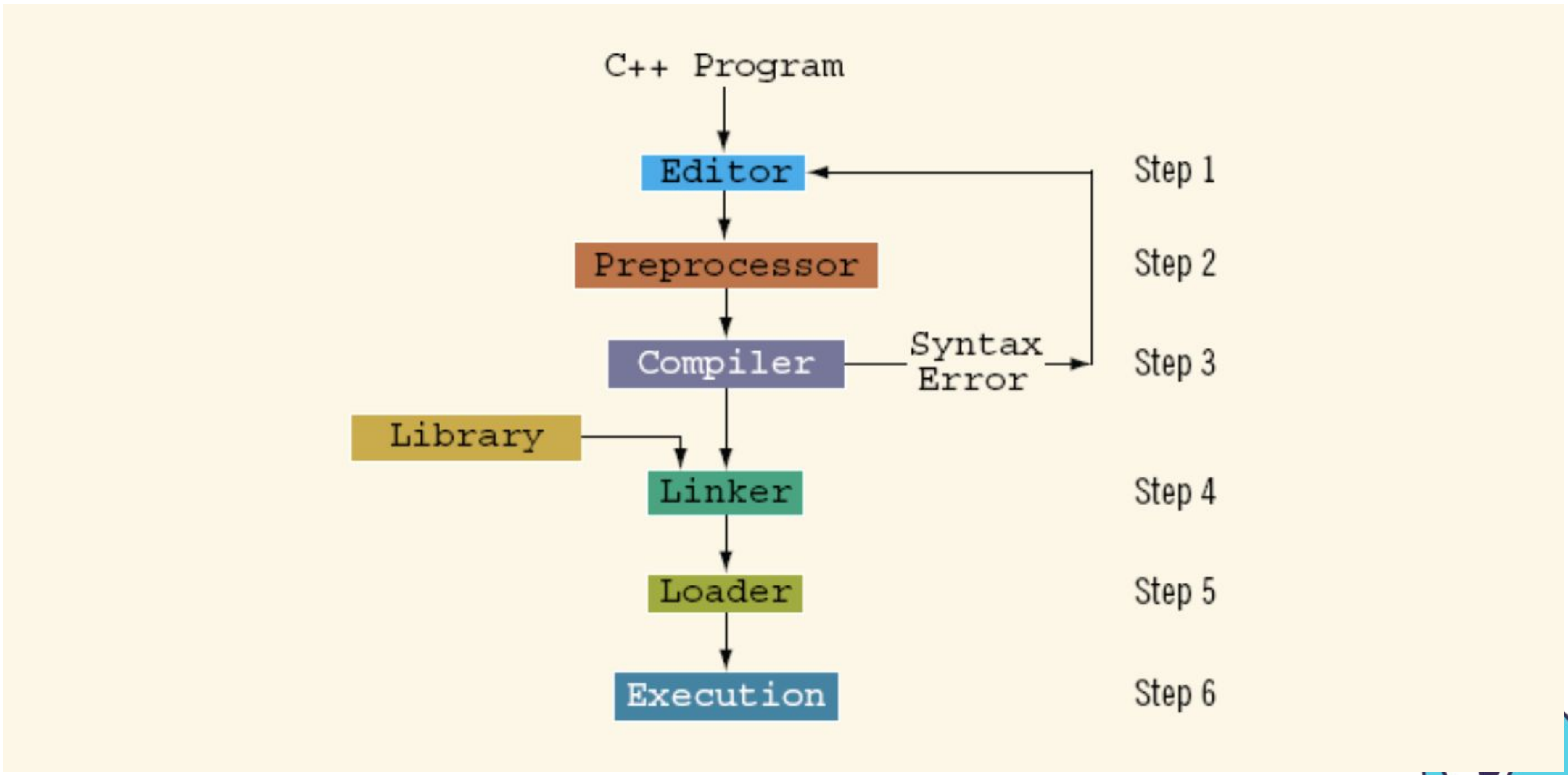
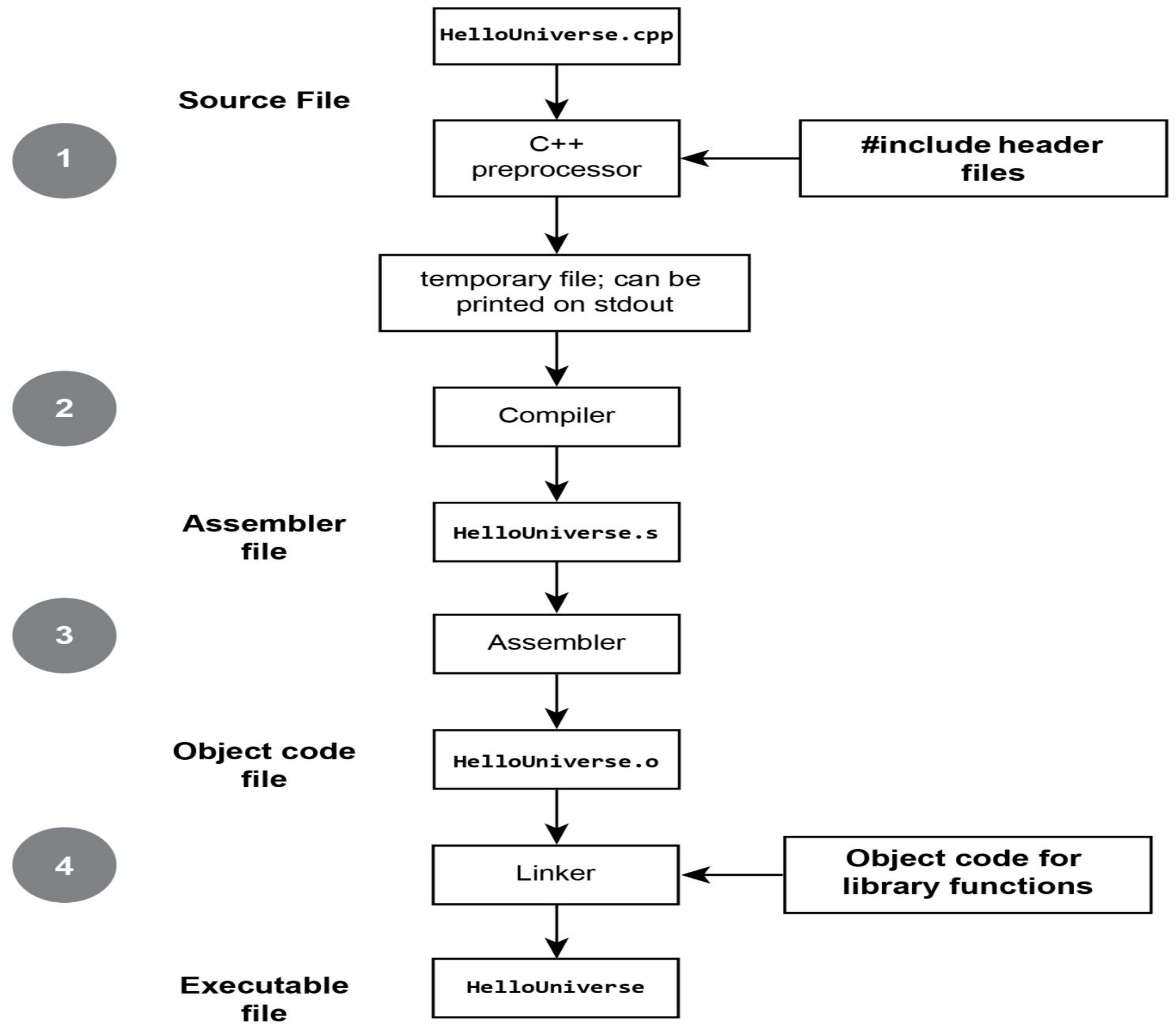
C++ Compilation Model

- Inherited from C
- Multi-step compilation model
- Preprocessing
- Compiling
- Assembling
- Linking
- Loading
- Executing

C++ Compilation Model



C++ Compilation Model



C++ Preprocessing

- Handles the preprocessor directives, like #include and #define
- Replaces #include directives with the content of the respective files
- Replacement of macros (#define)
- Different portions of text depending of #if, #ifdef and #ifndef directives
- Preprocessor produces a single output
- Stream of tokens resulting from the transformations described above
- g++ -E source.cpp -o preprocessed_source.i

C++ Compiling

- Performed on each output of the preprocessor
- Compiler parses the pure C++ source code and converts it into assembly code
- You don't need to recompile everything if you only change a single file
- g++ -S preprocessed_source.i -o compiled_source.s

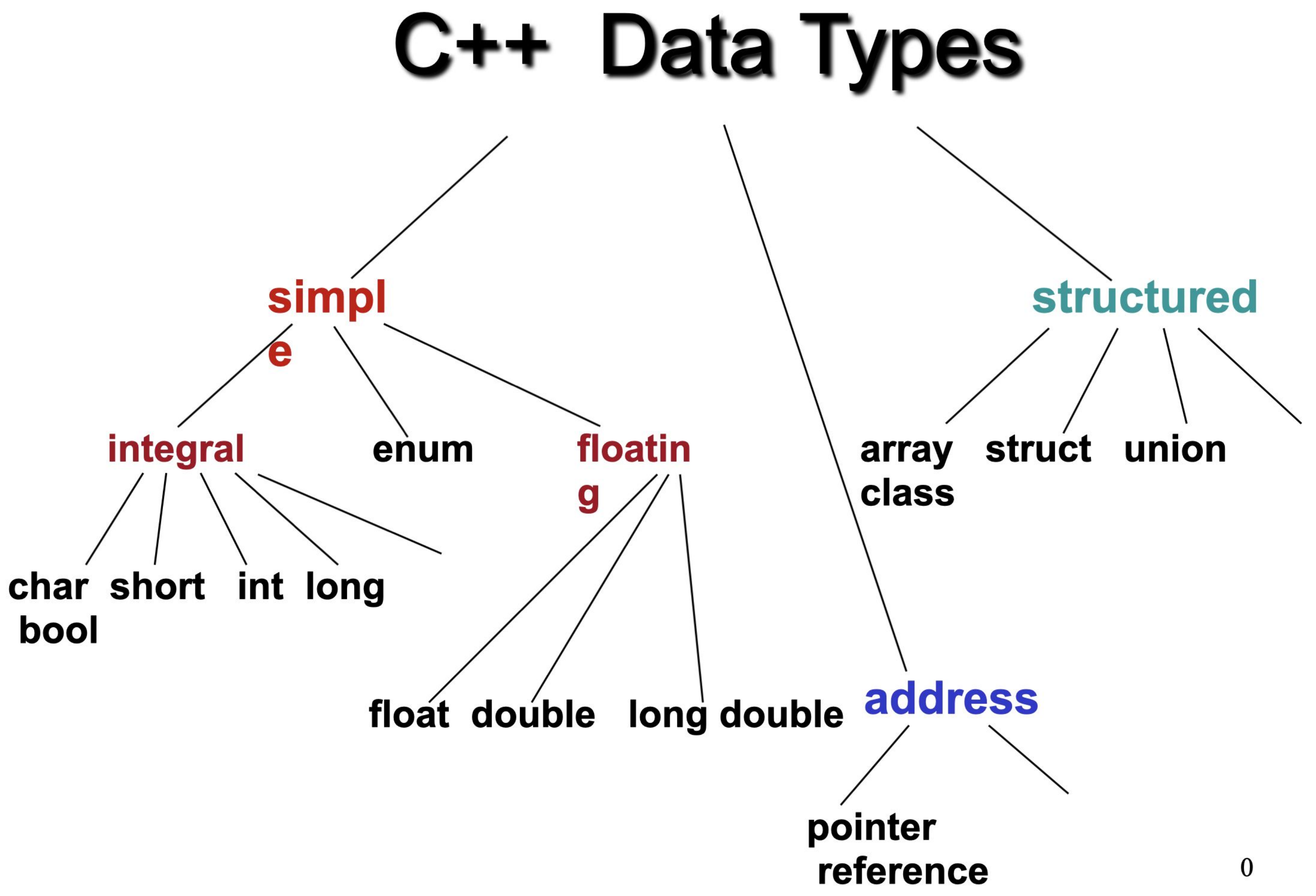
C++ Assembling

- Assembles that code into machine code producing actual binary file
- Object file contains the compiled code (in binary form) of the symbols
- Symbols in object files are referred to by name
- Object files can refer to symbols that are not defined
- Produced object files can be put in special archives called static libraries
- as compiled_source.s -o assembled_code.o

C++ Linking

- Produces the final compilation output from the object files
- Output can be either a shared (or dynamic) library or an executable
- Links all the object files by replacing the references to undefined symbols with the correct addresses
- Each of these symbols can be defined in other object files or in libraries
- If they are defined in libraries, you need to tell the linker about them
- The most common errors are missing definitions or duplicate definitions
- g++ assembled_code.o -o executable

C++ Data Types


C++ Data Types
simpl
e
structured
integral
enum
floatin
g
array
class
struct
union
char short int long
bool
float double long double
address
pointer
reference
0