Subprograms: procedures and functions

- Programming languages, in particular C++, not only provide a set of basic operations and statements, but also a means to define our own operations and statements.

- We call the operations and statements that we define functions and procedures, respectively.

- Procedures and functions (subprograms) may have parameters. These represent the objects from our program that are used in the subprogram.

Subprograms

• Functions are defined as follows:

```c
int times(int x, int y) {
    int p = 0;
    while (y > 0) {
        if (y%2 == 0) {
            y = y/2;
            x = x*2;
        } else {
            p = p + x;
            y = y - 1;
        }
    }
    return p;
}
```

• Procedures and functions (subprograms) may have parameters. These represent the objects from our program that are used in the subprogram.
Subprograms

- Procedures are defined similarly, but without delivering any result:

```c
void factors(int x) {
    // Code
}
```

Subprograms

- Subprogram definitions may appear before or after the main program.

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

int f() {
    // Code for f
}

int main() {
    // Code for the main program
}

void p(int a) {
    // Code for p
}
```

Subprograms

- A function can only be used if previously declared. A function can be declared and used before its code is defined.

```c
double volume_sphere(double radius);

void some_geometry() {
    ...
    double V = volume_sphere(1.0);
    ...
}

double volume_sphere(double radius) {
    return 4*Pi*radius*radius*radius/3;
}
```
Subprograms

Once a subprogram has been declared, it can be used.

- Functions are used as operations within expressions.
- Procedures are used as statements.

- Evaluating the expression

\[ \text{times}(3, \ i + 2) + 1 \]

means executing the code of `times` over the arguments 3 and \( i+2 \) and then adding 1 to the result returned by the function.

- Evaluating the statement

\[ \text{factors}(i); \]

means executing the code of `factors` over the argument \( i \).

Appropriate use of subprograms:

- Increases *readability*: programs are better structured and easier to understand.
- Enables the use of *abstraction* in the program design.
- Facilitates *code reuse*.
Subprograms: parameter passing

• When a subprogram is called, the arguments are passed to the subprogram, so that its code can be executed:

\[
times(3, i + 2) + \ldots
\]

\[
\text{int times(int } x, \text{ int } y) \{ \ldots \}
\]

• Each argument must have the same type as its corresponding parameter.

Subprograms: parameter passing

• In general, any expression can be the argument of a subprogram:

\[
double \text{ maximum(double } a, \text{ double } b); \ldots
\]

\[
z = \text{ maximum}(x, y); \ldots
\]

\[
r = \text{ maximum}(3, \text{ gcd}(s - 4, i) + \text{ alpha}); \ldots
\]

\[
m = \text{ maximum}(x, \text{ maximum}(y + 3, 2 \times \pi \times \text{radius}));
\]

Subprograms: parameter passing

• An object (a variable) is associated with a value and a memory location. In C++, there are two methods for parameter passing:

  – Passing the value (call-by-value). This is denoted by just declaring the type and the name of the parameter.

  \[
  \text{void } p(\text{int } x, \text{ int& } y) \{ \ldots \}
  \]

  Call-by-value

  Call-by-reference

  – Passing the memory location (call-by-reference). This is denoted by adding the symbol & next to the parameter type.

  \[
  \text{int times(int } x, \text{ int } y) \{ \ldots \}
  \]

  is equivalent to:

  \[
  \text{int times(int } x, \text{ int } y) \{ \ldots \}
  \]

  \[
  \text{times}(3, i + 2)
  \]

  \[
  \text{int } p = 0;
  \]

  \[
  \ldots
  \]

Call-by-value

Call-by-reference

Subprograms: parameter passing

• Call-by-value makes a copy of the argument at the beginning of the subprogram. It is equivalent to having, a statement that assigns the value of each argument to the corresponding parameter:
Subprograms: parameter passing

- The effect of call-by-reference is that the parameter becomes the same object (variable) as the argument, i.e., the parameter becomes an alias of the argument.

- Example: procedure to swap the value of two variables

```c
void exchange(int& x, int& y) {
    int z = x;
    x = y;
    y = z;
}
```

- Example: procedure to swap the value of two variables

```c
exchange(a, b)
```

Is equivalent to having:

```c
void exchange(int& x, int& y) {
    int z = a;
    a = b;
    b = z;
}
```

- Use call-by-value to pass parameters that must not be modified by the subprogram.

- Use call-by-reference when the changes made by the subprogram must affect the variable to which the parameter is bound.

- In some cases, call-by-reference is used to avoid copies of large objects, even though the parameter is not modified.

Warning: we do not recommend the use of non-void functions with reference parameters in this course.
Subprograms: parameter passing

• To define a subprogram that, given two integers \( x \) and \( y \), returns their quotient and remainder, we can write:

```c
void div(int x, int y, int& q, int& r) {
    q = x/y;
    r = x%y;
}
```

• For instance, if the parameters would be passed by reference in the function \texttt{times} , after the execution of the statements:

```c
int a = 4;
int b = 2;
int c = times(a, b);
```

the value of \( a \) would be 0 and the value of \( b \) would be 8 (and the value of \( c \) would be 8).

Subprograms: parameter passing

• For instance, after the definition:

```c
void exchange(int x, int y) {
    int z = x;
    x = y;
    y = z;
}
```

the statement \texttt{exchange(a,b)} would not have any effect on \( a \) and \( b \).

Subprograms: parameter passing

• A call-by-value parameter can receive any expression as an argument.

```c
void exchange(int x, int y) {
    int z = x;
    x = y;
    y = z;
}
```

• A call-by-reference parameter can only be bound to variables.

```c
void exchange (int& a, int& b);
...
exchange(a, b + 4);
```

Incorrect parameter passing.
• Design a function that calculates the LCM of two numbers. Assume that we can use a function \texttt{gcd(a,b)} that calculates the greatest common divisor.

\begin{verbatim}
// Pre: a>0, b>0
// Returns the LCM of a and b
int lcm(int a, int b) {
    return (a/gcd(a,b))*b;
}
\end{verbatim}