

# Math 157

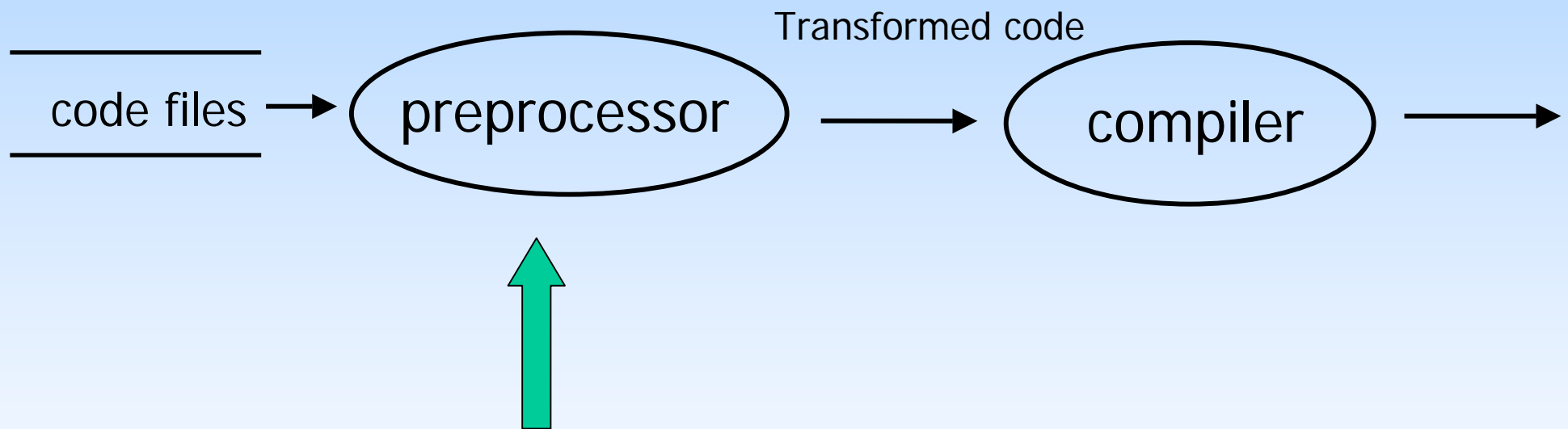
Preprocessor Directives

Feb. 5, 2007

Useful C/C++ Knowledge ...

## Compiler Directives and the Preprocessor

When you invoke the compiler, you really invoke a preprocessor & the compiler



Reads "directives" in the code or from the command line and transforms the code files

## A Partial List of Compiler Directives

`#include "afile"`

Inserts file `afile` into the code file

`#define aVar aValue`

Defines the symbol `aVar` to be `aValue`. In subsequent code processed, `aVar` is replaced by `aValue` (text replacement)

`#define aVar`

"defines" the symbol `aVar`

`#undef aVar`

"undefines" the symbol `aVar`

## A Partial List of Compiler Directives Cont.

### `#ifdef aVar`

If `aVar` is “defined” then the code between the `#ifdef` and `#endif` is evaluated by the preprocessor.

### `#endif`

### `#ifndef aVar`

If `aVar` is “undefined” then the code between the `#ifndef` and `#endif` is evaluated by the preprocessor

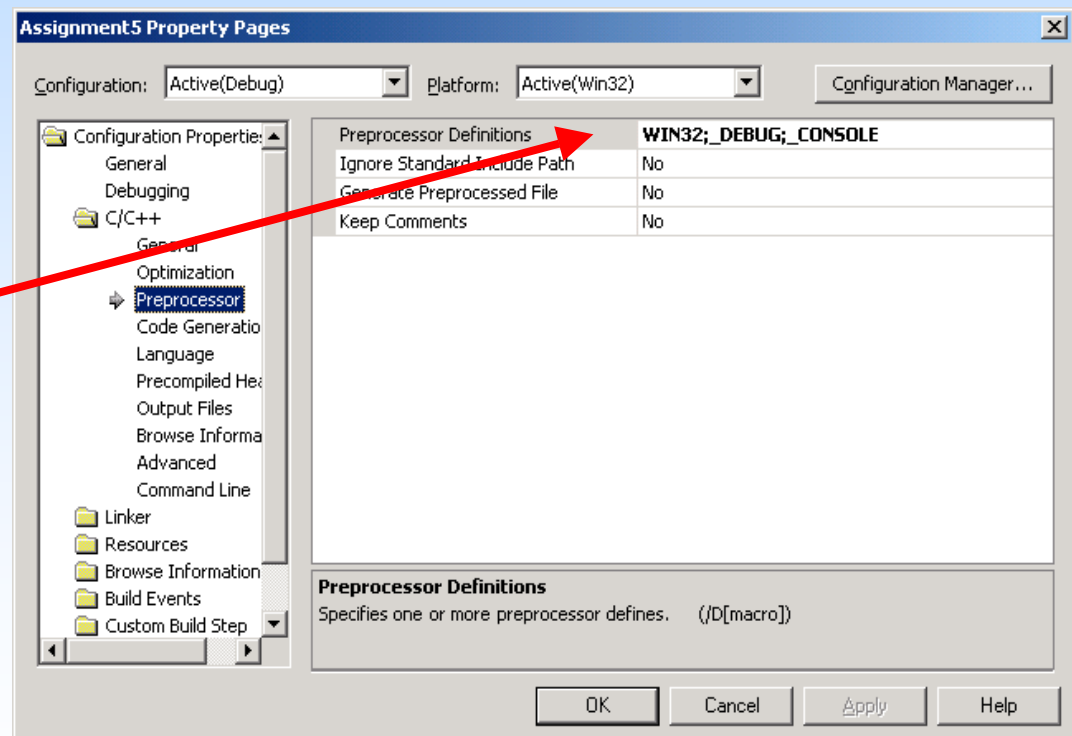
### `#endif`

# #defines specification

Preprocessor variables are defined and undefined

- (A) In code using #define or #undef
- (B) Externally using compiler options
- (C) Pre-defined by the compiler

(B)



## #defines specification cont.

(C) Pre-defined by the compiler

From the Microsoft C++ Documentation

(search `__MSC_VER`)

**`__MSC_VER`** : Defines the compiler version. Defined as 1200 for Microsoft Visual C++ 6.0 and  $\geq 1300$  for Microsoft Studio.net Always defined.

**`__cplusplus`** : Defined for C++ programs only.

\*

\*

\*

## Typical uses of compiler directives ...

- To avoid multiple inclusion of class declarations
- To create “portable” code
- To enable conditional compilation :  
creating separate code versions for “debugging” and “release”

### Example :

Avoiding multiple inclusion of class declarations ...



ClassA.h

```
ClassA
{
    *
    *
    *
};
```

ClassB.h

```
#include "ClassA.h"
ClassB
{
    *
    *
    *
};
```

main.cpp

```
#include "ClassA.h"
#include "ClassB.h"

int main()
{
    *
    *
    *
};
```

ClassA will be included twice

ERROR

ClassA.h

```
#ifndef __ClassA__
#define __ClassA__

ClassA
{
    *
    *
    *
};

#endif
```

main.cpp

```
#include "ClassA.h"
```

```
#include "ClassB.h"
```

ClassB.h

```
#include "ClassA.h"
#ifndef __ClassB__
#define __ClassB__

ClassB
{
    *
    *
    *
};

#endif
```

← Includes, defines \_\_ClassA\_\_

← \_\_ClassA\_\_ defined, so ClassA not included again

## Portability ...

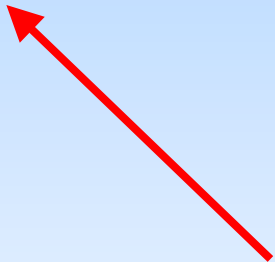
Handling compiler specific "features"

```
#ifndef _MSC_VER           // PC'c truncate the name of strstream.h
#include <strstrea.h>
#else
#include <strstream.h>
#endif
```

```
#ifndef _MSC_VER
    F.eatwhite(); ← This routine is not part of standard
                    IOstream class
#endif
```

## Conditional compilation for debugging ...

```
#ifdef _DEBUG
    T& operator()(long i1)
    {
        boundsCheck(i1, index1Begin, index1End,1);
        return *(dataPtr + (i1 - index1Begin));
    };
#else
    inline T& operator()(long i1)
    {
        return *(dataPtr + (i1 - index1Begin));
    };
#endif
```



Check index bounds  
if compiling with  
\_DEBUG defined

## Examples ...

Look at “other” code headers.

Look at system defined preprocessor directives ---  
what happens when you switch between debug and release?