

Re: Help is needed to compile C program using Visual Studie 2005

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2006-06/msg00241.html>

- *From:* wtxwtx@xxxxxxxxxx
 - *Date:* 7 Jun 2006 18:50:41 -0700
-

Hi,

I tried the following ways:

A.

1. Change my file name filter.c to filter.cpp;
 2. Compile it;
- Same error as before in the above 2 situations.

B.

Add stdafx.cpp that contains one statement:

The file is:

```
// stdafx.cpp : source file that includes just the standard includes
// Filter.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information
```

```
#include "stdafx.h"
```

```
// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```

Same errors as before.

C. Only using stdafx.cpp without any my files.

Error information:

Error 1 error C2859: c:\0-filter\newfilter\filter\debug\vc80.pdb is not the pdb file that was used when this precompiled header was created, recreate the precompiled

header. c:\0-filter\newfilter\filter\filter\stdafx.cpp

Error 2 error C2859: c:\0-filter\newfilter\filter\debug\vc80.idb is not the idb file that was used when this precompiled header was created, recreate the precompiled

header. c:\0-filter\newfilter\filter\filter\stdafx.cpp

D. Project contains 2 files:

filter.cpp

stdafx.cpp

Error information:

Error 1 error C2859: c:\0-filter\newfilter\filter\debug\vc80.pdb is not

the pdb file that was used when this precompiled header was created,
recreate the precompiled

header. c:\0-filter\newfilter\filter\filter\stdafx.cpp

Error 2 error C2859: c:\0-filter\newfilter\filter\debug\vc80.idb is not
the idb file that was used when this precompiled header was created,
recreate the precompiled

header. c:\0-filter\newfilter\filter\filter\stdafx.cpp

Error 3 error C2859: c:\0-filter\newfilter\filter\debug\vc80.pdb is not
the pdb file that was used when this precompiled header was created,
recreate the precompiled

header. c:\0-filter\newfilter\filter\filter.cpp 60

Error 4 error C2859: c:\0-filter\newfilter\filter\debug\vc80.idb is not
the idb file that was used when this precompiled header was created,
recreate the precompiled

header. c:\0-filter\newfilter\filter\filter.cpp 60

Error 5 fatal error C1083: Cannot open include file: 'iostream.h': No
such file or directory c:\0-filter\newfilter\filter\filter.cpp 65

The following is the text of filter.c:

```
// This program VHDL-Filter is to generate different versions of *.vhd
// In the input *.vhd, it handles 5 statements defined in C/C++
// #define
// #ifdef
// #ifndef
// #else
// #endif
//
// 1. After passing through the VHDL-Filter program, it will generate
an output file whose name has the following format:
// InputFileName-A-B.vhd,
// where A is the first defined variable, B second and so on.
// 2. If defined variable has a dash, only part after last dash is
appended to the output file name
//
// For example
// #define Chip_A
// #define debug
// input file name is 7460-X, then the output file name would be
7460-X-A-debug.vhd

// limitations:
// 1. '#' must be the first char of a line for above 5 key words to be
recognized;
// 2. Variable for #define, #ifdef and #ifndef must at the same row as
#define, #ifdef and #ifndef;
// 3. Variable for #define, #ifdef and #ifndef must be terminated by
space, tab or newline so that it can be recognized as variable;
// 4. The length of input file paths and name must be less than 256;
// 5. The length of each line of input file must be less than 256;
// 6. The number of defined variables must be less than 10;
//
```

```
// New features:
// 1. The program can generate 3 special versions:
// a. one is assumed to have Chip_A option built in the program
// b. one is assumed to have Chip_B option built in the program
// c. none is assumed to be built in the program
// So executing one of 3 different versions will generate 3
different VHDL versions for Chip_A or Chip_B, or no Chip_A/Chip_B
// without changing the original VHDL file.

// Command input format:
// VHDL_Filter
// program will ask to enter input file path and name after it starts

// How to create a new project with different file names:
// 1. Create new directory under C:
// 2. Create new project with Visual C++;
// 3. Select newly created directory as project directory
// 4. Select Win32 Console application without any new files;
// 5. Open 5425-PCI-X.c, save it as it has project name at the new
project directory
// 6. Click Build/Compile ..., get new file compiled;
// 7. Click Build/Set Active Configuration, and select Win32 release
// 8. Click Build/Build ..., a new project is build;
// 9. Move xxx.exe from c:\...\release directory to background of
monitor; The icon would take the C file name;

// used to generate 3 different versions for Chip_A, Chip_B or no
Chip_A/Chip_B
// If Chip_A version is to be generated, leave row '#define Chip_A'
uncommented and comment out row '#define Chip_B'

##define Chip_A
##define Chip_B
##define Chip_C
#define Chip_D

#include "stdafx.h"
#include <sys/stat.h> // for constant _S_IWRITE in chmod()
#include <stdio.h>
#include <io.h>
#include <string.h>
#include <iostream.h>
#include <direct.h>

int _tmain(int argc, _TCHAR* argv[]) // new interface
//int main(int argc) {
int IfdefLevel = 0; // number of levels of #ifdef &
#ifndef started from beginning
int NoCopyLevel = 0; // number of levels not to copy, = 0:
```

```
enable copy; > 0: not copy
char InputFileName[256] =
"c:\\0-pci-core-x\\5425-X\\5425_X_6C0.vhd";
// char InputFileName[256] =
"c:\\0-pci-core-x\\5425-PCI-X\\5425-PCI-X-SDRAM-700.vhd";
// char InputFileName[256] =
"c:\\0-pci-core-x\\7460-x\\7460_700.vhd";
// char InputFileName[256] =
"c:\\0-pci-core-x\\5425-PCI-X\\5425-PCI-X-SDRAM-700.vhd";
// char InputFileName[256] =
"c:\\0-pci-core-x\\5425-X\\5425_X_700.vhd";
char OutputFileName[256];
char InputLine[256];
char pTempName[] = "filter";
char Extension[20]; // input file extension with '.'
int IsExtension = 0;
char* pOutputFileNameEnd;
char* p1;
char* p2;
char* pVariable; // ptrs to defined variable after
#define, #ifdef, #ifndef
FILE* pFILEInput;
FILE* pFILEOutput;
char DefinedVariableArray[10][256];
int DefinedVariableArrayIndex = 0, i;
// Old: upon entering this program working dir
// New: dir where input file resides
char OldWorkingDir[256];
int IsNewWorkingDir = 0, Line = 0;
int DefineChipA = 0, DefineChipB = 0, DefineChipC = 0, DefineChipD
= 0; // indicator for define statements: #define Chip_A/Chip_B

#ifdef Chip_A
DefineChipA = 1;
#endif

#ifdef Chip_B
DefineChipB = 1;
#endif

#ifdef Chip_C
DefineChipC = 1;
#endif

#ifdef Chip_D
DefineChipD = 1;
#endif

// get old working dir
_getcwd(OldWorkingDir, sizeof OldWorkingDir);
```

```
// if there is no input file name, ask users to input
while(1) {
if(argc = 1)
break;
printf("\nPlease input your file path & name\n");
// InputFileName -->path/name
p1 = gets(InputFileName);
if(p1 == NULL)
continue;
else
break;
};
// change subdirectory where input file resides as current working
directory
// assume input file carries its dir path by searching its last '/'
or '\\'
p1 = strrchr(InputFileName, '/');
// if path name is found
if((p1 != NULL)) {
// separate path from its name
*p1 = '\0';
IsNewWorkingDir = 1;
// change directory where input file resides as current working
dir
chdir(InputFileName);
// output file name has the same name as input file name except
that
// it will be appended by names defined in #define of input file
strcpy(OutputFileName, ++p1);
// restore full input file name
*--p1 = '/';
} else {
// path marks '/' or '\' are exchangable in Window
p1 = strrchr(InputFileName, '\\');
if((p1 != NULL)) {
// separate path from its name
*p1 = '\0';
IsNewWorkingDir = 1;
// change directory input resides as current working dir
chdir(InputFileName);
// output file name has the same name as input file name
except that
// it will be appended by names defined in #define
strcpy(OutputFileName, ++p1);
// restore full input file name
*--p1 = '\\';
} else {
// no path is found, don't change dir
strcpy(OutputFileName, InputFileName);
}
}
```

```
};

// try to find its extension
p1 = strchr(OutputFileName, '.');
// if input file has extension, keep it with output file
if(p1 != NULL) {
    IsExtension = 1;
    strncpy(Extension, p1, sizeof Extension);
};

// OutputFileName contains input file name
pFILEInput = fopen(OutputFileName, "r");
if(pFILEInput == NULL) {
    printf("\nFail to open input file: %s\n", InputFileName);
    gets(InputFileName);
    return(1);
};

// first output file name is a temporary name, after new file name
// is established, change it back to new name
pFILEOutput = fopen(pTempName, "wt");
if(pFILEOutput == NULL) {
    perror(pTempName);
    printf("\nFail to open output file %s:\n", pTempName);
    // close opened input file
    fclose(pFILEInput);
    gets(InputFileName);
    return(2);
};

// pOutputFileNameEnd ---> char position to be overwritten by new
// appending string
pOutputFileNameEnd = strchr(OutputFileName, '.');
// if input file name doesn't contain extension
if(pOutputFileNameEnd == NULL)
// pOutputFileNameEnd ---> the last char
pOutputFileNameEnd = strchr(OutputFileName, '\0');
else
// pOutputFileNameEnd ---> end of OutputFileName
*pOutputFileNameEnd = '\0';

while(1) {
    if(DefineChipA) {
        strcpy(InputLine, "#define Chip-A");
        p1 = InputLine;
        DefineChipA = 0;

    } else if(DefineChipB) {
        strcpy(InputLine, "#define Chip-B");
        p1 = InputLine;
        DefineChipB = 0;
    }
}
```

```
} else if(DefineChipC) {
strcpy(InputLine, "#define Chip-C");
p1 = InputLine;
DefineChipC = 0;

} else if(DefineChipD) {
strcpy(InputLine, "#define Chip-D");
p1 = InputLine;
DefineChipD = 0;

} else {
Line++;
// get a line
p1 = fgets(InputLine, sizeof InputLine, pFILEInput);
};

// if file ends, quit the while loop
if(p1 == NULL)
break;
if(InputLine[0] != '#') {
// copy or not copy a line from input file to output file
based on current NoCopyLevel value
// it is determined by #ifdef, #ifndef, #endif appearances in
the file
if(NoCopyLevel == 0)
fputs(InputLine, pFILEOutput);
continue;

// line starts with '#'
} else {
// print out '#' lines
printf("\nAt line %d: %s \n", Line, InputLine);

// #define handling
if(InputLine[1] == 'd' && //#define found
InputLine[2] == 'e' &&
InputLine[3] == 'f' &&
InputLine[4] == 'i' &&
InputLine[5] == 'n' &&
InputLine[6] == 'e') {
// to get variable name defined by #define
pVariable = strtok(&InputLine[7], " \t\n");
// on return pVariable --> variable or NULL
if(pVariable == NULL) {
printf("\nFormat: #define variable \nor VARIABLE MUST BE
ON THE SAME LINE AS #define");
fclose(pFILEOutput);
fclose(pFILEInput);
gets(InputFileName);
return(3);
```

```

};
// if the variable has no '-', append full defined variable
to output file name
// if the variable has '-', append the substring after dash
to the output file name
*pOutputFileNameEnd++ = '-';
p2 = strchr(pVariable, '-');
// pOutputFileNameEnd --> first char after string, but not
a
if(p2 == NULL) {
//append full defined variable name
strcpy(pOutputFileNameEnd, pVariable);
pOutputFileNameEnd += strlen(pVariable);
} else {
//append substring to the output file name
strcpy(pOutputFileNameEnd, ++p2);
// let pOutputFileNameEnd ptr to '\0' after string
pOutputFileNameEnd += strlen(p2);
};
//put a new defined variable into defined variable array
if(DefinedVariableArrayIndex >= 9) {
printf("\nToo #define statements: > 10");
fclose(pFILEOutput);
fclose(pFILEInput);
gets(InputFileName);
return(4);
} else {

strcpy(DefinedVariableArray[DefinedVariableArrayIndex++], pVariable);
continue;
};

// #ifdef handling
} else if( InputLine[1] == 'i' &&
InputLine[2] == 'f' &&
InputLine[3] == 'd' &&
InputLine[4] == 'e' &&
InputLine[5] == 'f') { //ifdef found
// enter next level of #ifdef & #ifndef
IfdefLevel++;
// to get variable name defined by #define
pVariable = strtok(&InputLine[6], "\t\n");
if(pVariable == NULL) {
printf("\nFormat: #ifdef variable \nor VARIABLE MUST BE
ON THE SAME LINE AS #ifdef");
fclose(pFILEOutput);
fclose(pFILEInput);
gets(InputFileName);
return(5);
};

```

```
// if it is at no copy zone, just increase no copy level by
1
if(NoCopyLevel)
NoCopyLevel++;

// it is at copy zone
else {
i = 0;
//search to determine if it is a defined variable
while(i < DefinedVariableArrayIndex) {
// if there is a match
if(strcmp(pVariable, DefinedVariableArray[i]) == 0) {
break;
} else
i++;
};
// if not found, increase no copy level by 1
if(i == DefinedVariableArrayIndex)
NoCopyLevel++;
};

// #ifdef handling
} else if( InputLine[1] == 'i' &&
InputLine[2] == 'f' &&
InputLine[3] == 'n' &&
InputLine[4] == 'd' &&
InputLine[5] == 'e' &&
InputLine[6] == 'f') { //ifdef found
// enter next level of #ifdef & #ifndef
IfdefLevel++;
// to get variable name defined by #define
pVariable = strtok(&InputLine[7], " \t\n");
if(pVariable == NULL) {
printf("\nFormat: #ifndef variable \nor VARIABLE MUST BE
ON THE SAME LINE AS #ifndef");
fclose(pFILEOutput);
fclose(pFILEInput);
gets(InputFileName);
return(6);
};

// if it is at no copy zone, just increase no copy level by
1
if(NoCopyLevel)
NoCopyLevel++;

// it is at copy zone
else {
i = 0;
//search to determine if it is a defined variable
while(i < DefinedVariableArrayIndex) {
```

```
// if there is a match
if(strcmp(pVariable, DefinedVariableArray[i++]) == 0)
{
NoCopyLevel++;
break;
};
};
};
```

```
// #else handling
} else if( InputLine[1] == 'e' &&
InputLine[2] == 'l' &&
InputLine[3] == 's' &&
InputLine[4] == 'e') { //#else is found
// if IfdefLevel = 0, error
if(IfdefLevel == 0) {
printf("Error for #else at line: %d", Line);
fclose(pFILEOutput);
fclose(pFILEInput);
gets(InputFileName);
return(7);
}
}
```

```
// at copy zone
if(NoCopyLevel == 0)
NoCopyLevel = 1;
```

```
// at no copy zone
else if(NoCopyLevel == 1)
NoCopyLevel = 0;
```

```
// #endif handling
} else if( InputLine[1] == 'e' &&
InputLine[2] == 'n' &&
InputLine[3] == 'd' &&
InputLine[4] == 'i' &&
InputLine[5] == 'f') {
//#endif is found
if(IfdefLevel == 0) {
printf("Error for #endif at line: %d", Line);
fclose(pFILEOutput);
fclose(pFILEInput);
gets(InputFileName);
return(8);
} else {
// exit one level of #ifdef & #ifndef
IfdefLevel--;
// exit one level of no copy zone if it is
if(NoCopyLevel)
NoCopyLevel--;
};
} else {
```

```
printf("\n>Error: none of '#ifndef' '#ifdef' '#ifndef'
'#else' '#endif' at line: %d\n'%d", Line, InputLine);
fclose(pFILEOutput);
fclose(pFILEInput);
gets(InputFileName);
return(9);
};
};
};
fclose(pFILEOutput);
fclose(pFILEInput);
if(IsExtension)
// input file extension
strcat(OutputFileName, Extension);
else
// default extension
strcat(OutputFileName, ".vhd");

// change file read only mode if it is, otherwise a read only file
cannot be removed
chmod(OutputFileName, _S_IWRITE);
// delete previously generated file if any
remove(OutputFileName);
// rename new output file name: InputName-
rename(pTempName, OutputFileName);
// change file read only mode if it is, otherwise a read only file
cannot be removed
chmod(OutputFileName, _S_IREAD);
// restore new working dir to old working dir
if(IsNewWorkingDir)
chdir(OldWorkingDir);
printf("\nNew file %s is established", OutputFileName);
gets(InputFileName);
return(0);
};
```

.