plus_1 x = x + 1

```
def plus1 (x):
    return x + 1;
```

plus_2 :: (Num a) => a -> a
plus_2 x = plus_1 $ plus_1 x

```
def plus2 (x):
    return plus1 (plus1 (x));
```

fib_fact :: (Num a) => a -> a
fib_fact x = (fib x) + (fact x)

```
def fibFact (x):
    return fib (x) + fact (x);
```
Functions

```
Fac: return Positive is
  Fac := 1;
begin
  for i in 1 .. x loop
    Fac := Fac * i;
  end loop;
  return Fac;
end Fact;
```

```
unsigned int fact(int x) {  int fac = 1;
begin  for(i = 1, i <= x, i++) { fac = fac * i; 
return fac;
end  }
```

```
```

A compiler will likely use separate block for the stack? why?

What happens on exit parameters? When returning

what if this was holding some information during the function?

Universités: Université de Montréal, Université de Laval, Université de Sherbrooke, Université de Québec à Montréal

```
### Functions

**Components / phases of a function call**
- Parameters:传递给函数的参数。
- Local variables:函数内部的变量。
- Parameters to be initialized:初始化参数。

#### Facts
- Parameters to be initialized:初始化参数。
- Local variables:函数内部的变量。

#### Access
- Parameters:传递给函数的参数。
- Local variables:函数内部的变量。

#### Features
- Parameters:传递给函数的参数。
- Local variables:函数内部的变量。
- Parameters to be initialized:初始化参数。

### Functions

**ARM architecture calling practice**
- Call by:传递参数。
- Return by:返回返回值。
- Not expected to be used after return values.
- Not expected to be used after return values.

### Functions

**Python**
- All parameters passed as function arguments.

### Functions

**C**
- Parameters passed as function arguments.

### Functions

**Assembly**
- Parameters passed as function arguments.

### Functions

**Parameter passing**
- Call by name:传递参数。
- By value:传递参数。
- By reference (mutable):传递参数。

### Functions

**When to use?**
- Call by name:传递参数。
- By value:传递参数。
- By reference (mutable):传递参数。

### Functions

**Parameter unexpected**
- Call by name:传递参数。
- By value:传递参数。
- By reference (mutable):传递参数。

### Functions

**Parameter unexpected**
- Call by name:传递参数。
- By value:传递参数。
- By reference (mutable):传递参数。

### Functions

**Parameter unexpected**
- Call by name:传递参数。
- By value:传递参数。
- By reference (mutable):传递参数。

### Functions

**Python**
- All parameters passed as function arguments.

### Functions

**C**
- Parameters passed as function arguments.

### Functions

**Assembly**
- Parameters passed as function arguments.
Functions

Generic Stack-Frame

- Parameters (Global) variables
  - SB ( Savings Area )
  - Context (static) link
  - Parameters and return values can be passed on the stack.
  - SP ( Stack Pointer )
  - In case of a prior frame, the function can use the saved context.
- Local Variables
  - Temporary storage for temporary calculations.
- Function Value
  - A return value for the calling function.

One-way-by-reference

- Values are passed by reference.
- The function can read and write the parameter.
- Write access is prevented from inside the function.
- No read access outside the function.

Two-way-by-reference

- Values are passed by reference.
- The function can read and write the parameter.
- Write access is prevented from inside the function.
- No read access outside the function.

Two-way-by-copy

- Values are passed by value.
- The function can read the parameter.
- Write access is prevented from inside the function.
- No read access outside the function.

One-way-in-by-reference

- Values are passed by reference.
- The function can read the parameter.
- Write access is allowed.
- No read access outside the function.

Static vs. dynamic links

- Static link
  - The function is part of the program.
  - Calls are made directly.
  - No function name is needed.
- Dynamic link
  - The function is part of a library.
  - Calls are made by name.
  - Function name is needed.

Solved if it's the same

Parameters

- (Global) variables
- (Local) variables
- Temporarily saved variables
- Global, not visible to the function
- Local, visible to the function
- Temporarily saved, visible to the function

Context (static) link

- Function can read and write at any time.
- Outside code shall not write.

Dynamic vs. static links

- Static link
  - Fewer links needed.
  - No load time required.
- Dynamic link
  - More links needed.
  - Load time required.

Information in Information Flow

- Parameter passing mode
  - By value
  - By reference
  - By result
- Parameter passing mode dependencies
  - By value
  - By reference
  - By result
The Generic Stack-Frame

The last item to be stored before handing over control is the return address. The control flow returns to the address that was stored in the return address.

The next function call will produce the next stack frame.

The last function call will produce the next stack frame.

Note which variables and parameters are visible.

Variables and parameters from the context stay visible during function execution.

How fast / complex is the context switch?

Accessing the context like that can be inefficient!

At runtime, all context levels are available at once.

Which make all context levels accessible at once.

How to keep any memory allocation after function return?

... and a local variable in the calling function can keep this link.

Garbage collection (Java)? Smart pointers (C++)? Reference ownerships (Rust)?

Post_Call:

- Establish a new frame pointer
- Potentially initialize local variables
- Potentially restore the prior frame pointer

Function stack frames up to page 223 of 487 (chapter 3: “Functions” up to page 232)

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions

Functions
Summary

- Framework
  - Return address
  - Relative addressing
- Parameter passing modes and mechanisms
  - Copy versus reference
  - Information flow directions
  - Late evaluation
- Stackframes
  - Static and dynamic links
  - Parameters
  - Local variables