

yaec

| |
|----------------------|
| COLLABORATORS |
|----------------------|

| | | | |
|---------------|------------------------|-----------------|------------------|
| | <i>TITLE :</i> yaec | | |
| <i>ACTION</i> | <i>NAME</i> | <i>DATE</i> | <i>SIGNATURE</i> |
| WRITTEN BY | | August 25, 2024 | |

| |
|-------------------------|
| REVISION HISTORY |
|-------------------------|

| NUMBER | DATE | DESCRIPTION | NAME |
|--------|------|-------------|------|
| | | | |

Contents

| | | |
|----------|--|----------|
| 1 | yaec | 1 |
| 1.1 | main | 1 |
| 1.2 | YAEC - Introduction | 1 |
| 1.3 | YAEC - Distribution, Registration | 2 |
| 1.4 | YAEC - Requirements, installation and usage | 3 |
| 1.5 | YAEC - History | 3 |
| 1.6 | YAEC - Changes/improvements that affects compability | 4 |
| 1.7 | YAEC - Extras | 7 |
| 1.8 | YAEC - Bugs/ToDo | 11 |
| 1.9 | YAEC - Thanks to | 11 |
| 1.10 | YAEC - The extended type-system | 12 |
| 1.11 | YAEC - Shared Libraries | 17 |

Chapter 1

yaec

1.1 main

```
*-----*
|                                     |
|               YAEK 1.4a           |
|       Yet Another E Compiler      |
|   By Leif Salomonsson 2000-2001  |
|                                     |
*-----*
```

Introduction

Usage, installation and requirements

Distribution, registration

History

Bugs/todo

Differences that affects compability

Other changes/additions

Thanks to

1.2 YAEK - Introduction

YAEK tries to be very compatible with
the original AmigaE Compiler 3.3a By Wouter.

Some things had to break and some things
have been made better/extended.

This guide tries to cover things that will work differently compared to Wouters ec3.3a.

Included by kind permission from Wouter is the original E.guide which can be found in the docs/ drawer.

YAEC was programmed from scratch and compiled with Wouters AmigaE 3.3a. The goal was to make an Ecompiler that was a bit more modern then the old one that was programmed in pure asm. Some differences :

- o All Modules are in ASCII format.
- o Interfacing to external functions such as shared or linked libs or <anything> is easily accomplished by writing an .ext file for it. Again, in ASCII.
- o Inline asm can target any 68k cpu from 020 and up. fpu-instructions included. This is thanx to Frank Willes PhxAss/PhxLnk!
- o Completer error-checking.
- o Better error-reporting.
- o Slower at compiling. (oh yeah ? :))
- o + tons of improved details here and there that just came along..

1.3 YAEC - Distribution, Registration

This software is subject to the "Standard Amiga FD-Software Copyright Note". It is SHAREWARE as defined in paragraph 4s.

If you find this program useful, please consider registering. The fee is \$30 or 250 SKR. (cash)

For more information please read "AFD-COPYRIGHT" (Version 1.4 or higher).
In English or In Swedish

Be sure to include your name and email in the registration!

My adress:

Leif Salomonsson
Nyg. 75 B Lag 0005
80310 GAVLE
SWEDEN

This program is NOT CRIPPLED IN ANY WAY.
This is for YOU - the user.
I trust you to register if your heart tells you to.

1.4 YAEC - Requirements, installation and usage

Requirements :

Compiler:

Should work on v37+, 68000
YAEC needs Frank Willes PhxAss and PhxLnk
(Aminet:dev/asm/) to be somewhere in the command-path.
If its not, add it with the AmigaDOS PATH command.

Generated exes:

Minimum v40 (3.1) of the operatingsystem.
Minimum 68020 cpu and 881 fpu.

Installation:

Unpack yaec.lha to ram:
Copy the drawer yaec<version> anywhere.
Put "ASSIGN E: <yaec-drawer>" in user-startup.
Put "PATH E:bin ADD" in user-startup.

Usage :

Pretty much the same as ec3.3a, but not all
args are there.

1.5 YAEC - History

1.1.1 Beta: (Dec 2000)

First public release

1.2b: (Jan 1 2001)

Second public release

- o Fixed some archiveing errors.
- o Methods returned crap.

1.3b: (Jan 21 2001)

- o Added this documentation.
 - o SetStdRast() was missing.
 - o Enabled 68881/2 fpu-asm (Martin Kuchinka)
 - o Added Amigalib support. -> amigalib.ext
 - o Improved NEW, objectptr[] still missing.
-

- o Added END keyword.

1.4a: (Feb 22 2001)

Third public release.

- o Extended the type-system. (not fully documented yet)
- o NEW/END/++/-- fully implemented.
- o Added AFD-Copyright Notice.
- o YAEC is now SHAREWARE.
- o Fixed NILCHECK and added PTRCHECK cli-args.
- o Compiling LINKOBJ now automatically creates .ext file.
(this means its usable now :))
- o LIBRARY/DEVICE mode !
- o Fixed a nasty bug in externs that could cause crashes
in some executables.
- o Fixed arguments to procedures so they are evaluated
in the same way they are on ec3.3a, that is from
left to right! some programs should work beter now! :)
- o Fixed SELECT statement so it accepts "x OF y".
- o Amigalib support gone again.. :(

1.6 YAEC - Changes/improvements that affects compability

Generated binaries

~~~~~

Requires v40+ (3.1+) of the operatingsystem and minimum 020, 881  
hardware for operation.

Sharing globals between modules and main program

~~~~~

In YAEC you can not EXPORT globals from modules.
What you can do is IMPORT them. All shared globals
must be declared in the main-program with DEF.
To share them, just import them with IMPORT DEF
in the module(s).

Librarybases

~~~~~

As before, exec, dos, intuition and graphics.library -

bases are automatically defined. But any other libraries used needs their bases to be defined in the program.

#### Pointer-math

~~~~~

YAEK does not allow ptr-math by default.
To untype a variable and make it slip threw the type-system without complaints, enclose the variable in Any(). ex :

```
x := Any(ptr)+12.
```

Strings

~~~~~

Inserting an apostrofe in strings by using '' is not possible.  
Use \a or \' instead.

#### DEF statement

~~~~~

DEF statements can not continiue on several lines in YAEK. This may change. On the other hand, this looks alot prettier, especially in procedures.

ex:

```
DEF x, y, z
DEF win:PTR TO window, scr, gads[10]:ARRAY OF LONG
DEF bla
```

Inline Asm

~~~~~

Inline Asm works very different in YAEK.  
When using inline asm one must declare it as such.  
For single instructions, ASM <instruction> can be used.  
For multiline asm :

```
ASM..
    <instruction>
    ...
    ...
ENDASM
```

Labels must be placed directly to the left, instructions must be atleast one space to the right.

The instructions and adressingmodes are only limited by the assembler, which in this case is PhxAss.  
The target cpu is 020, 881.

The compiler does not check the asm-code in any way!  
PhxAss may give you some complaints if it spots errors in your code.



To pass values between Asm and E, YAEK can use the registers D0-D7/A0-A7 as variables.

```
ex :
    x := D1
    A0 := bla()
```

The registers D0-D2/A0-A1/A6 are scratch registers.  
Any other registers used in inline asm must be saved/restored.

List 'N'Quote-functions

~~~~~

Slight API-change here.
same function, and some more.

```
bool      := ForAll(list, qexp, qexp2=NIL)
bool, pos := Exists(list, qexp, qexp2=NIL)
listvar   := MapList(list, listvar, qexp)
listvarlen := SelectList(list, listvar, qexp)
```

Now. what happened to the {var} ?
Here it is : \x (and \y).

In the above functions \x gives the
listitem and \y gives the listposition.

qexp2 in ForAll() and Exists() will,
if <> NIL be evaluated IF the result
is TRUE. Then the result of qexp2 will
be returned instead.

```
ex : Exists(list, '\x=14, 'Printf('yes, at pos \d\n', \y))
```

Theese functions can be nested!

Unary operators

~~~~~

++/--

The ++/-- operators works a bit more intelligent  
in YAEK. In some cases though, this may break  
things. In ec3.3a theese operators always affects  
the base-variable. In YAEK ++/-- always affects  
the latest dereferenced 'variable'.

YAEK ex:

```
var++ -> affects var (as before)
var[]++ -> affects var (as before)
obj.member-- -> affects member !
self.member[]++ -> affects member !
obj.optr.member[]-- -> affects member !
```

`^` : This operator is not supported.

Lisp-cells

~~~~~

Lisp-cells are experimental at the moment.

SetChunkSize(size) is removed.

Instead, use InitCells(size). InitCells()

MUST be called before any lisp-cells are used !

"CELL" will be raised if out of cells.

1.7 YAEC - Extras

Unification

~~~~~

(experimental)

Extensions:

exp <=> [BLA, x, y,...]:LONG

-> exp does not have to be a list, but rather an array of LONG.

exp <=> [BLA, x, \*, y]

-> \* matches any value.

exp <=> ['blabla', 10, y]

-> using string constants is okay

Unification also checks exp to be <> NIL,

If NIL then expression returns FALSE right away.

FOR statement

~~~~~

Differences:

startval, endval and stepval may be any expression
returning an integer value (LONG/INT/CHAR/ANY)

If the author recalls correctly, stepval had to be
a constant before. not anymore.

examples:

FOR var := startval TO endval [STEP (-)stepval]

<code>

ENDFOR

or

```
FOR var := startval TO endval [STEP (-)stepval] DO <code>
```

LOOP statement

~~~~~

Besides the old ways LOOP may now take a parameter :

```
LOOP nroftimes
```

```
  <code>
```

```
ENDLOOP
```

or

```
LOOP nroftimes DO <code>
```

nroftimes can be any integer expression (LONG/INT/CHAR/ANY)

Math operators + - \* /

~~~~~

Finally theese are all fully 32-bit !

Mul() and Div() still exists for backwards compability.

New internal functions

~~~~~

o Stringfunctions:

```
CloneStr(str, len=ALL)
```

```
  equals : StrCopy(String(IF len > -1 THEN len ELSE StrLen(str)), str)
  returns copy of str as an estr.
```

```
StrFmt(estr, fstr, vlist)
```

```
  like StringF() but values are passed by reference.
```

```
  ex : StrFmt(estr, 'ive got \d cents in my \s\n', [10, 'pocket'])
```

```
  vlist can be anything, but should point to some longwords.
```

```
StrIns(estr, str, pos)
```

```
  Inserts str in estr at position pos.
```

```
  returns estr.
```

```
StrRem(estr, len, pos)
```

```
  Removes len # of characters at position pos in estr.
```

```
  returns estr.
```

o IO-functions:

OutFmt(fstr, vlist)

like Printf() but values are given by reference.  
see also : StrFmt()

o Listfunctions:

CloneList(list, len=ALL)

equals : ListCopy(List(IF len > -1 THEN len ELSE ListLen(list)), list)  
returns new list.

o Graphics functions:

Hbox(x1, y1, x2, y2, col=1)

draws a hollow box.

o Classes support functions:

ObjectName(classobj)

Gives the name of an classobject.  
This is really useful, example when searching for a particular  
class in a linked list of classes.

ObjectSize(classobj)

gives the size of an classobject.  
for ex it could be used for cloning an object.

Globals in modules

~~~~~

No limitations for the globals in modules!
You can now define STRINGS, objects, LISTS, ARRAYS...
Defaultvalues can also be used. Even objects with methods
(classes) can be defined, they are automatically END:ed
at the end of program! :)

ex:

```
DEF s[100]:STRING, x=9, bla:ARRAY OF LONG
```

Classes in procedures

~~~~~

A simple "DEF o:myclass" in a PROCedure allocates  
cleared memory for the class and initialises  
the classinfoPtr. The class is automatically END:ed  
when the procedure is finished.

Stringformatting

~~~~~

YAEC uses custom stringformatting-functions
(this affects StringF(), WriteF(), Printf(), StrFmt(), OutFmt()),
which makes stringformatting around 5 times faster !
Unfortunately, this makes executables that uses
any stringformatting about 1k bigger then usual. but hey... :)
You will notice big differences in programs using alot
stringformatting. This was quite a bottleneck before.

Using Externs

~~~~~

The keyword EXTERN is used to import external  
functions such as shared library functions, or  
functions in a linklibrary.

ex: EXTERN 'gadtools'

Lets you use any functions in gadtools library.  
This replaces the old MODULE 'alibrary' statement.

externs for all the v40 libraryfunctions and amigalib  
are included.

ex2 : EXTERN 'mylinklib'

Lets ya use all public functions in a linklibrary.

YAEC uses files that ends with .ext to interface  
with theese kinds of external functions. The files  
can be found in E:externs/.

#### Note:

The format of the ext-files is very  
machine/OS/compiler-version specific and  
may change anytime.

#### Note2 :

I didnt bother renaming some of the os-functions  
that begins with two or more uppercase letters.  
(ec3.3a does not allow this)  
They're quite few and I like the original names better.

#### Compiling Linkobjects

~~~~~

(semi-official feature)

Yep, in YAEC you can compile standard amiga
objectfiles (.o). You can compile several of them
and then join them together with the amigaDOS
command JOIN to create a linklibrary (.lib)

There are some thing that you can not do in linkobjects :
export macros, objects or constants.

YAEC automatically creates an .ext file for you.
This file needs to be placed in externs/ to be used.

1.8 YAEC - Bugs/ToDo

Bugs:

- o Unification with cell-lists may succed with exp
beeing a longer list then the emmdiate one.

ToDo:

- o Default arguments to functions.
- o PRIVATE/PUBLIC in objects does nothing.
- o EXIT keyword. (considering to allow : IF exp THEN EXIT)
- o JUMP, GJUMP, LAB, GLAB keywords.
- o Stringformatting : \z and (x,y) is not implemented
- o Make Cell-system find cells in global data and in cells!.
- o MODULEs does not use the automatic EXTERNs (exec,dos,intui,gfx)
workaround : EXTERN them yourself. Simplest would be to
type EXTERN 'edig' -> this externs exec,dos,intui,gfx
- o Methods can not be used before they are defined. (may fix)
- o OPT STACK
- o Improve preprocessor
- o FreeCells(), Cell()
- o private methods.

1.9 YAEC - Thanks to

Thanks to (in absolutely no particular order) :

Wouter.

All on the AmigaE-List.

Martin Kuchinka.

Frank Wille.

Jay Miner.

The Universe.

Ludde.

Xorb.

All the beers.

1.10 YAEC - The extended type-system

8.I The extended type-system

~~~~~

Links to the old (ec3.3a) type-system:

- 8.A. about the 'type' system
- 8.B. the basic type (LONG/PTR)
- 8.C. the simple type (CHAR/INT/LONG)
- 8.D. the array type (ARRAY)
- 8.E. the complex type (STRING/LIST)
- 8.F. the compound type (OBJECT)
- 8.G. initialisation
- 8.H. the essentials of the E typesystem

The extended type system.

~~~~~

What it can do for you is more readable code, much more secure type-checking by the compiler and support for big types such as DOUBLE.

This while still beeing almost 100% backwards compability. The is because the new type-system is not forced on you to use, rather it just adds to the old system when you want it to.

Types quick overview.

~~~~~

Basic types:

```
DEF x          -> typeless: (32-bit)
DEF x:ANY      -> typeless: (32-bit)
```

(theese two are the same and can be used as  
PTR TO CHAR for backwards compability)

Simple types:

```
DEF l:LONG          -> integer: (signed 32-bit)
```

```
i:INT    (only in OBJECTs)  -> integer: (signed 16-bit)

c:CHAR    (only in OBJECTs)  -> integer: (unsigned 8-bit)

DEF f:FLOAT                                -> float: (IEEE 32-bit)
```

#### Big types:

```
DEF d:DOUBLE                                -> double float: (IEEE 64-bit)
```

#### Ptr types: (32-bit)

```
DEF ap:PTR TO ANY
DEF lp:PTR TO LONG
DEF ip:PTR TO INT
DEF cp:PTR TO CHAR
DEF fp:PTR TO FLOAT
DEF dp:PTR TO DOUBLE
DEF op:PTR TO <objectname>
```

#### Array types:

```
DEF aoa[#]:ARRAY OF ANY    -> array of any
[,...]:ANY

DEF aol[#]:ARRAY OF LONG    -> array of long
[,...]:LONG

DEF aoi[#]:ARRAY OF INT     -> array of int
[,...]:INT

DEF aoc[#]:ARRAY            -> array of char
DEF aoc[#]:ARRAY OF CHAR    -> array of char
[,...]:CHAR

DEF aof[#]:ARRAY OF FLOAT   -> array of float
[,...]:FLOAT

DEF aod[#]:ARRAY OF DOUBLE  -> array of double
[,...]:DOUBLE

DEF aoo[#]:ARRAY OF <objectname> -> array of object
[,...]:<objectname>
```

#### Complex types:

```
DEF l[#]:LIST    (only as DEF) -> list of any
[,...]

DEF s[#]:STRING  (only as DEF) -> string of char
'bla'
```

---



Compound types:

```
DEF o:<objectname> -> object
[,...]:<objectname>
```

Indexing. ([])  
~~~~~

Types that can be indexed are : basic, array, complex and ptr-type.

The new expression changes its type. ARRAY OF xxx [] -> xxx,
PTR TO xxx [] -> xxx, LIST [] -> ANY, STRING [] -> CHAR.

The ANY type can be used as a PTR TO CHAR for backwards compability:
ANY [] -> CHAR.

Increment/decrement. (++/--)
~~~~~

Basic, simple, big and ptr-types can be incremented/decremented.

Math and logic. (+ - \* / AND OR ! #)  
~~~~~

Math and logic is type-sensitive.

Basic, simple and big types can be used in Math and logic expressions.

Assignment. (:=)
~~~~~

Assignment is type-sensitive.

ex :

```
DEF o:object, x:ANY, d:DOUBLE, li[10]:LIST, o2:PTR TO object,
    c:CHAR, l:LONG, s[120]:STRING
```

```
o := o2
-> copies SIZEOF object from o2 to o.
```

```
o := x
-> copies SIZEOF object from x to o
```

```
o := li
-> compiler error! types do not match.
```

```
o2 := o
-> o2 gets the ADDRESS of o (as o2 is a PTR)
```

```
d := x
-> copies 64bits of double float data from x to d
    (x should point to a double here)
```

---

```

x := d
-> x gets the ADDRESS of d (as it cant hold 64 bits)

c := l
-> nothing special here, c gets the value of l

l := li
-> type error! (l is supposed to hold an integer, and li is a list)

s := x
-> StrCopy(s, x)

li := x
-> ListCopy(li, x)

```

The first variable (leftmost) denotes how the cpying should take place. eg. a variable can never get more data than it can handle. (no mem overwrite).

allowed modes table:

```

ANY := ANY/LONG/INT/CHAR/FLOAT/DOUBLE/STRING/LIST/ARRAY/OBJECT/PTR

LONG := ANY/LONG/INT/CHAR

FLOAT := ANY/FLOAT

DOUBLE := (ANY)/DOUBLE

INT := ANY/LONG/INT/CHAR

CHAR := ANY/LONG/INT/CHAR

OBJECT := (ANY)/PTR TO OBJECT/OBJECT

LIST := (ANY)/LIST

STRING := (ANY)/STRING

PTR TO ANY := ANY/LIST/ARRAY OF LONG/ARRAY OF ANY/ARRAY OF FLOAT/
PTR TO FLOAT/PTR TO LONG/PTR TO ANY/LISTPTR

PTR TO LONG := ANY/LIST/ARRAY OF LONG/ARRAY OF ANY/
PTR TO LONG/PTR TO ANY/LISTPTR

PTR TO FLOAT := ANY/LIST/ARRAY OF FLOAT/ARRAY OF ANY/
PTR TO FLOAT/PTR TO ANY

PTR TO INT := ANY/ARRAY OF INT/PTR TO INT

PTR TO CHAR := ANY/STRING/ARRAY OF CHAR/PTR TO CHAR

PTR TO DOUBLE := ANY/PTR TO DOUBLE/DOUBLE/ARRAY OF DOUBLE

PTR TO OBJECT := ANY/PTR TO OBJECT/OBJECT/ARRAY OF OBJECT

```

---

phew...

Comparison. (</>/<>/=/<=>=)  
 ~~~~~

Comparison is type-sensitive.

ex:

```
DEF x:LONG, y, z:ANY, bla:STRING, bluu:STRINGPTR, d:DOUBLE

x = y  -> 32bit integer comparison
y = x  -> 32bit integer comparison
z <= d  -> 64 bit double float comparison
bluu <> bla -> type error ! no arrays/complex/objects allowed.
bluu = NIL -> 32bit ptr comparison
```

allowed modes:

```
ANY - PTR

PTR - ANY/LONG/PTR

LONG/INT/CHAR/ANY - LONG/INT/CHAR/ANY

FLOAT - FLOAT/ANY

DOUBLE - DOUBLE/ANY
```

The first type (leftmost) denotes how the comparison should be done.

When comparing array/complex or compound types with something,
 only, = and <> can be used.
 (you can only check if they are equal or not equal)

another angle; NOT allowed modes: (not exactly all of them)

```
ANY/LONG/INT/CHAR/FLOAT/DOUBLE - OBJECT/LIST/STRING/ARRAY

LIST - STRING

STRING - LIST
```

Think about why theese modes would be stupid.
 ...Agree ? :)

1.11 YAEC - Shared Libraries

In the works..