PBCExtension

P. Baillehache

January 26, 2019

Contents

Introduction

PBCExtension is a C library providing macros to extend the C language.

It uses no external library.

1 Interface

```
// ============ PBCEXTENSION.H ================

#ifndef PBCEXTENSION_H
#define PBCEXTENSION_H

// ================= Include =================

#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
```

```
// ================= Define ==================

// Return the number of arguments of a variadic function given
// the type 'Type' of these arguments
#define __VA_NB_ARGS__(Type, ...) (sizeof((Type[]){__VA_ARGS__})/sizeof(Type))

#endif
```

2 Code

2.1 pbcextension.c

```
// ============ PBCEXTENSION.C ================

// ================= Include =================

#include "pbcextension.h"

// ================= Define ==================
```

3 Makefile

```
# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=0

all: pbmake_wget main

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)

# Rules to make the executable
repo=pbcextension
$($(repo)_EXENAME): \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) `echo "$($(repo)_EXE_DEP) $($(repo)_EXENAME).o" | tr ' ' '\n' | sort -u` $(LINK_ARG) $($(repo)_LINK_ARG)

$($(repo)_EXENAME).o: \
$($(repo)_DIR)/$($(repo)_EXENAME).c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) `echo "$($(repo)_INC_DIR)" | tr ' ' '\n' | sort -u` -c $($(repo)_DIR)/$
```

4 Unit tests

```
#include <stdlib.h>
#include <stdio.h>
#include "pbcextension.h"

int _TestVANbArgsInt(int nbArg, ...) {
  return nbArg;
}
#define TestVANbArgsInt(...) \
  (_TestVANbArgsInt(__VA_NB_ARGS__(int, __VA_ARGS__), __VA_ARGS__))

int _TestVANbArgsStr(int nbArg, ...) {
  return nbArg;
}
#define TestVANbArgsStr(...) \
  (_TestVANbArgsStr(__VA_NB_ARGS__(char*, __VA_ARGS__), __VA_ARGS__))

int _TestVANbArgsFloat(int nbArg, ...) {
  return nbArg;
}
#define TestVANbArgsFloat(...) \
  (_TestVANbArgsFloat(__VA_NB_ARGS__(float, __VA_ARGS__), __VA_ARGS__))

void UnitTestVANbArgs() {
  if (TestVANbArgsInt(1) != 1) {
    printf("UnitTestVANbArgs OK\n");
    exit(1);
  }
  if (TestVANbArgsStr("a", "b") != 2) {
    printf("UnitTestVANbArgs OK\n");
    exit(1);
  }
  if (TestVANbArgsFloat(1.0, 2.0, 3.0) != 3) {
    printf("UnitTestVANbArgs OK\n");
    exit(1);
  }
  printf("UnitTestVANbArgs OK\n");
}

void UnitTestAll() {
  UnitTestVANbArgs();
}

int main(void) {
  UnitTestAll();
  return 0;
}
```

5 Unit tests output

```
UnitTestVANbArgs OK
```