Code Patching

Article Description	Change the title of the article to the one you are writing on.
Last Updated	@August 18, 2024 10:41 AM
Published	☑
Suggested	☑

```
assistflare-embed:

<iframe width="100%" height="300" src="//jsfiddle.net/the_archer
```

Imagine we are servicing equipment where computer code is running; perhaps in a kiosk, or on a micro-controller in an oil refinery, or in the firmware of some rover on Mars.

Maybe our aim is to:

- piggyback some additional data collection metrics code onto a variable or function, in order to log when and where it is used in the kiosk.
- Modify the fault tolerance parameters for a module to increase its service load in the refinery.
- update the protocol format of signals sent by the Mars rover.

Note that these objectives all share the commonality of running in an imbedded chip environment where we typically don't have access to mass storage, let alone an IDE (Integrated Development Environment), and hardware resources like RAM and CPU are limited.
We can't just shut off the system, tinker with it, and then restart it up again; so replacing the firmware with a newer version is simply not an option.

These constraints require us to interactively change the behavior of existing bits of compiled code, and do this while the entire system is running, without pausing its operation or shutting it down.

So we will need to do:

> *"Forward patching"* where some pre-existing routine is re-defined, and any code added later will use the newer definition.

and we will especially need the ability to do:

> *"Backward patching"* which is to swap out code of a function, and replace it with newer code, but where this exchange has an automatic ripple effect on ALL routines that are **already using** this function, so that the next time they execute they make use of the function's new code❣

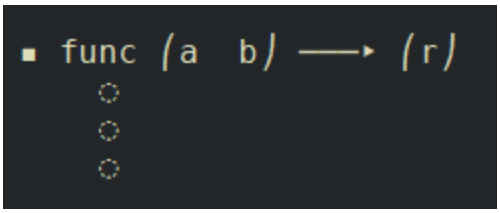
It is imperative that we be able to ***backward patch*** or ***forward patch*** ANY running code; we should be able to modify or redefine existing *words* (whether they be: functions, operators, variables, data structures, .. and also keywords for: conditionals, loops, directives, ...) so that the whole system automatically gets updated with our changes, while still running — without any recompilation.

Almost no computer language can do
forward patching of keywords, and none can do *backward patching* (although with some effort Forth can be modified to support this feature).

PAL natively supports both *FP* and *BP* out of the box.

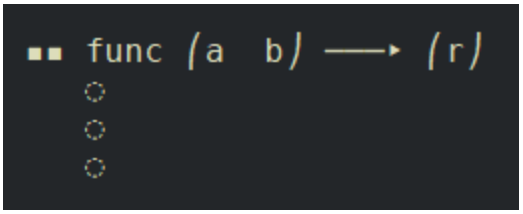
Patching in PAL

To make a
forward patch one simply defines the replacement word with the *mint* operator :



and if there was a pre-existing word with the identical signature, then a **redefinition warning** will be issued and the newly coded definition will supersede the older one when subsequently used by future code.

To do a
backward patch one re-defines a particular word by minting the intended replacement with the *double-mint* operator :



Now if there is no pre-existing word with the **exact same signature**, then an error message is issued, otherwise past and future code will use the newly created re-definition.

TL;DR

- Use of the *minting* operator means the word being defined (or re-defined) will be used from now onwards; forward in time.

- Use of the *double-mint* operator means the word being re-defined will be used forward in time, but ALSO backward in time (by previously defined routines) too.

Code Patching