Carbon Language - http://github.com/carbon-language

Brainstorming optional syntax

Authors: TODO
Status: Draft
Created: 2025-MM-DD (@today)
Docs stored in Carbon's Shared Drive
Access instructions

Rust

```
// `Option(Ptr(i32))`
// `opt(ptr(i32))`
// `opt ptr i32`

// C++ budget:
// `const int* absl_nullable`
// `int const * absl_nullable`
//
// `opt(ptr(const i32))`

fn F(nullable: i32*?) -> i32? {
  return *?nullable;

  // or
  return *!nullable;

  // Zigish:
  return nullable.?*;

  // rust await-ish:
  return *nullable.try;
}

class C {
  var x: i32;
}

fn Call(pc: C**) -> i32? {
  return G1(pc as Option(C**).Some);
  return G1((pc as Option(C**).Some) as Option(C**?).Some);
}



class Expr {
}
enum class Operand {
  case OpExpr extend Expr*;
  case Constant extend Constant*;
}
```

```
enum class ASTNode {
  case class BinOp {
    var lhs: Operand;
    var rhs: Operand;
  }
  case ...;
}


fn G1(npc: C**?) -> i32? {
  if let (some ppc: auto = npc) {
    return **ppc;
  }

  let some ppc: auto = npc else {
    return null;
  }

  // Multi-level unwrap
  let nnpc: C**?? = npc;
  if let (some some ppc: C** = nnpc) {
  }



  // We prefer:
  let node: ASTNode.BinOp = {.lhs = x, .rhs = y};

  // To avoid:
  let node: auto = {.lhs = x, .rhs = y} as ASTNode.BinOp;

  // If `node` were just some unknown `ASTNode`...
  if let (as .BinOp some {.lhs = as .OpExpr, .rhs = some rhs: Expr*} = node) {
  }

  // Swiftier:
  if let ({.lhs = is .OpExpr, .rhs = (rhs: Expr*) as .OpExpr} as .BinOp = node)
{
  }

  // Swift with terrible hacks to add field names:
  if let .BinOp(.lhs @ is .OpExpr, .rhs @ .OpExpr(rhs: Expr*)) = node {
  }



  return npc?->>x;
  // or
  return npc!->>x;

  // Zig-ish:
  return npc.?**.x;

  return (*npc?)->x

  // Smith-ish (alternatively with leading `.`)
```

```
  return npc?**.x;

  // rust await-ish:
  return npc.try->>x;
  return (*npc.try)->x;
}

fn G2(pc: C?*) -> i32? {
  return pc->?.x;
  // or
  return pc->!.x;

  // Zig-ish:
  return pc.*?.x;

  return (*pc)?.x;

  // rust await-ish
  return pc->try.x;
  return (*pc).try.x;
}

// IMAGINE A DIFFERENT WORLD

// `Option(Ptr(i32))`
fn F(nullable: ?*i32) -> ?i32 {
  var n: i32;
  let p: *i32 = n&;

  return nullable?*;

  // or
  return nullable!*;
}

class C {
  var x: i32;
}

fn G1(npc: ?*C) -> ?i32 {
  return npc?*.x;
  // or
  return npc!*.x;
}

fn G2(pc: *?C) -> ?i32 {
  return pc*?.x;
  // or
  return pc*!.x;
}

// `Ptr(Option(i32))`
fn Q(non_null: i32?*);

// `Ptr(Const(i32))`
var x: const i32*;
```

```
// `Option(Ptr(Const(i32)))`
var x: const i32*?;

// `Ptr(Const(Option(i32)))`
var x: const (i32?)*;

// `Const(Ptr(Option(Const(i32))))`
var x: const ((const i32?)*);
```