Funciones recursivas

Fundamentos de Programación

1. La recursividad
   1. Qué es
   2. Funciones recursivas
2. La pila
   1. Qué es
   2. La pila y la recursividad
3. Múltiples casos y llamadas

La recursividad
¿Qué es?

Recursividad:
- Herramienta de programación.
- Técnica de diseño de funciones.

Función recursiva: está definida en términos de sí misma.

Ejemplo: $n! = \begin{cases} 1 & si \quad n = 0 \\ n*(n-1)! & si \quad n > 0 \end{cases}$

```
4! = 4 * 3! =
   = 4 * ( 3 * 2! ) =
   = 4 * ( 3 * ( 2 * 1! ) ) =
   = 4 * ( 3 * ( 2 * ( 1 * 0! ) ) ) =
   = 4 * ( 3 * ( 2 * ( 1 * ( 1 ) ) ) ) =
   = 4 * ( 3 * ( 2 * ( 1 ) ) ) =
   = 4 * ( 3 * ( 2 ) ) =
   = 4 * ( 6 ) =
   = 24
```


La recursividad
Ejemplo

Implementa una función para calcular el factorial de un número

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

```
int factorial(int n)
{
  int fact;
  if (n==0)
    fact=1;
  else
    fact=n*factorial(n-1);
  return fact;
}
```

```
int factorial(int n)
{
  assert(n>=0);
  …
}
```

Precondición: n>=0

La recursividad
Construcción de funciones recursivas

Claves para construir una función recursiva:

Caso general: La solución al problema se plantea en términos de sí mismo, aunque sobre un problema de menor tamaño (más sencillo de resolver).

Caso base: En algún momento el problema tendrá un tamaño suficientemente pequeño (muy sencillo) de manera que podamos calcular la solución sin aplicar más recursividad.

Peligro: recursividad infinita —
- *Si no existe caso base*
- *Si no llegamos nunca al caso base*

Ejemplos:

$n! = n*(n-1)!$

$n! = \begin{cases} 1 & si \quad n = 0 \\ n*(n+1)! & si \quad n > 0 \end{cases}$

La recursividad
Ejercicios

1. Implemente una función recursiva que calcule el máximo común divisor de dos números (mayores o iguales que cero) de acuerdo a la siguiente expresión:

$$mcd(x,y)=\begin{cases} x & si \quad y=0 \\ mcd(y,x\%y) & si \quad y>0 \end{cases}$$

2. Implemente una función recursiva que calcule la sumatoria de todos los elementos de un vector de enteros.

3. Implemente una función recursiva que calcule la raíz cuadrada de un número **a**. Para ello use la siguiente serie:

$$x_i=\begin{cases} 1 & si \quad i=0 \\ \frac{1}{2}\left(x_{i-1}+\frac{a}{x_{i-1}}\right) & si \quad i>0 \end{cases}$$

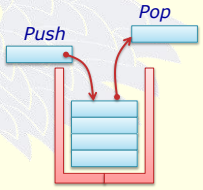
Se puede demostrar que: $x_i \to \sqrt{a}$ cuando $i \to \infty$

Funciones recursivas

Fundamentos de Programación

La pila
¿Cómo se organiza la memoria de un programa?

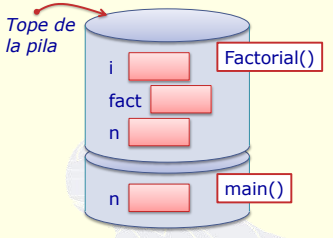
- Código (instrucciones) — Fijada en tiempo de compilación
- Memoria estática: Constantes, Globales, …
- Memoria dinámica / Montón / Heap — Gestionada en tiempo de ejecución
- Pila / Stack



En la pila se almacenan:
- Las variables locales de las funciones.
- Los parámetros formales de las funciones.
- Estado del programa (punto de retorno, …).

La pila
¿Cómo se organiza la memoria de un programa?

```
long int Factorial(int n)
{
  long int fact=n;
  int i=2;
  while (i<n) {
    fact *= i;
    i++;
  }
  return n==0?1:fact;
}
```




```
int main() {
  int n;
  cout << "Dime un número (>=0): ";
  cin >> n;
  cout << Factorial(n) << endl;
}
```

Cada función sólo puede acceder a los objetos que hay en su zona de pila.

La pila
Ejemplo: cómo funciona la pila

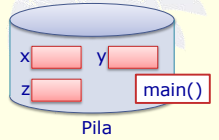
```
1  #include <iostream>
2  using namespace std;
3
4  int Maximo2(int a, int b) {
5    return (a>b) ? a : b;
6  }
7
8  int Maximo3(int a, int b, int c) {
9    int m1 = Maximo2(a,b);
10   int m2 = Maximo2(m1,c);
11   return m2;
12 }
13
14 int main() {
15   int x, y, z;
16   cout << "Dime 3 números: ";
17   cin >> x >> y >> z;
18   cout << Maximo3(x,y,z);
19 }
```

Al comienzo: pila vacía

La pila
Ejemplo: cómo funciona la pila

```
1  #include <iostream>
2  using namespace std;
3
4  int Maximo2(int a, int b) {
5    return (a>b) ? a : b;
6  }
7
8  int Maximo3(int a, int b, int c) {
9    int m1 = Maximo2(a,b);
10   int m2 = Maximo2(m1,c);
11   return m2;
12 }
13
14 int main() {
15   int x, y, z;
16   cout << "Dime 3 números: ";
17   cin >> x >> y >> z;
18   cout << Maximo3(x,y,z);
19 }
```

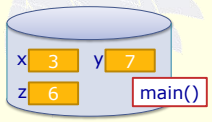
Comienza ejecución de: main()
- *Se crea su memoria de pila (en el tope).*
- *Se crean las variables locales.*



La pila
Ejemplo: cómo funciona la pila

```
1  #include <iostream>
2  using namespace std;
3
4  int Maximo2(int a, int b) {
5    return (a>b) ? a : b;
6  }
7
8  int Maximo3(int a, int b, int c) {
9    int m1 = Maximo2(a,b);
10   int m2 = Maximo2(m1,c);
11   return m2;
12 }
13
14 int main() {
15   int x, y, z;
16   cout << "Dime 3 números: ";
17   cin >> x >> y >> z;
18   cout << Maximo3(x,y,z);
19 }
```

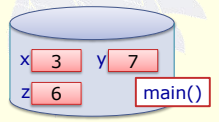
Se leen valores de x, y, z



La pila
Ejemplo: cómo funciona la pila

```
1  #include <iostream>
2  using namespace std;
3
4  int Maximo2(int a, int b) {
5    return (a>b) ? a : b;
6  }
7
8  int Maximo3(int a, int b, int c) {
9    int m1 = Maximo2(a,b);
10   int m2 = Maximo2(m1,c);
11   return m2;
12 }
13
14 int main() {
15   int x, y, z;
16   cout << "Dime 3 números: ";
17   cin >> x >> y >> z;
18   cout << Maximo3(x,y,z);
19 }
```

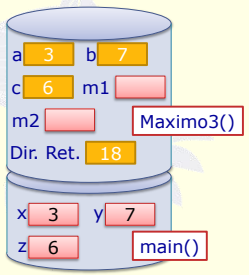
Llamada a función: Maximo3



La pila
Ejemplo: cómo funciona la pila

```
1  #include <iostream>
2  using namespace std;
3
4  int Maximo2(int a, int b) {
5    return (a>b) ? a : b;
6  }
7
8  int Maximo3(int a, int b, int c) {
9    int m1 = Maximo2(a,b);
10   int m2 = Maximo2(m1,c);
11   return m2;
12 }
13
14 int main() {
15   int x, y, z;
16   cout << "Dime 3 números: ";
17   cin >> x >> y >> z;
18   cout << Maximo3(x,y,z);
19 }
```

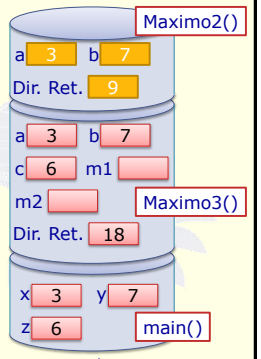
Llamada a Maximo3:
- *Crea memoria de pila.*
- *Crean objetos locales.*
- *Guarda estado prog.*



La pila
Ejemplo: cómo funciona la pila

```
1  #include <iostream>
2  using namespace std;
3
4  int Maximo2(int a, int b) {
5    return (a>b) ? a : b;
6  }
7
8  int Maximo3(int a, int b, int c) {
9    int m1 = Maximo2(a,b);
10   int m2 = Maximo2(m1,c);
11   return m2;
12 }
13
14 int main() {
15   int x, y, z;
16   cout << "Dime 3 números: ";
17   cin >> x >> y >> z;
18   cout << Maximo3(x,y,z);
19 }
```

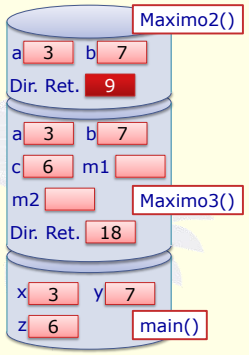
Llamada a Maximo2:



La pila
Ejemplo: cómo funciona la pila

```
1  #include <iostream>
2  using namespace std;
3
4  int Maximo2(int a, int b) {
5    return (a>b) ? a : b;
6  }
7
8  int Maximo3(int a, int b, int c) {
9    int m1 = Maximo2(a,b);
10   int m2 = Maximo2(m1,c);
11   return m2;
12 }
13
14 int main() {
15   int x, y, z;
16   cout << "Dime 3 números: ";
17   cin >> x >> y >> z;
18   cout << Maximo3(x,y,z);
19 }
```

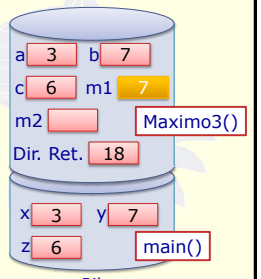
Finaliza Maximo2:

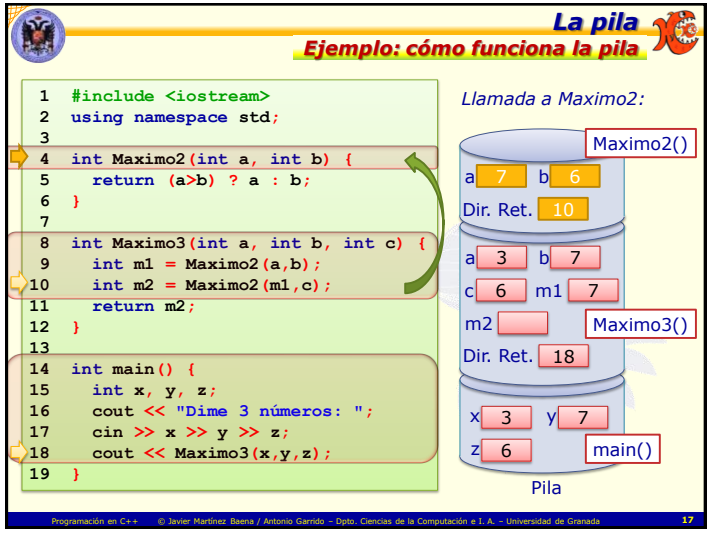


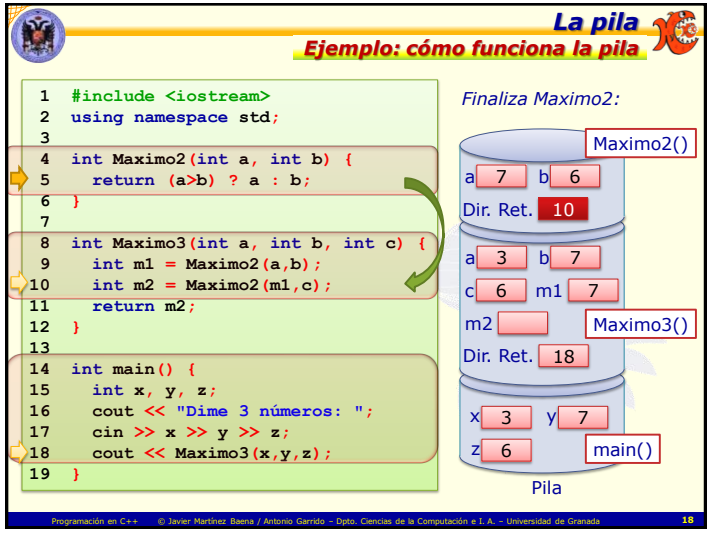
La pila
Ejemplo: cómo funciona la pila

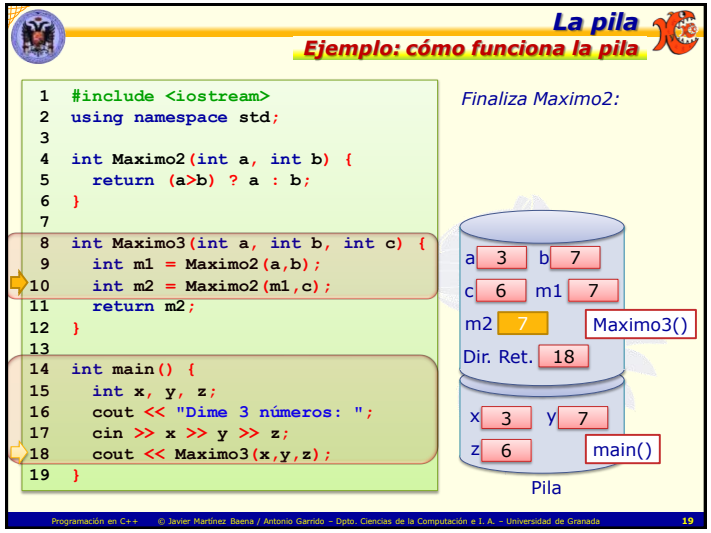
```
1  #include <iostream>
2  using namespace std;
3
4  int Maximo2(int a, int b) {
5    return (a>b) ? a : b;
6  }
7
8  int Maximo3(int a, int b, int c) {
9    int m1 = Maximo2(a,b);
10   int m2 = Maximo2(m1,c);
11   return m2;
12 }
13
14 int main() {
15   int x, y, z;
16   cout << "Dime 3 números: ";
17   cin >> x >> y >> z;
18   cout << Maximo3(x,y,z);
19 }
```

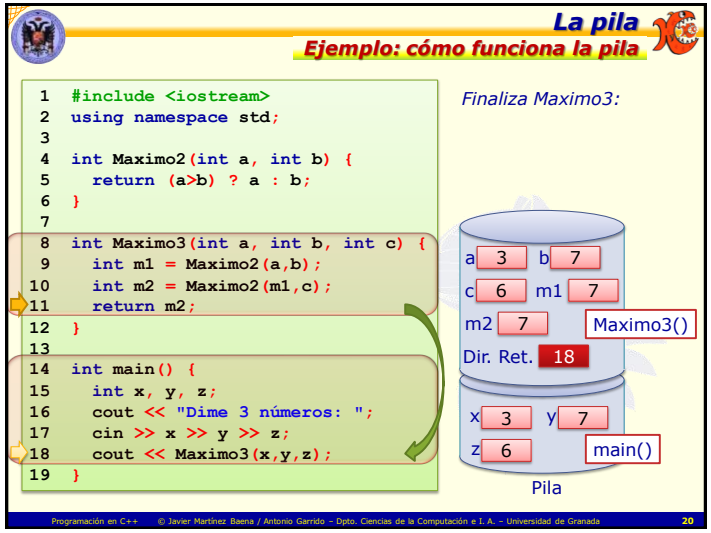
Finaliza Maximo2:




La pila
Ejemplo: cómo funciona la pila
1 #include <iostream>
2 using namespace std;
3
4 int Maximo2(int a, int b) {
5 return (a>b) ? a : b;
6 }
7
8 int Maximo3(int a, int b, int c) {
9 int m1 = Maximo2(a,b);
10 int m2 = Maximo2(m1,c);
11 return m2;
12 }
13
14 int main() {
15 int x, y, z;
16 cout << "Dime 3 números: ";
17 cin >> x >> y >> z;
18 cout << Maximo3(x,y,z);
19 }
Llamada a Maximo2:
Maximo2()
a 7 b 6
Dir. Ret. 10
a 3 b 7
c 6 m1 7
m2
Maximo3()
Dir. Ret. 18
x 3 y 7
z 6
main()
Pila
Programación en C++ © Javier Martínez Baena / Antonio Garrido – Dpto. Ciencias de la Computación e I. A. – Universidad de Granada 17


La pila
Ejemplo: cómo funciona la pila
1 #include <iostream>
2 using namespace std;
3
4 int Maximo2(int a, int b) {
5 return (a>b) ? a : b;
6 }
7
8 int Maximo3(int a, int b, int c) {
9 int m1 = Maximo2(a,b);
10 int m2 = Maximo2(m1,c);
11 return m2;
12 }
13
14 int main() {
15 int x, y, z;
16 cout << "Dime 3 números: ";
17 cin >> x >> y >> z;
18 cout << Maximo3(x,y,z);
19 }
Finaliza Maximo2:
Maximo2()
a 7 b 6
Dir. Ret. 10
a 3 b 7
c 6 m1 7
m2
Maximo3()
Dir. Ret. 18
x 3 y 7
z 6
main()
Pila
Programación en C++ © Javier Martínez Baena / Antonio Garrido – Dpto. Ciencias de la Computación e I. A. – Universidad de Granada 18

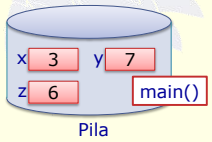

La pila
Ejemplo: cómo funciona la pila
1 #include <iostream>
2 using namespace std;
3
4 int Maximo2(int a, int b) {
5 return (a>b) ? a : b;
6 }
7
8 int Maximo3(int a, int b, int c) {
9 int m1 = Maximo2(a,b);
10 int m2 = Maximo2(m1,c);
11 return m2;
12 }
13
14 int main() {
15 int x, y, z;
16 cout << "Dime 3 números: ";
17 cin >> x >> y >> z;
18 cout << Maximo3(x,y,z);
19 }
Finaliza Maximo2:
a 3 b 7
c 6 m1 7
m2 7
Maximo3()
Dir. Ret. 18
x 3 y 7
z 6
main()
Pila
Programación en C++ © Javier Martínez Baena / Antonio Garrido – Dpto. Ciencias de la Computación e I. A. – Universidad de Granada 19


La pila
Ejemplo: cómo funciona la pila
1 #include <iostream>
2 using namespace std;
3
4 int Maximo2(int a, int b) {
5 return (a>b) ? a : b;
6 }
7
8 int Maximo3(int a, int b, int c) {
9 int m1 = Maximo2(a,b);
10 int m2 = Maximo2(m1,c);
11 return m2;
12 }
13
14 int main() {
15 int x, y, z;
16 cout << "Dime 3 números: ";
17 cin >> x >> y >> z;
18 cout << Maximo3(x,y,z);
19 }
Finaliza Maximo3:
a 3 b 7
c 6 m1 7
m2 7
Maximo3()
Dir. Ret. 18
x 3 y 7
z 6
main()
Pila
Programación en C++ © Javier Martínez Baena / Antonio Garrido – Dpto. Ciencias de la Computación e I. A. – Universidad de Granada 20

La pila
Ejemplo: cómo funciona la pila

```
1  #include <iostream>
2  using namespace std;
3
4  int Maximo2(int a, int b) {
5    return (a>b) ? a : b;
6  }
7
8  int Maximo3(int a, int b, int c) {
9    int m1 = Maximo2(a,b);
10   int m2 = Maximo2(m1,c);
11   return m2;
12 }
13
14 int main() {
15   int x, y, z;
16   cout << "Dime 3 números: ";
17   cin >> x >> y >> z;
18   cout << Maximo3(x,y,z);
19 }
```

Escribe en cout el valor 7 y finaliza main()



La pila
Ejemplo: cómo funciona la pila

```
1  #include <iostream>
2  using namespace std;
3
4  int Maximo2(int a, int b) {
5    return (a>b) ? a : b;
6  }
7
8  int Maximo3(int a, int b, int c) {
9    int m1 = Maximo2(a,b);
10   int m2 = Maximo2(m1,c);
11   return m2;
12 }
13
14 int main() {
15   int x, y, z;
16   cout << "Dime 3 números: ";
17   cin >> x >> y >> z;
18   cout << Maximo3(x,y,z);
19 }
```

Escribe en cout el valor 7 y finaliza main()

La pila
Ejemplo: la pila y la recursividad

```
int main() {
  int num;
  cout << "Dame un número (>=0):";
  cin >> num;
  cout << factorial(num) << endl;
}
```

¿ … y si la función es recursiva?

- *Comenzamos main()*
  - *Crear memoria de pila*
  - *Crear objetos (num)*
- *Leemos un valor de num (p. ej. 3)*
- *Llamamos a factorial(num)*

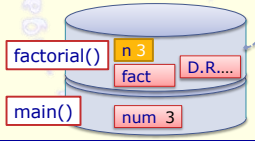
La pila
Ejemplo: la pila y la recursividad

```
int main() {
  int num;
  cout << "Dame un número (>=0):";
  cin >> num;
  cout << factorial(num) << endl;
}
```

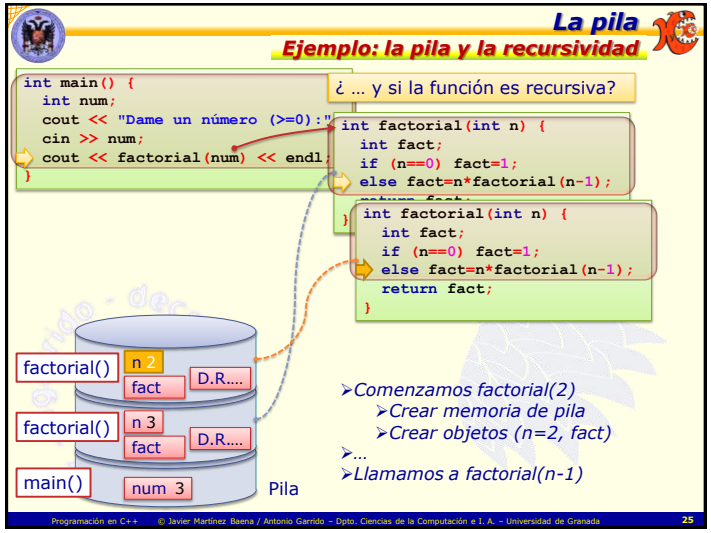
¿ … y si la función es recursiva?

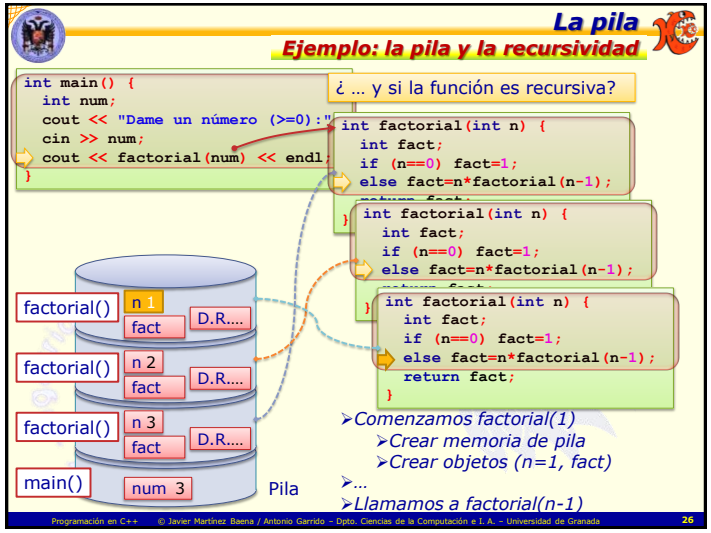
```
int factorial(int n) {
  int fact;
  if (n==0) fact=1;
  else fact=n*factorial(n-1);
  return fact;
}
```

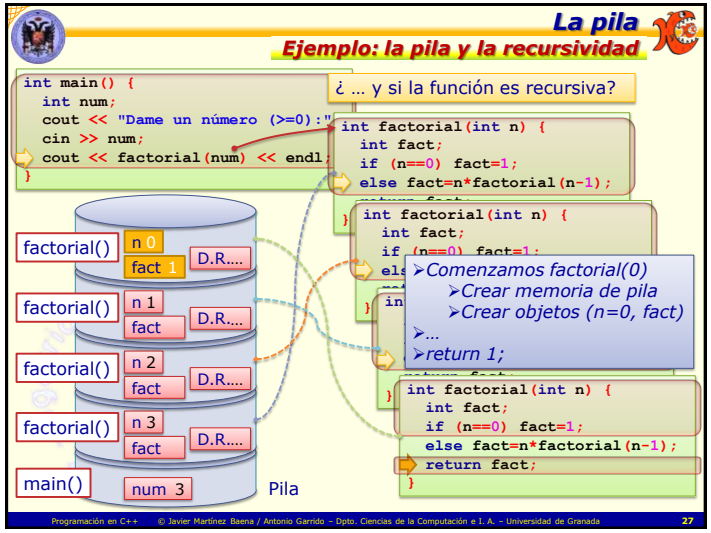
No se puede calcular la expresión hasta saber cuánto vale factorial(2)

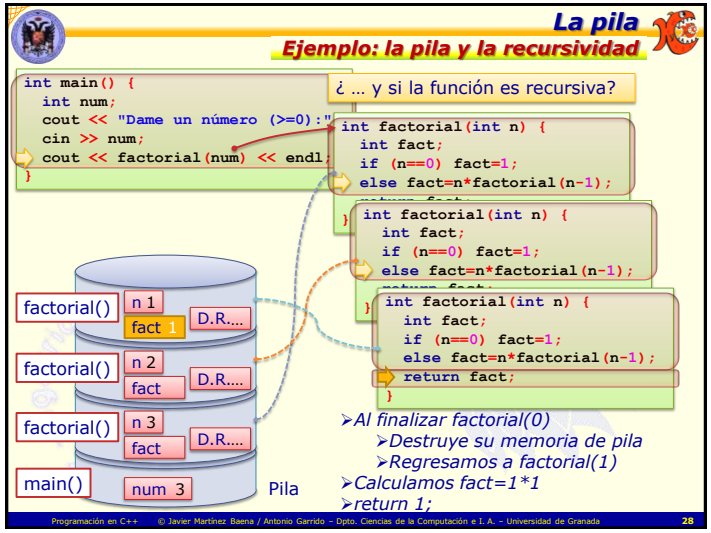


- *Comenzamos factorial(3)*
  - *Crear memoria de pila*
  - *Crear objetos (n=3, fact)*
- *…*
- *Llamamos a factorial(n-1)*


La pila
Ejemplo: la pila y la recursividad
¿ … y si la función es recursiva?
int main() {
int num;
cout << "Dame un número (>=0):";
cin >> num;
cout << factorial(num) << endl;
}
int factorial(int n) {
int fact;
if (n==0) fact=1;
else fact=n*factorial(n-1);
return fact;
}
factorial() n 2 fact D.R....
factorial() n 3 fact D.R....
main() num 3
Pila
➢Comenzamos factorial(2)
➢Crear memoria de pila
➢Crear objetos (n=2, fact)
➢…
➢Llamamos a factorial(n-1)
Programación en C++ © Javier Martínez Baena / Antonio Garrido – Dpto. Ciencias de la Computación e I. A. – Universidad de Granada 25


La pila
Ejemplo: la pila y la recursividad
¿ … y si la función es recursiva?
int main() {
int num;
cout << "Dame un número (>=0):";
cin >> num;
cout << factorial(num) << endl;
}
int factorial(int n) {
int fact;
if (n==0) fact=1;
else fact=n*factorial(n-1);
return fact;
}
factorial() n 1 fact D.R....
factorial() n 2 fact D.R....
factorial() n 3 fact D.R....
main() num 3
Pila
➢Comenzamos factorial(1)
➢Crear memoria de pila
➢Crear objetos (n=1, fact)
➢…
➢Llamamos a factorial(n-1)
Programación en C++ © Javier Martínez Baena / Antonio Garrido – Dpto. Ciencias de la Computación e I. A. – Universidad de Granada 26


La pila
Ejemplo: la pila y la recursividad
¿ … y si la función es recursiva?
int main() {
int num;
cout << "Dame un número (>=0):";
cin >> num;
cout << factorial(num) << endl;
}
int factorial(int n) {
int fact;
if (n==0) fact=1;
else fact=n*factorial(n-1);
return fact;
}
factorial() n 0 fact 1 D.R....
factorial() n 1 fact D.R....
factorial() n 2 fact D.R....
factorial() n 3 fact D.R....
main() num 3
Pila
➢Comenzamos factorial(0)
➢Crear memoria de pila
➢Crear objetos (n=0, fact)
➢…
➢return 1;
Programación en C++ © Javier Martínez Baena / Antonio Garrido – Dpto. Ciencias de la Computación e I. A. – Universidad de Granada 27

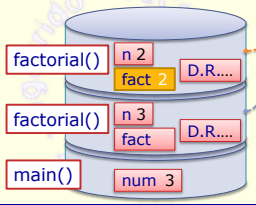

La pila
Ejemplo: la pila y la recursividad
¿ … y si la función es recursiva?
int main() {
int num;
cout << "Dame un número (>=0):";
cin >> num;
cout << factorial(num) << endl;
}
int factorial(int n) {
int fact;
if (n==0) fact=1;
else fact=n*factorial(n-1);
return fact;
}
factorial() n 1 fact 1 D.R....
factorial() n 2 fact D.R....
factorial() n 3 fact D.R....
main() num 3
Pila
➢Al finalizar factorial(0)
➢Destruye su memoria de pila
➢Regresamos a factorial(1)
➢Calculamos fact=1*1
➢return 1;
Programación en C++ © Javier Martínez Baena / Antonio Garrido – Dpto. Ciencias de la Computación e I. A. – Universidad de Granada 28

La pila
Ejemplo: la pila y la recursividad

```
int main() {
  int num;
  cout << "Dame un número (>=0):";
  cin >> num;
  cout << factorial(num) << endl;
}
```

¿ … y si la función es recursiva?

```
int factorial(int n) {
  int fact;
  if (n==0) fact=1;
  else fact=n*factorial(n-1);
  return fact;
}
```



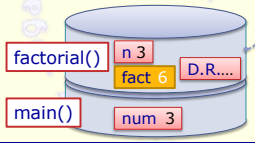
- *Al finalizar factorial(1)*
  - *Destruye su memoria de pila*
  - *Regresamos a factorial(2)*
- *Calculamos fact=2*1*
- *return 2;*

La pila
Ejemplo: la pila y la recursividad

```
int main() {
  int num;
  cout << "Dame un número (>=0):";
  cin >> num;
  cout << factorial(num) << endl;
}
```

¿ … y si la función es recursiva?

```
int factorial(int n) {
  int fact;
  if (n==0) fact=1;
  else fact=n*factorial(n-1);
  return fact;
}
```



- *Al finalizar factorial(2)*
  - *Destruye su memoria de pila*
  - *Regresamos a factorial(3)*
- *Calculamos fact=3*2*
- *return 6;*

La pila
Ejemplo: la pila y la recursividad

```
int main() {
  int num;
  cout << "Dame un número (>=0):";
  cin >> num;
  cout << factorial(num) << endl;
}
```

¿ … y si la función es recursiva?

- *Al finalizar factorial(3)*
  - *Destruye su memoria de pila*
  - *Regresamos a main()*
- *Escribimos en cout el valor 6*

La pila
Otro ejemplo

```
int funcion(int a, int b) {
  int result;
  if (a>=b)
    result = funcion(a-b,b)+1;
  else
    result = 0;
  return result;
}
```

Tiene sentido hacer múltiples return: menor consumo de memoria

```
int funcion(int a, int b) {
  if (a>=b)
    return funcion(a-b,b)+1;
  else
    return 0;
}
```

```
int funcion(int a, int b) {
  return (a>=b) ? funcion(a-b,b)+1 : 0;
}
```

… aunque en este caso podemos evitarlo

¿Qué devuelve esta función?

Funciones recursivas

Fundamentos de Programación

1. La recursividad
   1. Qué es
   2. Funciones recursivas
2. La pila
   1. Qué es
   2. La pila y la recursividad
3. Múltiples casos y llamadas

Múltiples casos y llamadas

Podemos tener varios casos base

Construir una función recursiva que comprueba si una cadena es o no un palíndromo

```
bool Palindromo(string cad)
{
  if (cad.size()<=1)
    return true;
  else if (cad.at(0)!=cad.at(cad.size()-1))
    return false;
  else
    return Palindromo(cad.substr(1,cad.size()-2));
}
```

Caso base 1

Caso base 2

Caso general (llamada recursiva)

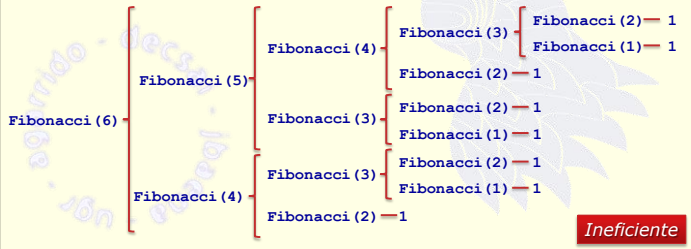
¿Eficiencia?

Múltiples casos y llamadas

Podemos tener varias llamadas recursivas

Construir una función recursiva que calcule el término n-ésimo de la serie de Fibonacci.

```
int Fibonacci(int n)
{
  return (n<=2) ? 1 : (Fibonacci(n-1)+Fibonacci(n-2));
}
```



Ineficiente

Múltiples casos y llamadas

Podemos tener varias llamadas recursivas

Sumar elementos de un vector de forma recursiva

```
int Suma(vector<int> v, int ini, int fin) {
  int sum;
  if (ini==fin)
    sum=v.at(ini);
  else
    sum=v.at(ini)+Suma(v,ini+1,fin);
  return sum;
}
```

```
int Suma(vector<int> v) {
  return Suma(v,0,v.size()-1);
}
```

¿Eficiencia?

Múltiples casos y llamadas

Podemos tener varias llamadas recursivas

Sumar elementos de un vector de forma recursiva

```
int Suma(vector<int> v, int ini, int fin)
{
  int sum;
  if (ini==fin)
    sum=v.at(ini);
  else {
    int centro = (ini+fin)/2;
    sum=Suma(v,ini,centro)+Suma(v,centro+1,fin);
  }
  return sum;
}
```

¿Eficiencia?

Funciones recursivas

Ejercicios

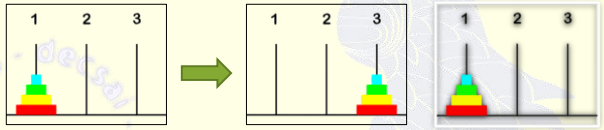
1. Implementa una función recursiva que escriba un string invertido en cout.
2. Implementa una función recursiva que permita convertir un número (en base 10) a otra base cualquiera. Dicha función escribirá el resultado en cout.
3. Implementa una función recursiva que permita calcular el elemento menor de un vector. La función debe devolver la posición de dicho elemento.
4. Implementa una función que ordene un vector de enteros por el método de selección de manera recursiva.

Funciones recursivas

Ejercicios (2)

5. Implementa una función recursiva que escriba la secuencia de movimientos necesarios para resolver el problema de las torres de Hanoi con N discos.

- Disponemos de 3 palos (torres).
- Disponemos de N discos de diferentes tamaños.
- Inicialmente: los discos forman una pirámide en el primer palo.



Estado inicial (N=4) *Estado final*

Reglas para mover discos:
- Los discos se mueven de uno en uno.
- Un disco sólo puede colocarse sobre otro mayor que él.

Funciones recursivas

Ejercicios (3)

6. Implementa el algoritmo de búsqueda binaria de forma recursiva.
7. Implementa una función recursiva para sumar los dígitos de un número entero.
8. Implementa una función recursiva que permita colocar 8 reinas en un tablero de ajedrez sin que se maten entre ellas. Una vez resuelto: generaliza el problema para N reinas en un tablero de NxN.