VULNERABILITY of macro definitions

Ex:

```
# define max(a, b)   a > b ? a : b
# define square(a)   a * a
```

Sample Invokations

(a) `x = max( y, z);`

i.e `x = y > z ? y : z;` ok

(b) `x = max (y, z) + 10;`

i.e. `x = y > z ? y : z + 10;` Not ok !

NOTE: Parts of a macro definition are vulnerable at the right & left corners to other operators.

Expected Result

`x = (y > z ? y : z) + 10;`

Hence, use:

```
# define max(a, b) (a > b ? a : b)
```

protective parantheses

(c) `x = square( y + 4);`

i.e. `x = y + 4 * y + 4` NOT ok !

vulnerability

NOTE that a parameter can be an expression & must be protected

NOTE Macro Invokation must be functionally equivalent to Procedure Invokation

PROFESSIONAL STYLES

```
# define max(a, b)  ((a) > (b) ? (a) : (b))
# define square(a)  ((a) * (a))
```

STRUCTS AND MACROS

Collect a few sample statements:

```
# define max(a, b)  ((a) > (b) ? (a) : (b))
# define square(a)  ((a) * (a))
struct emp {
   — —
   — —
};
# define NEMP 20
# define INC 20
# define CO_NAME "Uttara"
```

(statements are part of a payroll package)

SOME MODULES of the package

payenter.c : to enter details of employees

payslip.c : to print pay slips

paybank.c : to print Bank statements (Bankwise Details)

Questions:

Which of these modules need "struct emp" ?

Which of these need NEMP ?

Which of these need INC ?

ANS: probably all ! Even "max" may be needed.

Question:

Are we going to have copies of these statements in multiple modules ?

ANS: No! We need a new syntax under CPP.

INFORMATION CLASSIFICATION

struct emp, NEMP, INC : need in payroll

NEMP : probably not needed in Inventory

CO_NAME : need in all packages

information classification

```
# define   ITAX_SLAB  30    : Indian Govt. info
# define   SALES_TAX  15    : State Govt. info
```

NEED to modify symbolic literal values on decisions by:

(a) Indian Govt.
(b) State Govt.
(c) Board of Directors
(d) Finance Manager
(e) Personnel Manager, etc.

STATEMENT PLACING

Ex:

```
x = z;
y = w;
```

Q: Can these be interchanged?

NOTE that any number of statements may get inserted in between and effect the program.

RULE Every statement of a package must be in a specific place in a specific module.

INCLUDES ; HEADERS

to avoid repetitions

Ex: File "pay.h" has definitions for:

struct emp , NEMP , INC

Use: #include "pay.h"

in files payslip.c
paybank.c
payenter.c, etc.

and also finance.c

Probably "inven.c" does not need to include pay.h

ALSO, the literal CO_NAME needed by

payslip.c , inven.c , ledger.c , etc.

Must not define CO_NAME in pay.h

We need:

"company.h"

to hold all company related informations
(so that they can be conveniently edited)

THE MODULES needed

pay.h : payroll related definitions
company.h : company related info
state.h : state related info.
(good for porting to other states)

india.h : Central Govt. informations

STATEMENT SEQUENCE

payenten.c :

```
# include "pay.h"
# include "company.h"
```

Q: Is this the correct sequence ?

A: company.h may have a literal that is used by pay.h

HENCE most modules use :

```
# include "india.h"
# include "state.h"
# include "company.h"
# include "pay.h"
```

etc.

HEADER FILES

Any statements before

```
# include "pay.h"
```

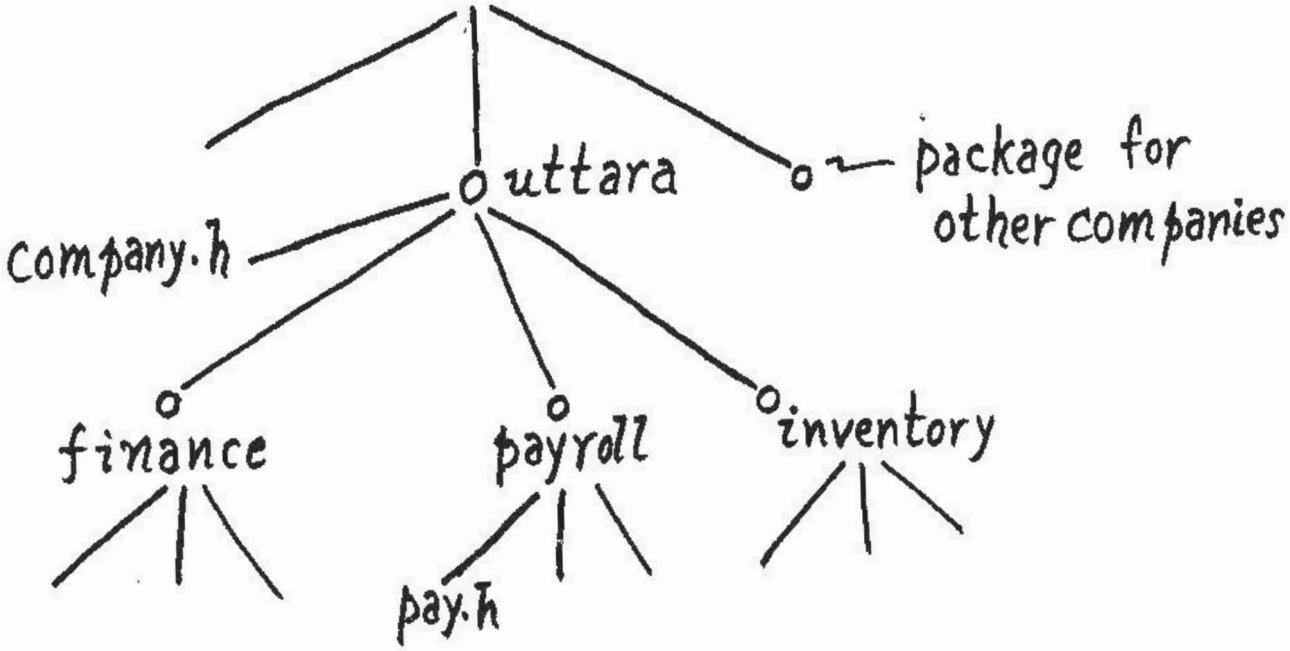
cannot make use of symbols in pay.h.

Universal Practice to place includes at top of files.

More includes ⇒ program is more professional.

DIRECTORY STRUCTURE

We may have confidential information.



NOTE: "company.h" not in package directory to avoid multiple copies.

directory structure

The statement

```
# include "company.h"
```

does not work in payslip.c. We have to use:

```
# include "../company.h"
```

Next: stdio.h is included by most C programs.

Q: How to have a single copy ?

STANDARD HEADERS like stdio.h

standard place: /usr/include

Include Formats

```
# include "../company.h"
# include "pay.h"
# include <stdio.h>
```

Double Quotes: Hunt in current directory

Angular Bnackets: Hunt in standard place

UTILITY MACROS

Q: Where should we define max ?

utility macros

max, min, etc. are
small useful jobs

and are best defined as macros. OUR macros in:

/usr/include/util/utils.h

NOW, # include <utils.h> does not work!
include <util/utils.h> is ok.

Another Mechanism

Have:
include <utils.h>

and compile using:
cc -I/usr/include/util ...

NESTED INCLUDES

How many
include "../company.h" ?

NOTE: any module that needs "pay.h"
definitely needs "../company.h"

nested includes

TRY:

payslip.c has only

```
# include "pay.h"
```

and pay.h has:

```
# include "../india.h"      } could be in
# include "../state.h"      }   govt.h
# include "../company.h"
struct emp {
 - - - -
 - - - -
```

Even this leads to repetition of statements.

TRY:

payslip.c:
```
# include "pay.h"
 - - -
```

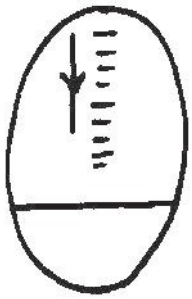
pay.h:
```
# include "../company.h"
struct emp {
  - - -
```

company.h:
```
# include "../state.h"
# define CO_NAME ..
```

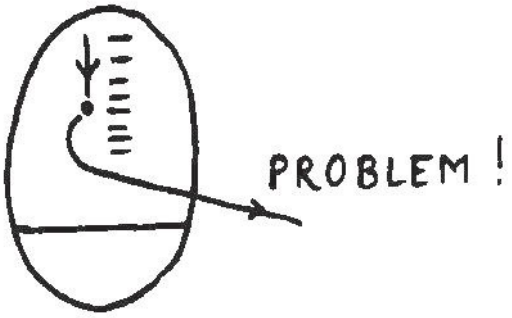
state.h:
```
# include "../india.h"
# define SALES_TAX ...
```

TESTABILITY

CPU executing text



CPU goes for a walk!



NOTE: Anonymous Accesses create problems.

CPU ACCESS FAULT

Single User System

Use CTRL ALT DEL
OR Reset System
OR Switch off
& on.

Multi User System

mmu generates
segmentation violation
& process is killed

Analogy: Ask driver to go to Mysore & come back
He does not come back.
Q: Where was the accident ?
When & How did it take place ?

A Method: Ask him to make a series of phone calls.
Binary Search can be used

ADB ; SDB

A process has
- text segment
- data segment
- entry in process table

Segmentation Violation saves the Process Image as file "core" in current directory.

adb(1) and sdb(1) can do post mortem analysis.

But they are difficult to use.

(NOTE: Best person to test is Programmer)

CTRACE

ctrace(1) is like the phone call method.

adb : very little info.

ctrace: too much info.

CHECK POINTS

dangerous areas in text

Ex: A tricky anonymous access

USE: printf's before & after

check points

Sample Messages	Better Messages
check point 1	Going to search item
check point 2	Finished searching
.....	.....

Method : Sprinkle few printf's

Can also print values: check point 3, x = ...

Software Delivery : All check point messages must be removed.

For Quick finding of messages,

use:

```
check point : —
check point : —
```

OR better:

```
ZZcheck point : —
ZZ check point : —
```

Problems

(a) Removal of so many lines

(b) Maintenance of s/w.

Removal of lines destroys knowledge of testability

UNCOMMENTING

Use:

(1) /* Uncomment next line for testing */
/* printf ("ZZcheck .... "); */

(2) /* Comment next line for despatch */
printf ("ZZCheck .... ");

Problem: Method is very laborious.

CONDITIONAL COMPILATION

Testing printf's to be present but not compiled:
Ask preprocessor to swallow them!

Ex:

```
# define TESTING          no value given
- - -
# ifdef TESTING   OR  # if defined(TESTING)
   printf ("ZZCheck...");
# endif
```

Also, use:

```
# ifndef TESTING   OR  # if !defined (TESTING)
```

for conditional to be false

conditional compilation

Now: Comment only one Time:

```
# define TESTING
```

BUT: suppose there are many modules.

USE:

```
#define TESTING
```

only in "india.h".

BEST: No need to have

```
# define TESTING
```

at all and use

```
cc - DTESTING .. ..
```

for compilation

SOME CC OPTIONS

```
cc -E ...        preprocess only (for macro faults)
cc -S ...        generate assembly code
cc -DTESTING ..  predefined symbol
cc -DNEMP=30 ..  predefined symbolic literal
cc -g ...        tracing : adb/sdb can give
                           better reports
```

PORTABILITY across different environments

Ex: OS — DOS, UNIX, Windows

Q: Can same pgm be compiled ?

Ex: Superfast → SuperCompact without removing macro definitions

Sample Code

```
# ifdef SPEED
   macro definitions
# else
   procedures
# endif
```

USE:

CC -DSPEED ...

OR CC -DSPACE ...

Sample Code

```
# if defined (UNIX)
    --- code specific to UNIX ---
# elif defined (DOS)
    --- code specific to DOS ---
# elif defined (WINDOWS)
    --- code specific to WINDOWS ---
# endif
```

Ex: # if defined (UNIX) && defined(SPEED)

operators : && || !