# Compiling

```
Module Code: ELEE1119

Module Name: Advanced Computer Engineering

Credits: 30

Module Leader: Seb Blair BEng(H) PGCAP MIET MIHEEM FHEA
```

Download as a PDF

## What we will cover

1. We will understand how ‘high’ and ‘low’ level programming languages are compiled to machine code so that it controls the hardware.
2. We will compare a number of programming languages and how they compile to machine code.

## What is programming?

- ?
- Why program?

# Types of Programming Languages

- High Level / low level – C#, Java, Python, Ruby, C, C++, assembly
- Declarative / imperative/procedural – SQL, Curl, Prolog
- General-purpose/domain specific – HTML, Markdown/up, MATLAB
- Object-orientated/concurrent – C#, Java, Python,
- Command/complied/script language – batch, bash, Javascript
- Answer set - Prolog

## Human Language and Programming Languages

- Are all programming languages in English?
- Does it matter when these are compiled down to machine code?

# Some Examples of Non-English Programming Languages

### Linotte

It has been a developer for using French keywords, and its “Hello world” program looks like this:

```
BonjourLeMonde:
  début
    affiche "Bonjour le monde!"
```

Has a web engine for HTML and PHP and JSP.

**SAKO**

System Automatycznego Kodowania Operacji (Automatic Operation Encoding System) programming language, which uses polish as for its keywords:

```
K) PROGRAM DRUKUJE NAPIS HELLO WORLD

   LINIA

   TEKST:

   HELLO WORLD

   KONIEC
```

Really only used in the late 1950s and early 1960s for the XYZ computers.

## Rapira

Rapira is another awesome example of non-english programming languages. It uses Russian keywords:

```
ПРОЦ СТАРТ()

    ВЫВОД: 'Привет, мир!'

КОН ПРОЦ
```

## Translated:

```
proc start()
     output: 'Hello, world!!!';
end proc
```

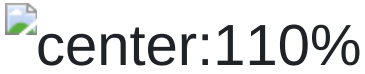
## EPL

Chinese engineers developed 易语言 (Easy Programming Language, as known as EPL):

```
公开 类 启动类
{
  公开 静态 启动()
  {
    控制台.输出("你好, 世界！");
  }
}

public class startup class
{
  public static start()
  {
    console.output("Hello, World!");
  }
}
```

# Compiling Code



# Lexical Analysis

The compiler begins converting the series of characters into tokens

High Level Code

```
int n = 11;
float q = 1.618;
if (n < 12)
{
  return q;
}
else
{
  return n;
}
```

| Token name | Example token values |
|---|---|
| identifier | n, q |
| keyword | int, float, if, else, return, while |
| separator | { }, ( ), [ ], ; |
| operator | +,- *, / , = ,< , >, : , ? |
| literal | True, false, 6.02e23, “string” |
| comment | // this is a comment */this is another comment/* |

# Syntax Analysis

Syntax analysis is based on the rules based on the specific programming language by constructing the parse tree with the help of tokens.

- Interior node: record with an operator filed and two files for children
  - Leaf: records with 2/more fields; one for token and other information about the token
  - Ensure that the components of the program fit together meaningfully
  - Gathers type information and checks for type compatibility
  - Checks operands are permitted by the source language

## Semantic Analyser

Semantic Analyser will check for Type mismatches, incompatible operands, a function called with improper arguments, an undeclared variable, etc.

```
int n = 11;
float q = 1.618*n;
```

In the above code, the semantic analyser will typecast the `int n 11` to `float 11.0` before multiplication.

# Intermediate Code Generation

Removes unnecessary code lines.

Arranges the sequence of statements to speed up the execution of the program without wasting resources.

Consider the following code, how can we remove unnecessary code?

```
a = int_to_float(10)
b = c * a
d = e + b
f = d
```

▶ Can become

# Code Generation

The objective of this phase is to allocate **memory locations**, **storage** and **generate relocatable machine code** or **machine instructions**.

The code generated by this phase is executed to take inputs and generate expected outputs, therefore, checks for unreachable statements.
Consider the following code, what error would be generated at this stage?

```
while (p == 10)
{
  break;
  int q = (0.5*8)*p;
}
```

# Code Generation

Now we are going to see how we go from C to Assembly to machine code…

```c
int square(int num) {
    return num * num;
}
```

```asm
square:
  pushq  %rbp
  movq   %rsp, %rbp
  movl   %edi, -4(%rbp)
  movl   -4(%rbp), %eax
  imull  %eax, %eax
  popq   %rbp
  ret
```

| | Memory Addresses |
|---|---|
| rbp[3] | 0x0007556ff0e0 |
| rbp[2] | 0x0007556ff0df |
| rbp[1] | 0x0007556ff0de |
| rbp[0] | 0x0007556ff0dd |
| | 0x0007556ff0dc |
| | 0x0007556ff0db |
| | 0x0007556ff0da |
| num | 0x0007556ff0d9 |

```c
int square(int num) {
    return num * num;
}
```

```
square:
  pushq  %rbp
  movq   %rsp, %rbp
  movl   %edi, -4(%rbp)
  movl   -4(%rbp), %eax
  imull  %eax, %eax
  popq   %rbp
  ret
```

```
HEX
55              01010101
48 89 e5        01001000 10001001 11100101
89 7d fc        10001001 01111101 11111100
0f af c0        00001111 10011111 11000000
54              01010100
c3              11000011
```

# Symbol Management Table

A symbol table contains a record for each identifier with fields for the attributes of the identifier.

| Operation | Function |
|---|---|
| allocate | to allocate a new empty symbol table |
| free | to remove all entries and free storage of symbol table |
| lookup | to search for a name and return a pointer to its entry |
| insert | to insert a name in a symbol table and return a pointer to its entry |
| set_attribute | to associate an atrribute to a given entry |
| get_attribute | to get an attribute associated with a given entry |

# Error Handling Routine

During compilation process error(s) may occur in all the below-given phases:

- Lexical analyser: Wrongly spelled tokens
- Syntax analyser: Missing parenthesis
- Semantic analyser: Mismatched data types, missing arguments
- Intermediate code generator: Mismatched operands for an operator
- Code Optimizer: When the statement is not reachable
- Code Generator: Unreachable statements
- Symbol tables: Error of multiple declared identifiers

## Labs

Begin the lab from blackboard, where you are going experience programming in several languages <C , Python and Ada> to do similar operations, and see how the code compiles and the subsequent outputs!