Software Development Tools

The UNIX Userland is an IDE – essential tools that follow the paradigm of “Do one thing, and do it right” can be combined.

The most important tools are:

- `$EDITOR` – essential for editing code
- the compiler toolchain
- `gdb(1)` – debugging your code
- `make(1)` – project build management, maintain program dependencies
- `diff(1)` and `patch(1)` – report and apply differences between files
- `cvs(1)`, `svn(1)`, `git(1)` etc. – revision control, distributed project management
```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello, World!\n");
}
```
Compilers

A compiler translates *source code* from a high-level programming language into *machine code* for a given architecture by performing a number of steps:

• preprocessing

```c
#include <stdio.h>
#define NUM 42

int
main(int argc, char **argv) {
  printf("%d\n", NUM);
}

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

#include <stdio.h>  // Repeating inclusion

int
main(int argc, char **argv) {
  printf("%d\n", 42);
}
```

```c
... void perror(const char *);
int printf(const char * __restrict, ...)
  __printflike(1, 2);
int putc(int, FILE *);
int putchar(int);
...

int
main(int argc, char **argv) {
  printf("%d\n", NUM);
}
```
A compiler translates source code from a high-level programming language into machine code for a given architecture by performing a number of steps:

• preprocessing
• lexical analysis

```c
if (num > 42) msg = "Don't panic!";
```

Compilers

A compiler translates source code from a high-level programming language into machine code for a given architecture by performing a number of steps:

- preprocessing
- lexical analysis
- syntax analysis

```c
if (num > 42) msg = "Don’t panic!";
```

```
if
  greater than
    num
    42
  assign
    msg
    "Don’t panic!"
```
Compilers

A compiler translates source code from a high-level programming language into machine code for a given architecture by performing a number of steps:

• preprocessing
• lexical analysis
• syntax analysis
• semantic analysis

```c
if (num > 42) msg = “Don’t panic!”;
```

```
if
greater than
num
42
assign
msg
“Don’t panic!”
```
A compiler translates source code from a high-level programming language into machine code for a given architecture by performing a number of steps:

- preprocessing
- lexical analysis
- syntax analysis
- semantic analysis

```c
if (msg > 42) num = "Don’t panic!";
```

"Ok, this part makes sense."

"Wait a second, you can’t compare a string to a number!"

"Assigning a string to a number makes no sense!"

"Don’t panic!"
Compilers

A compiler translates *source code* from a high-level programming language into *machine code* for a given architecture by performing a number of steps:

- preprocessing
- lexical analysis
- syntax analysis
- semantic analysis
- code generation
- code optimization

```
int i;

void func() {
    int j;
    j = 42;
    i = 5;
    i = 10;
    return;
    i = 20;
}
```

```
int i;

void func() {
    i = 10;
    return;
}
```

unused
-----------------------------
redundant
-----------------------------
dead code
A compiler translates source code from a high-level programming language into machine code for a given architecture by performing a number of steps:

- preprocessing
- lexical analysis
- syntax analysis
- semantic analysis
- code generation
- code optimization

```c
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello, World!\n");
}
```
A compiler translates *source code* from a high-level programming language into *machine code* for a given architecture.

- preprocessing
- lexical analysis
- syntax analysis
- semantic analysis
- code generation
- code optimization
- assembly

Jan Schaumann
Compilers

A compiler translates source code from a high-level programming language into machine code for a given architecture by performing a number of steps:

- preprocessing
- lexical analysis
- syntax analysis
- semantic analysis
- code generation
- code optimization
- assembly
- linking
A compiler translates *source code* from a high-level programming language into *machine code* for a given architecture by performing a number of steps:

- preprocessing
- lexical analysis
- syntax analysis
- semantic analysis
- code generation
- code optimization
- assembly
- linking

\[
\begin{align*}
\text{programming language specific} \\
\text{compiler specific} \\
\text{platform specific}
\end{align*}
\]
Compilers

A compiler translates source code from a high-level programming language into machine code for a given architecture by performing a number of steps:
Compilers

There are many different closed- and open-source compiler chains:

• Intel C/C++ Compiler (or *icc*)
• Turbo C / Turbo C++ / C++ Builder (Borland)
• Microsoft Visual C++
• Clang (a frontend to *LLVM*)
• GNU Compiler Collection (or *gcc*)
• Portable C Compiler (or *pcc*)
• …
The GNU Compiler Collection

GCC is the default on many Unix platforms, although some have recently switched to clang(1) / LLVM.

The compiler chain or driver usually performs preprocessing (e.g. via cpp(1)), compilation (cc(1)), assembly (as(1)) and linking (ld(1)).

To be continued...