H# Official Documentation

The official specification for H#

Contents

Abstract

This document is the official document for the H-Sharp (H#) programming language. The H stands for hybrid - as this is a multi-paradigm programming language heavily inspired by C# and Scala. This document will only contain the grammar of the language, the operational semantics as well as the type semantics of the language. The operational semantics may be explained with code samples - but obvious uses will not be explained.

This document will also contain the byte-instruction semantics. The official compiler, is written in C# for productivity purposes while the Virtual Machine is implemented using C++. At no point will this document be documenting the internal processes of those applications.

Grammar

The offical H# grammar. Note: $Id \in$ VarEnv and $TypeId \in$ TypeEnv. Both refer to the same grammatical definition, defined by the regular expression:

$$(\_|[\mathtt{a}-\mathtt{Z}])^{+}(\_|[\mathtt{0}-\mathtt{9}]|[\mathtt{a}-\mathtt{Z}])^{*}$$

If multiple elements can occur - and it'd be convenient, the element be sufficed with 'n' to show it may be contain n of such elements. $n \in \{0, 1, 2, \dots\}$. The notation $e_0, \dots, e_n$ represents the 1 to nth element with a specific separator. The full grammar is defined as follows:

CompileUnit ::= *CompileUnitElement*

CompileUnitElement ::= *CompileUnitElement CompileUnitElement*
| *Directive* | *Scope* | *ScopeElement* | *Declaration*

Scope ::= `{` *ScopeElement* `}`

ScopeElement ::= *ScopeElement ScopeElement*
| *Expr*`;` | *Statement* | *VarDeclaration*`;`

Expr ::= *Expr* | `(`*Expr*`)` | *Scope* | *LambdaExpr*
| *Expr BinaryOp Expr* | *UnaryOp Expr* | *Id UnaryOp*
| *Expr* `?` *Expr* `:` *Expr*
| *Id* | *Name* | `this` | `base`
| *Expr*`(`*Argument*`)`
| `new` *TypeId*`(`*Argument*`)` | `new` *TypeId*`[`*Argument*`]`
| *Expr*`.`*Id* | *Expr*`?.`*Id*
| *Expr*`.`*Id*`(`*Argument*`)` | *Expr*`?.`*Id*`(`*Argument*`)`
| *Expr* `is` *TypeID* | *Expr* `is null` *Expr* `is` *TypeID Id*
| *Expr* `is` *TypeID Id* `where` *Expr*
| *Expr* `is not` *TypeID* | *Expr* `is not null`
| *Expr* `is not` *TypeID Id* `where` *Expr*
| *Expr* `as` *TypeId*
| `(`*TypeID*`)` *Expr*
| `sizeof(`*TypeId*`)` | `addressof(`*Id*`)` | `typeof(`*Expr*`)`
| *Assignment*
| *Literal*

LambdaExpr ::= `(`*Param*`) =>` *Expr*

Directive ::= `type` *Id* `=` *TypeId*`;`
| `using` *Name*`;` | `using` *TypeId* `from` *Name*`;`
| *NamespaceDecl*

NamespaceDecl ::= `namespace` *Name Scope*

Statement ::= *Assignment*`;` | *ControlStatement* | *MatchStatement*`;`
| *TryCatchStatement*

ControlStatement ::= `if` *Expr Scope*
| `if` *Expr Scope* `else` *Scope*
| `if` *Expr Scope* `else if` *Expr Scope*
| `if` *Expr Scope* `else if` *Expr Scope* `else` *Scope*
| `while` *Expr Scope*
| `do` *Scope* `while` *Expr*`;`
| `for (`*Assignment*`;` *Expr*`;` *Expr*`)` *Scope*
| `for (`*VarDeclaration*`;` *Expr*`;` *Expr*`)` *Scope*
| `foreach (`*TypeId Id* in *Expr*`)` *Scope*
| `throw` *TypeID*`(`*Argument*`)`
| `return` *Expr*`;`
| `break;`

$$
\begin{array}{lcl}
\mathit{MatchStatement} & ::= & \mathit{Expr}\ \texttt{match}\ \{\ \mathit{MatchCase}\ \}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{MatchCase} & ::= & \mathit{MatchCase}\texttt{,}\ \mathit{MatchCase} \\
& | & \texttt{case}\ \mathit{Literal}\ \texttt{=>}\ \mathit{Expr} \mid \texttt{case}\ \mathit{TypeId}\ \texttt{=>}\ \mathit{Expr} \\
& | & \texttt{case}\ \mathit{TypeId}\ \mathit{Id}\ \texttt{=>}\ \mathit{Expr} \mid \texttt{case}\ \mathit{TypeId}\ \mathit{Id}\ \texttt{when}\ \mathit{Expr}\ \texttt{=>}\ \mathit{Expr} \\
& | & \texttt{case (}\mathit{MatchCaseId}\texttt{) =>}\ \mathit{Expr} \\
& | & \texttt{case (}\mathit{MatchCaseId}\texttt{) when}\ \mathit{Expr}\ \texttt{=>}\ \mathit{Expr} \\
& | & \texttt{case}\ \mathit{TypeId}\ \texttt{(}\mathit{MatchCaseId}\texttt{) =>}\ \mathit{Expr} \\
& | & \texttt{case}\ \mathit{TypeId}\ \texttt{(}\mathit{MatchCaseId}\texttt{) when}\ \mathit{Expr}\ \texttt{=>}\ \mathit{Expr} \\
& | & \texttt{case \_ =>}\ \mathit{Expr}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{MatchCaseId} & ::= & \mathit{MatchCaseId}\texttt{,}\ \mathit{MatchCaseId} \\
& | & \mathit{Id} \mid \texttt{\_}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{TryCatchStatement} & ::= & \texttt{try}\ \mathit{Scope}\ \texttt{catch (}\mathit{TypeId}\ \mathit{Id}\texttt{)}\ \mathit{Scope} \\
& | & \texttt{try}\ \mathit{Scope}\ \texttt{catch (}\mathit{TypeId}\ \mathit{Id}\texttt{) when}\ \mathit{Expr}\ \mathit{Scope} \\
& | & \texttt{try}\ \mathit{Scope}\ \texttt{catch (}\mathit{TypeId}\ \mathit{Id}\texttt{)}\ \mathit{Scope}\ \texttt{finally}\ \mathit{Scope} \\
& | & \texttt{try}\ \mathit{Scope}\ \texttt{catch (}\mathit{TypeId}\ \mathit{Id}\texttt{) when}\ \mathit{Expr}\ \mathit{Scope}\ \texttt{finally}\ \mathit{Scope}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{Assignment} & ::= & \mathit{Id}\ \texttt{=}\ \mathit{Expr} \\
& | & \mathit{Id}\ \texttt{+=}\ \mathit{Expr} \mid \mathit{Id}\ \texttt{-=}\ \mathit{Expr} \\
& | & \mathit{Id}\ \texttt{*=}\ \mathit{Expr} \mid \mathit{Id}\ \texttt{/=}\ \mathit{Expr} \\
& | & \mathit{Id}\ \texttt{\&=}\ \mathit{Expr} \mid \mathit{Id}\ |\texttt{=}\ \mathit{Expr} \\
& | & \mathit{Id}\ \texttt{\%=}\ \mathit{Expr}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{Declaration} & ::= & \mathit{VarDecl} \mid \mathit{FuncDecl} \\
& | & \mathit{ClassDecl} \mid \mathit{StaticClassDecl} \mid \mathit{InterfaceDecl} \\
& | & \mathit{UnionDecl} \mid \mathit{EnumDecl} \mid \mathit{StructDecl}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{ClassDecl} & ::= & \texttt{class}\ \mathit{Id}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{AccessMod}\ \texttt{class}\ \mathit{Id}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{StorageMod}\ \texttt{class}\ \mathit{Id}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{AccessMod}\ \mathit{StorageMod}\ \texttt{class}\ \mathit{Id}\ \{\ \mathit{ClassMember}\ \} \\
& | & \texttt{class}\ \mathit{Id}\texttt{(}\mathit{Param}\texttt{)}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{AccessMod}\ \texttt{class}\ \mathit{Id}\texttt{(}\mathit{Param}\texttt{)}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{StorageMod}\ \texttt{class}\ \mathit{Id}\texttt{(}\mathit{Param}\texttt{)}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{AccessMod}\ \mathit{StorageMod}\ \texttt{class}\ \mathit{Id}\texttt{(}\mathit{Param}\texttt{)}\ \{\ \mathit{ClassMember}\ \}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{StructDecl} & ::= & \texttt{struct}\ \mathit{Id}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{AccessMod}\ \texttt{struct}\ \mathit{Id}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{StorageMod}\ \texttt{struct}\ \mathit{Id}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{AccessMod}\ \mathit{StorageMod}\ \texttt{struct}\ \mathit{Id}\ \{\ \mathit{ClassMember}\ \} \\
& | & \texttt{struct}\ \mathit{Id}\texttt{(}\mathit{Param}\texttt{)}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{AccessMod}\ \texttt{struct}\ \mathit{Id}\texttt{(}\mathit{Param}\texttt{)}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{StorageMod}\ \texttt{struct}\ \mathit{Id}\texttt{(}\mathit{Param}\texttt{)}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{AccessMod}\ \mathit{StorageMod}\ \texttt{struct}\ \mathit{Id}\texttt{(}\mathit{Param}\texttt{)}\ \{\ \mathit{ClassMember}\ \}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{StaticClassDecl} & ::= & \texttt{object}\ \mathit{Id}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{AccessMod}\ \texttt{object}\ \mathit{Id}\ \{\ \mathit{ClassMember}\ \} \\
& | & \texttt{object}\ \mathit{Id}\texttt{(}\mathit{Param}\texttt{)}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{AccessMod}\ \texttt{object}\ \mathit{Id}\texttt{(}\mathit{Param}\texttt{)}\ \{\ \mathit{ClassMember}\ \}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{InterfaceDecl} & ::= & \texttt{interface}\ \mathit{Id}\ \{\ \mathit{ClassMember}\ \} \\
& | & \mathit{AccessMod}\ \texttt{interface}\ \mathit{Id}\ \{\ \mathit{ClassMember}\ \}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{ClassMember} & ::= & \mathit{ClassMember}\ \mathit{ClassMember} \\
& | & \mathit{Id}\texttt{(}\mathit{Param}\texttt{)}\ \mathit{FuncBody} \mid \mathit{AcessMod}\ \mathit{Id}\texttt{(}\mathit{Param}\texttt{)}\ \mathit{FuncBody} \\
& | & \mathit{VarDecl}\texttt{;} \mid \mathit{AcessMod}\ \mathit{VarDecl}\texttt{;} \\
& | & \mathit{FuncDecl} \mid \mathit{ClassDecl} \mid \mathit{UnionDecl} \mid \mathit{EnumDecl} \\
& | & \texttt{event}\ \mathit{TypeId}\ \mathit{id}\texttt{;} \mid \mathit{AccessMod}\ \texttt{event}\ \mathit{TypeId}\ \mathit{id}\texttt{;}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{VarDecl} & ::= & \mathit{TypeId}\ \mathit{Id}\ \texttt{=}\ \mathit{Expr} \mid \mathit{StorageMod}\ \mathit{TypeId}\ \mathit{Id}\ \texttt{=}\ \mathit{Expr} \\
 & \mid & \texttt{var}\ \mathit{Id}\ \texttt{=}\ \mathit{Expr} \mid \mathit{StorageMod}\ \texttt{var}\ \mathit{Id}\ \texttt{=}\ \mathit{Expr} \\
 & \mid & \mathit{TypeId}\ \mathit{Id} \mid \mathit{StorageMod}\ \mathit{TypeId}\ \mathit{Id} \\
 & \mid & \mathit{LambdaType}\ \mathit{Id}\ \texttt{=}\ \mathit{LambdaExpr} \\
 & \mid & \mathit{TupleDecl} \\
\\
\mathit{TupleDecl} & ::= & \texttt{(}\mathit{ParamType}\texttt{)}\ \mathit{Id}\ \texttt{=}\ \texttt{(}\mathit{Argument}\texttt{)} \\
 & \mid & \texttt{(}\mathit{Param}\texttt{)}\ \texttt{=}\ \texttt{(}\mathit{Argument}\texttt{)} \\
\\
\mathit{FuncDecl} & ::= & \mathit{Id}\texttt{(}\mathit{FuncParam}\texttt{):}\ \mathit{TypeId}\ \mathit{FuncBody} \\
 & \mid & \mathit{AccessMod}\ \mathit{Id}\texttt{(}\mathit{FuncParam}\texttt{):}\ \mathit{TypeId}\ \mathit{FuncBody} \\
 & \mid & \mathit{Id}\ \texttt{=}\ \texttt{(}\mathit{FuncParam}\texttt{):}\ \mathit{TypeId}\ \mathit{FuncBody} \\
 & \mid & \mathit{AccessMod}\ \mathit{Id}\ \texttt{=}\ \texttt{(}\mathit{FuncParam}\texttt{):}\ \mathit{TypeId}\ \mathit{FuncBody} \\
 & \mid & \mathit{AccessMod}\ \texttt{const}\ \mathit{Id}\ \texttt{=}\ \texttt{(}\mathit{FuncParam}\texttt{):}\ \mathit{TypeId}\ \mathit{FuncBody} \\
\\
\mathit{FuncBody} & ::= & \mathit{Scope} \mid \texttt{=>}\ \mathit{Expr} \\
\\
\mathit{FuncParam} & ::= & \mathit{FuncParam}\texttt{,}\ \mathit{FuncParam} \mid \mathit{Param} \\
 & \mid & \mathit{TypeId}\ \mathit{Id}\ \texttt{=}\ \mathit{Literal} \\
\\
\mathit{UnionDecl} & ::= & \texttt{union}\ \mathit{Id}\ \{\ \mathit{UnionMember}\ \} \\
 & \mid & \mathit{AccessMod}\ \texttt{union}\ \mathit{Id}\ \{\ \mathit{UnionMember}\ \} \\
 & \mid & \mathit{AccessMod}\ \texttt{static}\ \texttt{union}\ \mathit{Id}\ \{\ \mathit{UnionMember}\ \} \\
\\
\mathit{UnionMember} & ::= & \mathit{UnionMember}\ \mathit{UnionMember} \\
 & \mid & \mathit{TypeId}\ \mathit{Id}\texttt{;} \\
\\
\mathit{EnumDecl} & ::= & \texttt{enum}\ \mathit{Id}\ \{\ \mathit{EnumBodyMember}\ \} \\
 & \mid & \mathit{AccessMod}\ \texttt{enum}\ \mathit{Id}\ \{\ \mathit{EnumBodyMember}\ \} \\
 & \mid & \texttt{enum}\ \mathit{Id}\texttt{(}\mathit{EnumMember}\texttt{)}\{\ \mathit{EnumBodyMember}\ \} \\
 & \mid & \mathit{AccessMod}\ \texttt{enum}\ \mathit{Id}\ \texttt{(}\mathit{EnumMember}\texttt{)}\ \{\ \mathit{EnumBodyMember}\ \} \\
\\
\mathit{EnumBodyMember} & ::= & \mathit{EnumMember} \mid \mathit{FuncDecl} \mid \mathit{FuncDecl}\ \mathit{EnumBodyMember} \\
\\
\mathit{EnumMember} & ::= & \mathit{EnumMember}\texttt{,}\ \mathit{EnumMember} \\
 & \mid & \mathit{Id} \mid \mathit{Id}\ \texttt{=}\ \mathit{LiteralNoNull} \\
\\
\mathit{StorageMod} & ::= & \texttt{const} \mid \texttt{constexpr} \mid \texttt{static} \mid \texttt{abstract} \mid \texttt{override} \mid \texttt{virtual} \mid \texttt{final} \mid \texttt{lazy} \\
\\
\mathit{AccessMod} & ::= & \texttt{public} \mid \texttt{private} \mid \texttt{protected} \mid \texttt{internal} \mid \texttt{external} \\
\\
\mathit{Param} & ::= & \mathit{Param}\texttt{,}\ \mathit{Param} \\
 & \mid & \mathit{TypeId}\ \mathit{Id} \mid \texttt{const}\ \mathit{TypeId}\ \mathit{Id} \\
 & \mid & \mathit{ParamType}\ \mathit{Id} \\
\\
\mathit{ParamType} & ::= & \mathit{TypeId} \mid \mathit{TypeId}\texttt{,}\ \mathit{TypeId} \\
\\
\mathit{Argument} & ::= & \mathit{Expr} \mid \mathit{Expr}\texttt{,}\ \mathit{Expr} \\
\\
\mathit{Name} & ::= & \mathit{Id} \mid \mathit{Name}\texttt{.}\mathit{Name}
\end{array}
$$

$LambdaType \quad ::= \; \texttt{(}ParamType\texttt{):}\; TypeId$
$\quad\quad \mid \; TypeId \; \texttt{:} \; TypeId$

$BinaryOp \quad ::= \; \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/} \mid \texttt{\%} \mid < \mid > \mid <= \mid >= \mid \texttt{==} \mid \texttt{!=}$
$\quad\quad \mid \; || \mid \&\& \mid | \mid \& \mid << \mid >> \mid \texttt{=>} \mid \texttt{::} \mid \texttt{??} \mid \texttt{..}$

$UnaryOp \quad ::= \; \texttt{-} \mid \texttt{!} \mid \texttt{\#} \mid \texttt{++} \mid \texttt{--}$

$LiteralNoNull \quad ::= \; IntLit \mid FloatLit \mid DoubleLit \mid BoolLit \mid CharLit \mid StringLit$

$Literal \quad ::= \; LiteralNoNull \mid NullLit$

$Letter \quad ::= \; [\texttt{a-Z}]$

$Digit \quad ::= \; [\texttt{0-9}]$

$IntLit \quad ::= \; Digit^{+}$

$FloatLit \quad ::= \; Digit^{+}\texttt{.}Digit^{+}\texttt{f}$

$DoubleLit \quad ::= \; Digit^{+}\texttt{.}Digit^{+}$

$CharLit \quad ::= \; \texttt{'}Letter\texttt{'} \mid \texttt{'}\backslash Letter\texttt{'}$

$StringLit \quad ::= \; \ddot{}(Letter|Digit)^{*}\ddot{}$

$BoolLit \quad ::= \; \texttt{true} \mid \texttt{false}$

$NullLit \quad ::= \; \texttt{null}$

Operational Semantics

$$\textsc{VariableLookup}\quad \frac{\rho, \mu, \phi, \kappa, \sigma \vdash \rho(x) = v \neq (\ell, \omega, \sigma)}{\rho, \mu, \phi, \kappa, \sigma \vdash x \Rightarrow v, \sigma}$$

$$\textsc{HeapObjectLookup}\quad \frac{\rho, \mu, \phi, \kappa, \sigma \vdash \rho(x) = (\ell, \omega, \sigma) \qquad \sigma(\ell) = v \qquad \omega = \texttt{O}}{\rho, \mu, \phi, \kappa, \sigma \vdash x \Rightarrow v, \sigma}$$

$$\textsc{HeapStringLookup}\quad \frac{\rho, \mu, \phi, \kappa, \sigma \vdash \rho(x) = (\ell, \omega, \sigma) \qquad \sigma(\ell) = v \qquad \omega = \texttt{S}}{\rho, \mu, \phi, \kappa, \sigma \vdash x \Rightarrow v, \sigma}$$

$$\textsc{HeapArrayLookup}\quad \frac{\rho, \mu, \phi, \kappa, \sigma \vdash \rho(x) = (\ell, \omega, \sigma) \qquad \sigma(\ell) = v \qquad \omega = \texttt{A}}{\rho, \mu, \phi, \kappa, \sigma \vdash x \Rightarrow v, \sigma}$$

Type Semantics

$\theta = TypeEnv$ $\gamma = TypeLookupEnv$ $\eta = ReferenceTypeEnv$	$\theta, \gamma, \eta \vdash Expr : Type$ $\theta, \gamma, \eta \vdash Decl : \theta, \gamma$ $\theta, \gamma, \eta \vdash Decl : \theta, \gamma, \eta$	`int <: float <: double <: INumeric`

$$\textsc{T-IntLit}\quad \frac{i \in \mathbb{N}}{\theta, \gamma, \eta \vdash i : \texttt{int}}$$

$$\textsc{T-Addition}\quad \frac{\theta, \gamma, \eta \vdash e_1 : \tau_1 \qquad \theta, \gamma, \eta \vdash e_2 : \tau_2 \qquad \tau <: \texttt{INumeric}}{\theta, \gamma, \eta \vdash e_1 \texttt{ + } e_2 :}$$

$$\textsc{T-DeclVar-NonRef}\quad \frac{\theta, \gamma, \eta \vdash e : \tau \qquad \tau \notin \eta \qquad \theta' = \theta[x \mapsto \tau'] \qquad \tau' = \texttt{typeof}(t, \gamma) \qquad \tau <: \tau'}{\theta, \gamma, \eta \vdash t\ x \texttt{ = } e : \theta', \gamma, \eta}$$

$$\textsc{T-DeclVar-Ref}\quad \frac{\theta, \gamma, \eta \vdash e : \tau \qquad \tau \in \eta \qquad \theta' = \theta[x \mapsto \texttt{Ref}(\tau')] \qquad \tau' = \texttt{typeof}(t, \gamma) \qquad \tau <: \tau'}{\theta, \gamma, \eta \vdash t\ x \texttt{ = } e : \theta', \gamma, \eta}$$

$$\textsc{T-NewObject}\quad \frac{\theta, \gamma, \eta \vdash e_1, \ldots, e_n : \tau_1, \ldots, \tau_n \qquad \tau = \gamma(t)}{\theta, \gamma, \eta \vdash \texttt{new}\ t\texttt{(}e_1, \ldots, e_n\texttt{)} : \tau}$$

Compilation Semantics

Bytecode Semantics