Fundamentals of Programming I

4A More About Subprograms

SUPPLEMENT

Grado en Ingeniería Informática

Luis Hernández Yáñez
Facultad de Informática
Universidad Complutense

Index

Files as Parameters	499
`main()` Function	502
Implicit Arguments	505
Subprogram Overload	509

Files as Parameters

Luis Hernández Yáñez

Files as Parameters

```
#include <iostream>
#include <fstream>
using namespace std;

void summation_file(ifstream &file, double &sum);

int main() {
   double result;
   ifstream file;
   file.open("data.txt");
   if (!file.is_open())
      cout << "Couldn't open the file!" << endl;
   else {
      summation_file(file, result)
      cout << "Sum = " << result << endl;
      file.close();
   }

   return 0;
}
```

Luis Hernández Yáñez

BY NC SA

Files as Parameters

```
void summation_file(ifstream &file, double &sum) {
   double data;

   sum = 0;
   file >> data;

   while (data != -1) {
      sum = sum + data;
      file >> data;
   }
}
```

👀 Files are always passed by reference

Luis Hernández Yáñez

Fundamentals of Programming I

main() Function

Luis Hernández Yáñez

Parameters of `main()`

Communication with the Operating System

Optional parameters of `main()` function:

```
int main(int argc, char *argv[])
```

To obtain data provided when executing the program:

`C:\>test` *`cad1 cad2 cad3`*

Executes `test.exe` with three arguments (strings)

Parameters of `main()`:

- `argc`: number of arguments provided
  4 in the example (first one: program name with path)
- `argv`: array with the strings provided as arguments

Luis Hernández Yáñez

What `main()` Returns

How was the execution?

`main()` function returns a termination code to the O.S.

- `0`: *Everything OK*
- Different than `0`: *There has been an error!*

If execution reaches the end of `main()`, then everything OK:

```
    ...
    return 0; // End of the program
}
```

Luis Hernández Yáñez

Implicit Arguments

Luis Hernández Yáñez

Default values for parameters passed by value

Default value for a parameter:

After an = following the name of the parameter: `void proc(int i = 1);`

If no argument is provided the parameter gets that value

`proc(12);` `i` gets the explicit value `12`

`proc();` `i` gets the implicit value (`1`)

There can be implicit arguments only for final parameters:

```
void p(int i, int j = 2, int k = 3); // CORRECT
void p(int i = 1, int j, int k = 3); // INCORRECT
```

Once an implicit argument is assigned, all the following parameters must have also an implicit argument

Luis Hernández Yáñez

Implicit Arguments

Parameters and implicit arguments

```
void p(int i, int j = 2, int k = 3);
```

Arguments are copied to parameters from first to last

→ Those with no argument will get the implicit value

```
void p(int i, int j = 2, int k = 3);
  p(13); // i gets 13, j and k their implicit values
  p(5, 7); // i gets 5, j gets 7 and k its implicit value
  p(3, 9, 12); // i gets 3, j gets 9 and k gets 12
```

Implicit arguments are declared in the prototype (preferably) or in subprogram's heading, but NOT in both

Luis Hernández Yáñez

Example

By default, + sign

By default, Δ is 1

$$f(x,y) = \pm\Delta\frac{x}{y}$$

```
#include <iostream>
using namespace std;

double f(double x, double y, int sign = 1, double delta = 1.0);

int main() {
   double x, y;
   cout << "X = ";
   cin >> x;
   cout << "Y = ";
   cin >> y;
   cout << "By default sign and delta: " << f(x, y) << endl;
   cout << "sign - and by default delta: " << f(x, y, -1) << endl;
   cout << "Explicit sign and delta: " << f(x, y, 1, 1.25) << endl;

   return 0;
}

double f(double x, double y, int sign, double delta) {
   return sign * delta * x / y;
}
```

We can't leave `sign` by default and use an explicit `delta`

Luis Hernández Yáñez

Subprogram Overload

Luis Hernández Yáñez

Same name, different parameters

Functions or procedures with same name but different parameters:

```
int abs(int n);
double abs(double n);
long int abs(long long int n);
```

The one with the type of parameter will be executed :

```
abs(13)    // int argument --> first function
abs(-2.3)  // double argument --> second function
abs(3L)    // long long int argument --> third function
```

To indicate that it's a `long long int` literal, not an `int` one

Luis Hernández Yáñez

Subprogram Overload

ex.cpp

```
#include <iostream>
using namespace std;

void exchange(int &x, int &y);
void exchange(double &x, double &y);
void exchange(char &x, char &y);

void exchange(int &x, int &y) {
   int tmp;
   tmp = x;
   x = y;
   y = tmp;
}

void exchange(double &x, double &y) {
   double tmp;
   tmp = x;
   x = y;
   y = tmp;
}

void exchange(char &x, char &y) {
   char tmp;
   tmp = x;
   x = y;
   y = tmp;
}

int main() {
   int i1 = 3, i2 = 7;
   double d1 = 12.5, d2 = 35.9;
   char c1 = 'a', c2 = 'b';
   cout << i1 << " - " << i2 << endl;
   cout << d1 << " - " << d2 << endl;
   cout << c1 << " - " << c2 << endl;
   exchange(i1, i2);
   exchange(d1, d2);
   exchange(c1, c2);
   cout << i1 << " - " << i2 << endl;
   cout << d1 << " - " << d2 << endl;
   cout << c1 << " - " << c2 << endl;
   return 0;
}
```


Promote Open Culture!

Creative Commons License

Attribution
You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Non commercial
You may not use this work for commercial purposes.

Share alike
If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

Click on the upper right image to learn more...