

```

1 #include <iostream>
2 using namespace std;
3 int main(int argc, char *argv[]){
4     if (argc<3){
5         cerr << "Uso: "
6             << " <Fichero1> <Fichero2> ..." << endl;
7         return 1;
8     }
9     else{
10        cout<<"Numero argumentos: " << argc << endl;
11        for (int i=0; i<argc; ++i){
12            cout<<argv[i]<<endl;
13        }
14    }
15    return 0;
16 }

```




La función main III

Podemos convertir las cadenas estilo C al tipo string

```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     string par;
7     cout<<"Argumentos: " << endl;
8     for (int i=0; i<argc; ++i)
9     {
10        par=argv[i];
11        cout<<par<<endl;
12    }
13    return 0;
14 }
15

```



Contenido del tema

- 1 La función main
- 2 **Parámetros con valor por defecto**
- 3 Sobrecarga de funciones
- 4 Funciones inline
- 5 Variables locales static

Parámetros con valor por defecto

Una función o método puede tener parámetros con un valor por defecto

- Deben ser los últimos de la función.
- En la llamada a la función, si sólo se especifican un subconjunto de ellos, deben ser los primeros.

```

void funcion(char c, int i=7){
    ...
}
int main(){
    funcion('a',8);
    funcion('z');
}


```

Parámetros con valor por defecto: Ejemplo

```

1 #include <iostream>
2 using namespace std;
3 int volumenCaja(int largo=1, int ancho=1, int alto=1);
4 int main()
5 {
6     cout << "Volumen por defecto: " << volumenCaja() << endl;
7     cout << "El volumen de una caja (10,1,1) es: " << volumenCaja(10)
<< endl;
8     cout << "El volumen de una caja (10,5,1) es: " << volumenCaja(10,5)
<< endl;
9     cout << "El volumen de una caja (10,5,2) es: " <<
volumenCaja(10,5,2) << endl;
10    return 0;
11 }
12 int volumenCaja( int largo, int ancho, int alto )
13 {
14     return largo * ancho * alto;
15 }

```



Contenido del tema

- 1 La función main
- 2 Parámetros con valor por defecto
- 3 Sobrecarga de funciones
- 4 Funciones inline
- 5 Variables locales static

Sobrecarga de funciones

Sobrecarga de funciones

C++ permite definir varias funciones en el mismo ámbito con el mismo nombre. C++ selecciona la función adecuada en base al número, tipo y orden de los argumentos.

```

void funcion(int x){
    ...
}
void funcion(double x){
    ...
}
void funcion(char *c){
    ...
}
void funcion(int x, double y){
    ...
}

int main(){
    char *c;
    funcion(3);
    funcion(4.5);
    funcion(4,9.3);
    funcion(c);
}

```

Sobrecarga de funciones

Conversión implícita de tipos

C++ puede aplicar conversión implícita de tipos para buscar la función adecuada.

```

void funcion(double x){
    cout << "double" << x << endl;
}
void funcion(char *p){
    cout << "char *" << *p << endl;
}
int main(){
    funcion(4.5);
    funcion(3); // conversion implicita
}

```

Sobrecarga de funciones

Distinción por el tipo devuelto

C++ no puede distinguir entre dos versiones de función que sólo se diferencian en el tipo devuelto.

```
int funcion(int x){
    return x*2;
}
double funcion(int x){
    return x/3.0;
}
int main(){
    int x=funcion(3);
    double f=funcion(5);
}
```

Sobrecarga de funciones

Uso de const en punteros y referencias

C++ puede distinguir entre versiones en que un parámetro puntero o bien referencia es const en una versión y en la otra no.

```
1 #include <iostream>
2 using namespace std;
3 void funcion(double &x){
4     cout << "funcion(double &x): " << x << endl;
5 }
6 void funcion(const double &x){
7     cout << "funcion(const double &x): " << x << endl;
8 }
9 int main(){
10     double x=2;
11     const double A=4.5;
12     funcion(A);
13     funcion(x);
14 }
```



Sobrecarga de funciones

Uso de const en punteros y referencias

C++ puede distinguir entre versiones en que un parámetro puntero o bien referencia es const en una versión y en la otra no.

```
1 #include <iostream>
2 using namespace std;
3 void funcion(double *p){
4     cout << "funcion(double *p): " << *p << endl;
5 }
6 void funcion(const double *p){
7     cout << "funcion(const double *p): " << *p << endl;
8 }
9 int main(){
10     double x=2;
11     const double A=4.5;
12     funcion(&A);
13     funcion(&x);
14 }
```



Sobrecarga de funciones

Uso de const en parámetros por valor

Sin embargo, C++ no puede distinguir entre versiones en que un parámetro por valor es const en una versión y en la otra no.

```
#include <iostream>
using namespace std;
void funcion(double x){
    cout << "funcion(double x): " << x << endl;
}
void funcion(const double x){
    cout << "funcion(const double x): " << x << endl;
}
int main(){
    double x=2;
    const double A=4.5;
    funcion(A);
    funcion(x);
}
```

Sobrecarga de funciones

Ambigüedad

A veces pueden darse errores de ambigüedad

```
void function(int a, int b){
    ...
}
void function(double a, double b){
    ...
}
int main(){
    function(2,4);
    function(3.5,4.2);
    function(2,4.2); //Ambiguo
    function(3.5,4); //Ambiguo
    function(3.5,static_cast<double>(4));
}
```

Contenido del tema

- 1 La función main
- 2 Parámetros con valor por defecto
- 3 Sobrecarga de funciones
- 4 Funciones inline**
- 5 Variables locales static

Sobrecarga de funciones

Otro ejemplo de ambigüedad

En este caso al usar funciones con parámetros por defecto

```
void function(char c, int i=7){
    ...
}
void function(char c){
    ...
}
int main(){
    function('a',8);
    function('z');
}
```

Funciones inline

Función inline

Es una forma de declarar una función para que el compilador genere una copia de su código, cada vez que es llamada, para evitar una llamada a función, y así aumentar la velocidad de ejecución del programa.

- Se definen colocando `inline` antes del tipo de retorno en la definición de la función.
- Suelen ser funciones pequeñas y que son llamadas con mucha frecuencia.
- Fueron introducidas en C++ para solucionar los problemas de las macros (no comprobación de tipos, problemas al expandirlas, etc).
- Ejecución más rápida en general.
- Código generado de mayor tamaño.
- El compilador puede que no haga caso al calificador `inline`.
- Suelen colocarse en ficheros de cabecera (`.h`) ya que el compilador necesita su definición para poder expandirlas.

Funciones inline: Ejemplo

```

1 #include <iostream>
2 inline bool numeroPar(const int n){
3     return (n%2==0);
4 }
5 int main(){
6     std::string parimpar;
7     parimpar=numeroPar(25)?"par":"impar";
8     std::cout<<"Es 25 par?: " << parimpar;
9 }

```



Contenido del tema

- 1 La función main
- 2 Parámetros con valor por defecto
- 3 Sobrecarga de funciones
- 4 Funciones inline
- 5 Variables locales static

Variables locales static

Variable local static

Es una variable local de una función o método que no se destruye al acabar la función, y que mantiene su valor entre llamadas.

- Se inicializa la primera vez que se llama a la función.
- Conserva el valor anterior en sucesivas llamadas a la función.
- Es obligatorio asignarles un valor en su declaración.

```

1 #include<iostream>
2 double cuadrado(double numero){
3     static int contadorLlamadas=1;
4     std::cout<<"Llamadas a cuadrado: "
5         <<contadorLlamadas<<std::endl;
6     contadorLlamadas++;
7     return numero*numero;
8 }
9 int main(){
10     for(int i=0; i<10; ++i)
11         std::cout<<i<<"^2 = "<<cuadrado(i)<<std::endl;
12 }

```

