Lecture 12

- Preprocessor

The Preprocessor

- Compilation: transforming a textual program description into an executable form
- Preprocessor: separate first step in compilation:
  - Remove comments
  - Macro substitution (#define)
  - Conditional compilation (#if)
  - File inclusion (#include)
- Preprocessor directive: first non-white-space character in line is #
- Only one per line

Macro Substitution

```
#define FOREVER for(;;)

FOREVER { foo(); } becomes
for (;;) { foo(); }
```

- Syntax of a macro definition:
  #define <identifier> <replacement text>
- Subsequent occurrences of the identifier in C-identifier context get replaced by the replacement text. E.g.
  xxFOREVERxx = 0; and "FOREVER" are not replaced!
- Replacement text normally is the remainder of line
- Long definitions may be continued by placing \ at the end of each line to be continued
- Scope is from point of definition to the end of current source file
- #undef <identifier> deletes definition

Macros With Parameters

```
#define index_mask 0xff00
#define extract(word,mask)((word) & (mask))

index = extract(packed_data,index_mask);

becomes

index = ((packed_data) & (0xff00));
```

- Syntax:
  #define <ident>(<ident>,...,<ident>) <text>
- Macro parameters get replaced by actual parameters when macro is expanded
- Macro expansion is done recursively until no more matches are found

More Macro Examples

```
#define FOR(i,n) for(i=0; i<(n); i++)

FOR (i, 10) { foo(i); }
becomes
for (i=0; i<(10); i++) { foo(i); }

#define MAX(a,b)  ((a)>(b)?(a):(b))
not recommended! multiple evaluation!
also, use lots of () to ensure evaluation
order!

MAX(a++,b++)
becomes
((a++)>(b++)?(a++):(b++)) OOPS! 2x a++,b++!
```


#if Statement

- Syntax & Semantics

#if <const-expr>	- true iff const-expr != 0
#ifdef <ident>	- true iff <ident> is defined
#ifndef <ident>	- true iff <ident> is undefined
#else	- alternative path
#elif <const-expr>	- else-if condition
#endif	- end of #if statement

- <const-expr> consist of macro names, integer constants, operators, parenthesis and defined(macro name).
- #error "text" - generates error msg. "text"

Conditional Compilation

```
#ifdef UNIX
  ... Unix code
#elifdef WINDOWS
  ... Windows code
#else
  #error "Unsupported OS"
#endif
```

```
#define TEST 1

#if TEST
  ... test code
#endif
```

- Compiling parts of programs depending on constant expressions. If false, program text is skipped
- Useful for dealing with different environments and debugging
- Can pass values to g++ via -D option. E.g.
  - g++ -DUNIX -DNDEBUG foo.c // UNIX,NDEBUG defined
  - g++ -DFOO=3 foo.c // FOO has value 3

File Inclusion

- Two forms:
  - #include "*filename*"
  - #include <*filename*>
- Line is replaced by the content of the file *filename*, which itself may contain #include lines
- *"filename"*: search for file begins in directory where the source program was found. If not found, search in system header directories
- *<filename>*: search file in system header directories
- Main purpose: including interface information such as function prototypes and types

#include Examples

```
#include <iostream>

std::cin, std::cout, std::cerr,
overloaded operators << >> etc. now visible


#include "mytypes.h"

User types and function prototypes defined in
local file mytypes.h now visible
```


Another #include Example

How to avoid including the same file twice which would cause compiler error msgs.?

```
mytypes.h:

#ifndef MYTYPES_H  // distinct macro for
#define MYTYPES_H  // each header file

#define FOR(i,n) for (i=0; i<(n); i++)
typedef int sint4;
typedef unsigned int uint4;

int square(int x);
int swap(int &x, int &y);
int count1(unsigned int x);
#endif
```


Testing and Debugging

```
// computes the square root of x
// precondition: x >= 0

double sqrt(double x) {
  if (x < 0) { ... complain ... }
  ... compute square root sr
  if (fabs(sr*sr-x)>EPS) { ... complain ... }
  return sr;
}
```

- Testing each function is **CRUCIAL**
- cout statements are useful to follow execution
- Pre- and post-conditions should be checked during program execution

assert Macro

- Syntax: #include <cassert>
  assert(<expression>)
- Execution stops iff the expression evaluates to 0. An error message infoms about the program file and line number where the assertion failed
- Check can be turned off by defining NDEBUG before #include <cassert>
  (usually done with compiler option -DNDEBUG in makefile)
- Turn assert on when debugging program
- Turn off to speed up execution when sure that code is functional

assert Example

```
//#define NDEBUG  // uncomment to turn assert
      // checks off or pass -DNDEBUG to g++
#include <cassert>

// computes the square root of x
// precondition: x >= 0

double sqrt(double x) {
  assert(x >= 0); // pre-condition

  ... compute square root sr

  assert(fabs(sr*sr-x)<=EPS); // post-cond.
  return sr;
}
```