C++

Definition and declaration

Definition

- A **definition** introduces the type and name of a variable or a function plus allocates the space for the thing being declared
- A variable definition reserves memory for storing the variable value
- Example:

```
int func() {
  int a;
//…
}
```

- “int a” above is a definition: it introduces the type and name of the variable and allocates memory on the stack for the object (on x86-32 bits, it would allocate 4 bytes)

Definition

- A function definition corresponds to the function signature (return type and number and type of arguments) as well as the function body (code)
- A function definition allocates the space necessary for storing the function code in memory
- Example:
```
int func(int x) {
  return x + 1;
}
```
- A (variable, function, member function) definition should appear only **once** in a program

Declaration

- Declaring a variable or a function tells the compiler that the variable (or the function) exists and has already been defined somewhere else
- It does not allocate any storage
- A variable declaration has the syntax: "**extern** *type name*;"
  For example:
  extern int a; // tells the compiler that a is defined somewhere else
- A function declaration has the syntax: "[extern] type func_name(list of arguments);"
  For example:
  extern void my_func(Obj& o); // extern is optional
- Note:
  - In a function declaration there is no code (function body)
  - The extern keyword is optional

Definition and declaration

- Declaration unlike definition does not allocate memory
- There can be several declarations (corresponding to a name) unlike definition (only one definition)
- The connection between declared variables (or functions) and their definition is done when the object files are linked by the linker.

Example - definition

```
int g_length = 10;  // (global) variable definition

int func (int n) {     // function definition
  int sum = 0;
  for (int i = 0; i < n; i++) {
    sum += i;
  }
  return sum;
}
```

Example – declaration and definition

```
#include <iostream>

extern int g_length;  // variable declaration

int func (int n);         // no body so function declaration

int main () {
  std::cout << func(g_lenth) << std::endl;

  int a = 5;              // definition
  int b;                  // another definition
}
```

Declaration and header files

- Header files are used for:
  - External object declarations (definitions are usually in the associated .cpp file)
  - Function declarations (definitions are usually in the associated .cpp file)
  - Type and class definitions (member functions definitions are usually in the associated .cpp file unless declared inline)
  - Inline function definitions
- Using header files is good practice because:
  - It guarantees that all files including these headers will have a consistent declaration of global objects or functions
  - If a declaration is updated then only one change in the header file is needed

Inline functions

- When the compiler inline-expands a function call, the function's code gets inserted into the caller's code
- The concept is similar to macros in C but it is safer (for example there are additional type checks)
- Inline functions may also *sometime* improve speed

Inline functions

- Declarations of an inline function is same as for a normal function:
  void f(int i);
- Definition of an inline function is prefixed by the keyword **inline**
- Note: it is important that the function definition is placed in an header file. If you put your definition in a .cpp file and try to call the inline function in another .cpp file you will get an error from the linker.

Inline member functions

- Similar to inline functions
- Two ways for defining an inline member function:

```
// A.h
class A {
public:
  // f will be inlined
  void f(int i) {
    // body of the function
  }
};
```

```
// A.h
class A {
public:
 void f(int i);
  // rest of the class
};

inline void A::f(int i) {
  // body of the function
}
```

Inline member functions

- Note: it is important that the inline method definition is placed in an header file. If you put your definition in a .cpp file and try to call the inline method in another .cpp file you will get an error from the linker.