Dep: Computer science

Subject: Compiler

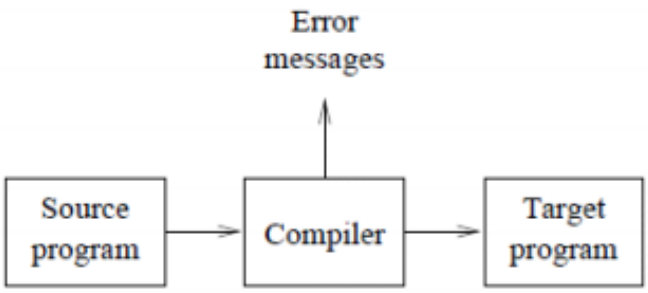
Third Class

Lecturer (1)

1. Introduction

1.1 Compilation

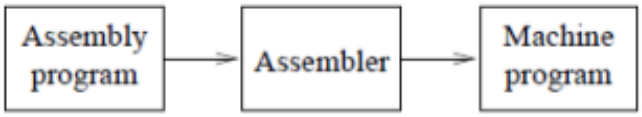
Definition. Compilation is a process that translates a program in one language (the source language) into an equivalent program in another language (the object or target language).



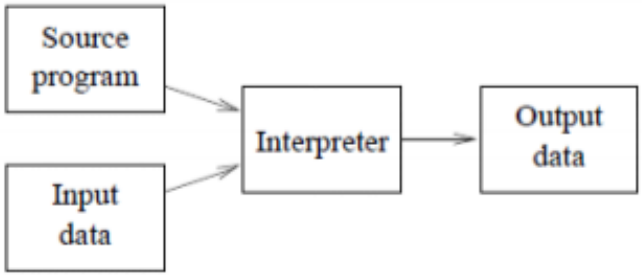
An important part of any compiler is the detection and reporting of errors; this will be discussed in more detail later in the introduction. Commonly, the source language is a high-level programming language (i.e. a problem-oriented language), and the target

language is a machine language or assembly language (i.e. a machine-oriented language). Thus compilation is a fundamental concept in the production of software: it is the link between the (abstract) world of application development and the low-level world of application execution on machines.

Types of translators. An assembler is also a type of translator:



An interpreter is closely related to a compiler, but takes both source program and input data. The translation and execution phases of the source program are one and the same.



Although the above types of translator are the most well-known, we also need knowledge of compilation techniques to deal with the recognition and translation of many other types of languages including:

- Command-line interface languages;
- Typesetting / word processing languages (e.g.TEX);
- Natural languages;
- Hardware description languages;
- Page description languages (e.g. PostScript);
- Set-up or parameter files.

Early Development of Compilers.

1940's. Early stored-program computers were programmed in machine language. Later, assembly languages were developed where machine instructions and memory locations were givensymbolic forms.

1950's. Early high-level languages were developed, for example FORTRAN. Although more problem-oriented than assembly languages, the first versions of FORTRAN still had many machine-dependent features. Techniques and processes involved in compilation were not well-understood at this time, and compiler-writing was a huge task: e.g. the first FORTRAN compiler took 18 man years of effort to write. Chomsky's study of the structure of natural languages led to a classification of languages according to the complexity of their grammars. The context-free languages proved to be useful in describing the syntax of programming languages.

1960's onwards. The study of the parsing problem for context-free languages during the 1960's and 1970's has led to efficient algorithms for the recognition of context-free languages. These algorithms, and associated software tools, are central to compiler construction today. Similarly, the theory of finite state machines and regular expressions (which correspond to Chomsky's regular languages) have proven useful for describing the lexical structure of programming languages.

From Algol 60, high-level languages have become more problem-oriented and machine independent, with features much removed from the machine languages into which they are compiled.

The theory and tools available today make compiler construction a managable task, even for complex languages. For example, your compiler assignment will take only a few weeks (hopefully) and will only be about 1000 lines of code (although, admittedly, the source language is small).