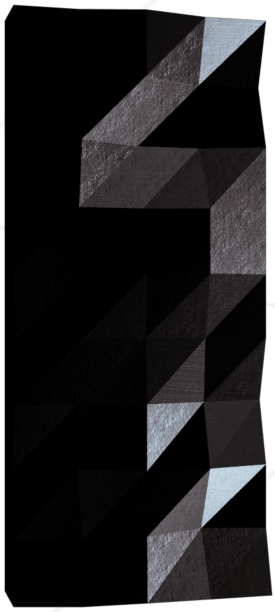
β

This project is currently under beta-test, until it is validated by a fair number of students in the 42 community. Please report any typo, incoherence, inconsistency, error, using the form https://tally.so/r/3lVDJo



B

The letter before C

Summary: Implementation of a B compiler using lex and yacc

Version: 1.0

Contents

Chapter I

Forewords

The Babel fish is small, yellow and leech-like, and probably the oddest thing in the Universe. It feeds on brainwave energy received not from its own carrier but from those around it. It absorbs all unconscious mental frequencies from this brainwave energy to nourish itself with. It then excretes into the mind of its carrier a telepathic matrix formed by combining the conscious thought frequencies with the nerve signals picked up from the speech centres of the brain which has supplied them. The practical upshot of all this is that if you stick a Babel fish in your ear you can instantly understand anything said to you in any form of language. The speech patterns you actually hear decode the brainwave matrix which has been fed into your mind by your Babel fish.
Now it is such a bizarrely improbable coincidence that anything so mindbogglingly useful could have evolved purely by chance that some thinkers have chosen it to see it as a final and clinching proof of the non-existence of God.
The argument goes something like this: "I refuse to prove that I exist," says God, "for proof denies faith, and without faith I am nothing."
"But," says Man, "the Babel fish is a dead giveaway isn't it? It could not have evolved by chance. It proves you exist, and therefore, by your own arguments, you don't. QED."
"Oh dear," says God, "I hadn't thought of that," and promptly vanishes in a puff of logic.
"Oh, that was easy," says Man, and for an encore goes on to prove that black is white and gets killed on the next zebra crossing.

Douglas Adams *H2G2*

Chapter II

The Only Things Necessary to Know

- A compiler is a program that **translates** code from a high-level language called the **source language** to a lower-level language called the **target language.**

Chapter III

Objectives

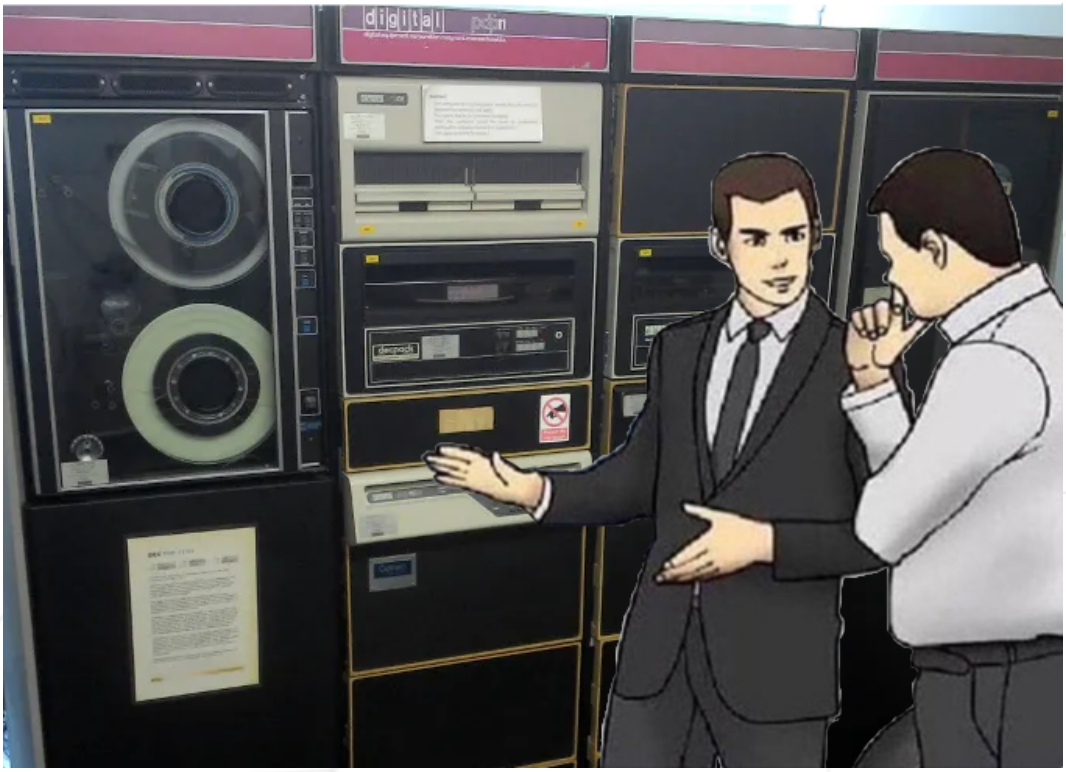
In this project, you will implement a B compiler using syntax directed translation. Syntax-directed translation is a compilation technique in which the parser completely drives the translation process, each non-terminal symbol is processed immediately after being identified. This technique, especially used with S-attributed grammars may allow the creation of AST-free compilers where the program is not stored and the only space used by the compiler is for the environment (eg: symbol table) and the parser stack (in shift-reduce parsers with only left-recursive rules that represent the maximum "depth" of the program). With this kind of compiler, the output code is produced on the fly, allowing the translation of programs of arbitrary length.

Chapter IV

Description

B is a programming language based on BPCL (so there is no language A). It was written by Ken Thompson (nobody, just the guy who made Unix). It looks a lot like C except that it has no types. B was originally created for the pdp-11 """minicomputer""".

Car Salesman (slapping roof of pdp-11):
This bad boy can fit so much f***ing bits in it



IV.1 Example

```
$> cat main.b
main()
{
  extrn putchar, char;
  auto i, s;
  i = 0;
  s = "hello, world\n";
  while (char(s, i))
  {
     putchar(char(s, i));
     i++;
  }
}
$> ./B <main.b
.intel_syntax noprefix
.text
.text
.globl main
main:
.long "main" + 4
enter 0, 0
push 0
push 0
lea eax, [ebp - 4]
push eax
mov eax, 0
pop ebx
mov [ebx], eax
lea eax, [ebp - 8]
push eax
.section .rodata
.LC0:
.long .LC0+4
.string "hello, world\n"
.text
mov eax, .LC0
pop ebx
mov [ebx], eax
.L1:
lea eax, "char"
mov eax, [eax]
push eax
lea eax, [ebp - 8]
mov eax, [eax]
push eax
lea eax, [ebp - 4]
mov eax, [eax]
push eax
mov ebx, [esp+0]
mov ecx, [esp+8]
mov [esp+8], ebx
mov [esp+0], ecx
pop eax
call eax
add esp, 8
cmp eax, 0
je .L2
lea eax, "putchar"
mov eax, [eax]
push eax
lea eax, "char"
mov eax, [eax]
push eax
lea eax, [ebp - 8]
mov eax, [eax]
push eax
lea eax, [ebp - 4]
mov eax, [eax]
push eax
mov ebx, [esp+0]
```

```
mov ecx, [esp+8]
mov [esp+8], ebx
mov [esp+0], ecx
pop eax
call eax
add esp, 8
push eax
mov ebx, [esp+0]
mov ecx, [esp+4]
mov [esp+4], ebx
mov [esp+0], ecx
pop eax
call eax
add esp, 4
lea eax, [ebp - 4]
mov ebx, [eax]
mov ecx, ebx
add ebx, 1
mov [eax], ebx
mov eax, ecx
jmp .L1
.L2:
leave
ret
```

Chapter V

General Instructions

This project will be corrected by humans only. You're allowed to organise and name your files as you see fit, but you must follow the following rules:

- Your program must be written in C using lex and yacc (or flex and bison...).
- You must submit a Makefile.
- You must not submit a generated C file, you must submit the .l and .y files and your makefile must call lex and yacc automatically.
- Your Makefile must compile the project and must contain the usual rules. It must recompile and re-link the program only if necessary.
- You have to handle errors carefully. In no way can your program quit in an unexpected manner (segmentation fault, bus error, double free, etc).
- You are allowed to use any function of the standard library or the glibc.

Chapter VI

Mandatory part

VI.1 The program

- You must write a B compiler.
- The program must be named B.
- The input language is B as defined by Thompson's technical memo (the attachment b.pdf) with the following additions/exceptions:
  - Internal declaration (reference to a variable not declared as external or automatic) are treated as label declarations.
  - Labels are in the same namespace as other symbols.
  - Since B doesn't have any type, there is no way to distinguish a function from an int, this is why all function call are made by pointer and the symbol of a function must designate a pointer to the function. For example, the function `f() return (42);` could be defined as follow:

```
f:
.long f + 4
enter 0, 0
mov eax, 42
leave
ret
```

    The same rule must be applied to labels for example: `lbl:` could be defined as follow:

```
.LREF12:
.section .rodata
.L12:
.long .LREF12
.text
```

    or

```
jmp [.L12]
.L12:
.long .L12 + 4
```

  - B was originally created for the PDP-11 computer which have word addressing while i386 uses byte addressing, to address this problem you must translate the expression a[b] as *(a + 4 * b) instead of *(a + b)

  - argc, argv and envp can be passed to the main in the same way as in C
  - You can do character/string literal parsing as in C (with \ as escape char instead of *)
  - You are not required to implement switch/case statements.

- The output language is i386 GNU assembly in Intel or AT&T syntax (I advise you to choose Intel)
- There is a file brt0.o in attachment, this file contains the entrypoint for a b program and a function `syscall` that work the same way has the c function `syscall(2)` that will allow you to use syscalls in your B programs (using the extrn keyword to declare it).
- During the evaluation, the following function will be used to compile B files:

```
compile()
(
  set -e
  for i in $(seq 1 $#)
  do
    eval S$i=$(mktemp)
    eval O$i=$(mktemp)
    ./B <"$(eval echo \$$i)" >$(eval echo \$S$i)
    gcc -c -m32 -x assembler "$(eval echo \$S$i)" -o $(eval echo \$O$i)
    rm "$(eval echo \$S$i)"
  done
  ld -m elf_i386 $(eval echo $(seq -f '$O%.0f' -s ' ' 1 $#)) brt0.o
  rm $(eval echo $(seq -f '$O%.0f' -s ' ' 1 $#))
)
```

  This function need to be executed in the folder containing B and brt0.o .

- You must use syntax directed translation so your program must not use any form of AST (think simple).
- Your program must read the input code from stdin, write the output code to stdout and errors to stderr.

This project is about golfing a compiler, don't worry about error diagnostics and optimisation of the output code.

Chapter VII

Bonus part

- switch/cases.
- floating point operators (like in the Thinkage version).
- the B standard library (in a separate folder, with a makefile).
- any other feature that you'd like to see in this language (up to two).

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter VIII

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double-check the names of your files to ensure they are correct.