Preprocessor Statements

R. Inkulu
http://www.iitg.ac.in/rinkulu/

File inclusion

abc.h:

int func1(float);
float func2(double, double);
. . .

abc.c:

#include "abc.h"
int func1(float) { . . . }
float func2(double, double) {
. . . }
. . .

main.c:

#include "abc.h"
int main(void) {
int k = func1(35.678);
}

abc.c after compiler pre-processing:

int func1(float);
float func2(double, double);
. . .
int func1(float) { . . . }
float func2(double, double) {
. . . }
. . .

main.c after compiler pre-processing:

int func1(float);
float func2(double, double);
. . .
int main(void) {
int k = func1(35.678);
}

Macro Substitution

```
...
#define ARRAYSIZE 512
...
int func(double) {
...
int buffer1[ARRAYSIZE];
...
double buffer2[ARRAYSIZE];

...
char buffer3[ARRAYSIZE];
...
}
```

```
int func(double) {
...
int buffer1[512];
...
double buffer2[512];
...
char buffer3[512];
...
}
```

- Errors due to duplication can be eliminated

Macro with arguments: parenthesization issue

```
...
#define SQUARE(x) x*x
...
int func(void) {
...
double f = SQUARE(23.89);
...
int i = 2, j = 3;
int k = SQUARE(i+j);
...
}
```

```
int func(void) {
...
double f = 23.89*23.89;
...
int i = 2, j = 3;
int k = i+j*i+j;
...
}
```

- precedence issue with the second substitution

Macro with arguments: local fix

```
...
#define SQUARE(x) x*x
...
int func(void) {
...
double f =
SQUARE((23.89));
...
int i = 2, j = 3;
int k = SQUARE((i+j));
...
}
```

```
int func(void) {
...
double f = (23.89)*(23.89);
...
int i = 2, j = 3;
int k = (i+j)*(i+j);
...
}
```

Macro with arguments: global fix

```
...
#define SQUARE(x)
((x)*(x))
...
int func(void) {
...
double f = SQUARE(23.89);
...
int i = 2, j = 3;
int k = SQUARE(i+j)/7;
...
}
```

```
int func(void) {
...
double f = ((23.89)*(23.89));
...
int i = 2, j = 3;
int k = ((i+j)*(i+j))/7;
...
}
```

Macro with arguments: unavoidable side effect

```
...
#define SQUARE(x)
((x)*(x))
...
int func(void) {
...
double f = SQUARE(23.89);
...
int i = 2, j = 3;
int k = SQUARE(++i);
assert (i == 3);
...
}
```

```
int func(void) {
...
double f = ((23.89)*(23.89));
...
int i = 2, j = 3;
int k = ((++i)*(++i));
assert (i == 3);
...
}
```

- in the second substitution, i is incremented twice → could be an unexpected result

if statement and a macro

incorrect code: semantics changed

```
. . .
#define SWAP(x, y, w) w=x;
x=y; y=w;
. . .
int func(void) {
. . .
int i=3,j=8,t=-1;
if (i > j) SWAP(i, j, t);
. . .
printf("%d,%d", i, j);
. . .
}
```

```
int func(void) {
. . .
int i=3,j=8,t=-1;
if (i > j) t=i; i=j; j=t;;
. . .
printf("%d,%d", i, j);
//prints 8, -1 ! . . .
}
```

if-else statement and a macro: poor fix

incorrect code: leads to compile-time error

```
...
#define SWAP(x, y, w)
{w=x; x=y; y=w;}
...
void func2(void) { ...}
int func(void) {
...
int i=3,j=8,t=-1;
if (i > j) SWAP(i, j, t);
else func2();
...
}
```

```
int func(void) {
...
int i=3,j=8,t=-1;
if (i > j) {t=i; i=j; j=t;};
else func2();
...
}
```

- semicolon after SWAP(i, j) is the reason for havoc

if-else statement and a macro: right fix

```
...
#define SWAP(x, y, w) do
{w=x; x=y; y=w;} while (0)
...
void func2(void) { ...}
int func(void) {
...
int i=3,j=8,t=-1;
if (i > j) SWAP(i, j, t);
else func2();
...
}
```

```
int func(void) {
...
int i=3,j=8,t=-1;
if (i > j) do {t=i; i=j; j=t;}
while(0);
else func2();
...
}
```

homework: bring the same effect with an if-else statment instead of using a do-while(0)

Quoted string macro

```
...
#define DPRINT(expr)
printf("debug start " #expr
"=%d", expr)
...
int func(void) {
...
int x = 8, y = 4;
DPRINT(x/y);
...
}
```

```
int func(void) {
...
int x = 8, y = 4;
printf("debug start " "x/y"
"= %d", x/y);
...
}
```

```
int func(void) {
...
int x = 8, y = 4;
printf("debug start
x/y=%d", x/y);
...
}
```

Macros with arguments vs Functions

adv with macros:

- no func invocation cost in the runtime, hence faster
- useful in making small functions inline

disadv with macros:

- size of object file increases
- no type checking of parameters
- side-effects

  multiple prefix/postfix operations per substitution
  precedence issues
  using macro between if-else keywords

- difficult to debug while having breakpoints within a macro

Few popular macros

- getchar
- putchar
- va_start
- va_arg
- va_end
- __DATE__ date of compilation
- __TIME__ time of compilation
- __LINE__ line number
- __FILE__ name of file

Duplicate inclusion due to multiple header file inclusions

abc1.h:
```
typedef struct {
...
} Home;
extern int globalVar;
...
```
abc2.h:
```
#include "abc1.h"
...
Home func2(double, double);
...
```
abc3.h:
```
#include "abc1.h"
...
void func3(Home);
...
```
abc4.c:
```
#include "abc2.h"
#include "abc3.h"

int main(void) {
...
Home home = func2(35.67,
89.05);
...
func3(home);
...
}
```

Conditional compilation

abc1.h:
```
#ifndef __FILEABC1__
#define __FILEABC1__
typedef struct {
. . .
} Home;
extern int globalVar;
. . .
#endif
```
abc2.h:
```
#include "abc1.h"
. . .
Home func2(double, double);
. . .
```
abc3.h:
```
#include "abc1.h"
. . .
void func3(Home);
```
abc4.c:
```
#include "abc2.h"
#include "abc3.h"

int main(void) {
. . .
Home home = func2(35.67,
89.05);
func3(home);
. . .
}
```

Conditional compilation (cont)

```
...
#if SYSTEMTYPE == LINUX
...
#elif SYSTEMTYPE == SOLARIS
...
#elif SYSSTEMTYPE == WINDOWS
...
#else
...
#endif
...
```

- If the system is LINUX, preprocessor gives the code block listed after LINUX test and before SOLARIS test to compiling phase → hence, the term *conditional compilation*

Undefining a #define

```
#define ARRAYSIZE 20
...
#undef ARRAYSIZE
#define ARRAYSIZE 30
...
#undef getchar
int getchar(void) { ...}
...
```

Pragma Directives

```
...
#pragma setlocale("dutch")
...
```

```
...
#pragma optimize(on)
...
#pragma optimize(off)
...
```

- pragma statement gives additional information to compiler