CHAPTER 7
HIGHER LEVEL LANGUAGES - C/C++

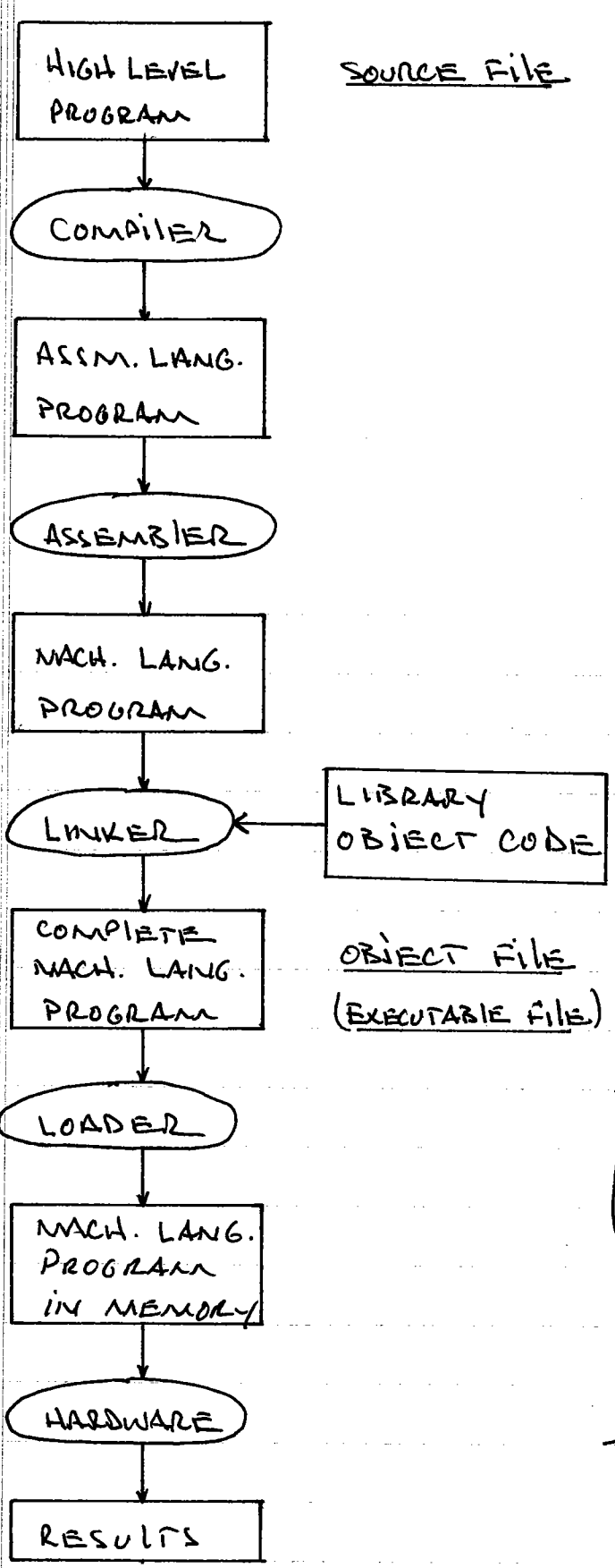
A TRANSLATOR FOR A HIGH LEVEL LANGUAGE IS CALLED A <u>COMPILER</u>. COMPILERS USUALLY MAKE SEVERAL PASSES OVER THE CODE, COMPLETING THEIR TASK IN STAGES.

THE FIRST STAGE IS TO TRANSLATE THE HIGH LEVEL PROGRAM INTO ASSEMBLY LANGUAGE, THEN INTO MACHINE LANGUAGE.

OFTEN A GROUP OF HIGH LEVEL INSTRUCTIONS IS SO USEFUL THAT WE WISH TO MAKE THEM AVAILABLE FOR REUSE BY STORING THEM IN A <u>CODE LIBRARY</u>. TO USE THESE <u>LIBRARY ROUTINES</u> WE PLACE CERTAIN COMMANDS IN OUR HIGH LEVEL PROGRAM TO ACCESS THEM. A PIECE OF SYSTEM SOFTWARE CALLED A <u>LINKER</u> INSERTS THE REQUESTED OBJECT CODE FROM THE CODE LIBRARIES INTO THE OBJECT CODE FOR OUR PROGRAM.

OFTEN THIS ENTIRE PROCESS: HIGH LEVEL PROGRAM TO ASSEMBLY LANGUAGE TO MACHINE LANGUAGE TO LINKER IS AUTOMATED BY A SINGLE SYSTEM COMMAND. (IN UNIX g++.) WE SOMETIMES CALL THIS PROCESS COMPILATION, ALTHOUGH STRICTLY SPEAKING THE COMPILER TRANSLATES ONLY FROM HIGH LEVEL TO ASSEMBLY LANGUAGE.

HIGH LEVEL LANGUAGE TRANSLATION



```
////////////////////////////////////////////////////////////////
//  Name
//  Student ID#
//  CS10-01
//  Date
//  Lab Assignment 1
//  File: convert.cpp
//  Description: convert temperature from Fahrenheit to Celsius
//  Compile: g++ -o convert convert.cpp
////////////////////////////////////////////////////////////////

#include<iostream>
using namespace std;

#define CONVERSION_FACTOR  (5.0/9.0)

int main(void)
{
  double fahrenheit,   // temperature in Fahrenheit
         celsius;      // temperature in Celsius

  // Get temperature in Fahrenheit
  cout << "Enter the temperature in Fahrenheit: ";
  cin  >> fahrenheit;

  // Convert to Celsius
  celsius = (fahrenheit-32)*CONVERSION_FACTOR;

  // Print temperature in Celsius
  cout << fahrenheit << " degrees Fahrenheit is equivalent to "
       << celsius  << " degrees Celsius.\n";

  return(0);
}
```

Compile with:

```
%g++ -o convert convert.cpp
```

Run with:

```
%convert
Enter the temperature in Fahrenheit: 75
75 degrees Fahrenheit is equivalent to 23.8889 degrees Celsius.
%
```

THE C++ PROGRAMMING LANGUAGE

WE REVIEW convert.CPP FROM PA1.

COMMENTS

ANYTHING ON A LINE FOLLOWING // IS A COMMENT AND IS IGNORED BY THE COMPILER. THE OLDER C STYLE /* COMMENTS */ SYNTAX IS ALSO ACCEPTED IN C++.

GOOD PROGRAMMING STYLE DEMANDS THAT YOU USE CLEAR, CONCISE, TO-THE-POINT COMMENTS. ALL PROGRAMS SHOULD START WITH A COMMENT BLOCK GIVING: FILE NAME, DESCRIPTION OF OPERATION, COMPILE COMMAND LINE.

FOLLOW THE PATTERN IN convert.CPP.

PREPROCESSOR COMMANDS

ALL C/C++ COMPILERS BEGIN WITH AN INITIAL PASS BY THE PREPROCESSOR.

PREPROCESSOR COMMANDS ARE INDICATED BY THE # SYMBOL AS THE FIRST CHARACTER ON A LINE.

#include <iostream>

DIRECTS THE COMPILER TO INSERT THE STANDARD LIBRARY HEADER FILE iostream.h, WHICH CONTAINS DEFINITIONS OF OBJECTS USED FOR INPUT AND OUTPUT.

BOTH C AND C++ HAVE RELATIVELY FEW BUILT IN COMMANDS. (C++ HAS 62 RESERVED WORDS, C HAS A MERE 32.) MANY COMMON OPERATIONS (SUCH AS I/O) ARE DEFINED IN SO CALLED STANDARD LIBRARIES.

SOME COMMON HEADER FILES:

iostream	C++
cstdio	C/C++
cstdlib	C/C++
cstring	C/C++
cmath	C/C++

THERE ARE MANY MORE.

#define CONVERSION_FACTOR (5.0/9.0)

DIRECTS THE COMPILER TO DO A LITERAL TEXT SUBSTITUTION OF (5.0/9.0) FOR

CONVERSION_FACTOR, A CONSTANT MACRO.

CONSTANT MACROS ADD TO PROGRAM CLARITY AND MAINTAINABILITY.

FUNCTION MAIN

EVERY C/C++ PROGRAM MUST CONTAIN A FUNCTION CALLED MAIN.

```
int main(void)
```

IS THE FUNCTION HEADING, AND EVERYTHING BETWEEN BRACES, { TO }, CONSTITUTES THE FUNCTION BODY. (MORE ON FUNCTIONS LATER.)

THE GENERAL FORM OF FUNCTION MAIN IS:

```
int main(void)
{
      // VARIABLE DECLARATIONS
         :

      // EXECUTABLE STATEMENTS
         :

      return (0);
}
```

VARIABLE DECLARATIONS ARE STATEMENTS WHICH DIRECT THE COMPILER TO SET ASIDE SPACE IN MEMORY FOR DATA VALUES WHICH WILL BE REFERED TO BY SYMBOLIC VARIABLE NAMES.

WE WILL WORK WITH FOUR BASIC DATA TYPES: int, double, char, AND bool. THESE TYPES ARE USED TO REPRESENT SIGNED INTEGERS, REAL NUMBERS, CHARACTERS, AND THE BOOLEAN VALUES TRUE/FALSE RESPECTIVELY.

Ex.
```
int count;
double sum, average;
char first,
     middle,
     last;
bool done = false;
```

NOTE: All DECLARATIONS AND EXECUTABLE STATEMENTS IN C/C++ MUST END IN A SEMICOLON;

EXTRA WHITE-SPACE CHARACTERS (i.e. SPACE, TAB, NEWLINE) ARE IGNORED BY THE COMPILER.

C++/C IS A FREE FORMAT LANGUAGE, i.e. IT DOES NOT MATTER WHERE THINGS ARE PLACED ON A LINE.

AN IDENTIFIER IS THE SYMBOLIC NAME WE GIVE TO A DATA VALUE. A LEGAL C/C++ IDENTIFIER MAY CONSIST OF

1.) LETTERS a, b .., A, B, ...
2.) DIGITS 1, 2, ..
3.) UNDERSCORES _

A C/C++ IDENTIFIER MAY NOT BEGIN WITH A DIGIT, AND MAY NOT BE A RESERVED WORD.

EX. VALID IDENTIFIERS:
happy, Happy, happy1, happy_happy

EX. INVALID IDENTIFIERS:
1happy, return, int, "happy"

NOTE C/C++ IS CASE SENSITIVE SO happy AND Happy ARE DISTINCT IDENTIFIERS.

DECLARATION STATEMENTS SERVE THE SAME PURPOSE AS .DATA COMMANDS IN ASSEMBLY LANGUAGE. THEY DIRECT THE COMPILER TO SET ASIDE SPACE IN MEMORY WHICH WILL BE REFERED TO BY A SYMBOLIC NAME.