

## Outline

- Monday: design, interfaces,  
representation of information
- Tuesday: testing, debugging,  
mechanization
- Thursday: programming style

1

## The good old days...

```
DO 14 I = 1,N
DO 14 J = 1,N
14 V(I,J) = (I/J)*(J/I)
```

```
DO 14 I = 1,N
DO 12 J = 1,N
12 V(I,J) = 0.0
14 V(I,I) = 1.0
```

```
V(1:N,1:N) = 0.0
DO I = 1,N
V(I,I) = 1.0
END DO
```

2

## Programming style

- low-level expression
  - control flow
  - idioms
  - boundary conditions
  - defensive programming
  - comments
  - design and organization
- 
- principles: simplicity, clarity, consistency

3

## Branching around

```
GRVAL = A(1)
DO 25 I = 2,10
  IF A(I).GT.GRVAL) GO TO 30
  GO TO 25
30 GRVAL = A(I)
25 CONTINUE
```

```
GRVAL = A(1)
DO 25 I = 2,10
  IF (A(I).GT.GRVAL) GRVAL = A(I)
25 CONTINUE
```

```
GRVAL = MAXVAL(A)
```

4

## de Morgan Rules!

```
DO I = 1, N
  IF ((Sensor(I) >= 10) .and. (Sensor(I) <= 20)) THEN
    CYCLE
  END IF
  Sensor(I) = Sensor(I) + 1
  modifyFlag = .true.
END DO
```

“The code above could have eliminated the CYCLE by writing:

```
DO I = 1, N
  IF ((Sensor(I) < 10) .and. (Sensor(I) > 20)) THEN
    Sensor(I) = Sensor(I) + 1
    modifyFlag = .true.
  END IF
END DO
```

5

## Too clever...

```
subkey = subkey >> (bitoff -((bitoff >> 3) << 3));
```

```
subkey = subkey >> (bitoff & 07);
```

```
subkey >>= bitoff & 07;
```

6

## Too cryptic...

```
newchild=(!EMPLC&&!EMPRC)?0:(!EMPLC?EMPRC:EMPLC);
```

```
if (EMPLC == 0 && EMPRC == 0)
    newchild = 0;
else if (EMPLC == 0)
    newchild = EMPRC;
else
    newchild = EMLPC;
```

7

## Not clever enough...

```
switch (bit) {
case 0 : mask[byte] = mask[byte] | 128; break;
case 1 : mask[byte] = mask[byte] | 64; break;
case 2 : mask[byte] = mask[byte] | 32; break;
case 3 : mask[byte] = mask[byte] | 16; break;
case 4 : mask[byte] = mask[byte] | 8; break;
case 5 : mask[byte] = mask[byte] | 4; break;
case 6 : mask[byte] = mask[byte] | 2; break;
case 7 : mask[byte] = mask[byte] | 1; break;
}
```

8

## Not clever enough...

```
switch (bit) {
case 0 : mask[byte] = mask[byte] | 128; break;
case 1 : mask[byte] = mask[byte] | 64; break;
case 2 : mask[byte] = mask[byte] | 32; break;
case 3 : mask[byte] = mask[byte] | 16; break;
case 4 : mask[byte] = mask[byte] | 8; break;
case 5 : mask[byte] = mask[byte] | 4; break;
case 6 : mask[byte] = mask[byte] | 2; break;
case 7 : mask[byte] = mask[byte] | 1; break;
}

mask[byte] |= 1 << (7 - bit);
```

9

## Too dumb?

```
/* Check for a blank line */
blank_flag = True;

i = 1;
while ((i <= line_length ) &&
      (blank_flag == True) )
{
    /* While */
    if (line[i] != ' ')
        blank_flag = False;
    else
        i++;
}
/* While */

blank_flag = strchr(line, " ") == 0;
```

10

## Decision, decisions...

```
if (STRT_OF_BUF(c))
  if (GOOD_POINTER(rp = prev_block(rp->r_fn, rp->r_ibn, c-rp->r_bufstart)))
    if (BOUNDARY(rp))
      if (second_nl) return(extract_line(rp, rp->r_scanpt));
      else if (GOOD_POINTER(la = exp_blink(rp->r_fn)))
        if (GOOD_POINTER(rp = get_block(la->la_fn, la->la_block)))
          c = rp->r_scanpt+la->la_offset;
          else return((struct LINE *) 0);
        else return(sof_line(rp->r_fn, rp->r_ibn, rp->r_scanpt-rp->r_bufstart));
      else c = rp->r_scanpt; /*Continue Scan*/
    else return((struct LINE *) 0); /*Retrieval Error*/
else c--; /*Continue scan*/
```

11

## If ... If ... If ...: Bad Bad Bad

```
if (argc == 3)
  if ((fin = fopen(argv[1], "r")) != NULL)
    if ((fout = fopen(argv[2], "w")) != NULL) {
      while ((c = getc(fin)) != EOF)
        putc(c, fout);
      fclose(fin); fclose(fout);
    } else
      printf("Can't open output file %s\n", argv[2]);
  else
    printf("Can't open input file %s\n", argv[1]);
else
  printf("Usage: cp inputfile outputfile\n");
```

12

## Every time you make a decision, do something

```
if (argc != 3)
    printf("Usage: cp inputfile outputfile\n");
else if ((fin = fopen(argv[1], "r")) == NULL)
    printf("Can't open input file %s\n", argv[1]);
else if ((fout = fopen(argv[2], "w")) == NULL) {
    printf("Can't open output file %s\n", argv[2]);
    fclose(fin);
} else {
    while ((c = getc(fin)) != EOF)
        putc(c, fout);
    fclose(fin);
    fclose(fout);
}
```

13

## Idioms

```
for (i = 0; i <= n-1; )
    array[i++] = 0;

for (i = 0; i < n; )
    array[i++] = 0;

for (i = n; --i >= 0; )
    array[i] = 0;

for (i = 0; i < n; i++)
    array[i] = 0;
```

14

## Why idioms matter

- if there's a violation, it's often an error

```
int *array = calloc(n, sizeof(int));
for (i = 0; i <= n; i++)
    array[i] = i;
```

```
int *array = calloc(n, sizeof(int));
for (i = 0; i < n; i++)
    array[i] = i;
```

15

## Why idioms matter (2)

```
p = malloc(strlen(s));
strcpy(p, s);
```

```
p = malloc(strlen(s)+1);
strcpy(p, s);
```

16

## Why idioms matter (3)

```
do {  
    c = getchar();  
    putchar(c);  
} while (c != EOF);
```

```
while ((c = getchar()) != EOF)  
    putchar(c);
```

17

## Fortran idioms?

- Fortran 90/95 is so different from Fortran 77 that new idioms are called for
  - formatting and commenting
  - control flow
  - array operations
  - dynamic storage
  - modules
  - ...
- watch out for array-origin differences between Fortran and C

18

## Avoid the bad features of a language

```
      IF(KA-KB) 5, 5, 4
4 KR = KA
  KA = KB
  KB = KR
5 IF(KA) 6, 7, 6
6 KR = KB - KB/KA*KA
  KB = KA
  KA = KR
  GO TO 5
7 PRINT 102, KB

5 IF (KA .EQ. 0) GOTO 7
  KR = MOD(KB,KA)
  KB = KA
  KA = KR
  GOTO 5
7 PRINT 102, KB
```

19

## Use the good features

```
5 IF (KA .EQ. 0) GOTO 7
  KR = MOD(KB,KA)
  KB = KA
  KA = KR
  GOTO 5
7 PRINT 102, KB

DO WHILE (KA /= 0)
  KR = MOD(KB,KA)
  KB = KA
  KA = KR
END DO
PRINT "(15)", KB
```

20

## Avoid the bad features

```
#define SIZE 10

main() {
    static char number[SIZE];

    puts("Enter an integer, please.");
    gets(number);
    if (number[SIZE - 1] != '\0' ) {
        puts("Too many digits; you wiped me out.");
        exit(1);
    }
    ...
}
```

(from a C textbook)

21

## Language pitfalls...

- how to interchange 2 bytes of a `short`:

```
#define flip2(a) \
    (((a) & 0xFF) << 8 + ((a) & 0xFF00) >> 8)
```

```
#define flip2(a) \
    ((a) << 8 | ((a) >> 8) & 0xFF)
```

22

## Defensive programming

```
char s[64];  
    ...  
sprintf(s, "There's no box called '%s' in  
    this schematic type.",
```

23

## Defensive programming (2)

```
char postString[ 1024 ];  
  
contentLength = atoi(getenv( "CONTENT_LENGTH" ));  
cin.read( postString, contentLength );
```

from a C++ book (4th edition, 2003)

- program defensively  
"Always validate all your inputs -- the world outside your function should be treated as hostile and bent upon your destruction."

Howard & LeBlanc, *Writing Secure Code*, p 80

24

## Boundary conditions

- what's the obvious boundary?

```
strcpy(char *dest, char *source)
    /* copy source to dest */
{
    int i;

    for (i = 0; source[i] != '\0'; i++)
        dest[i] = source[i];
}
```

25

## Boundary condition testing

- what's the obvious boundary?

```
strcpy(char *dest, char *source)
    /* copy source to dest */
{
    int i;

    for (i = 0; source[i] != '\0'; i++)
        dest[i] = source[i];
    dest[i] = '\0';
}
```

26

## Don't let your pointers dangle

```
char *combine(char *s, char *t)
    /* Return a char pointer to caller */
{
    int x, y;      /* s and y are ints */
    char r[100];  /* r is a char array with 100 bytes */

    strcpy(r, s); /* copy string pointed to by s into r */
    y = strlen(r); /*get the length of the string to y */

    for (x = y; *t != '\0'; ++x) /* count x starting at x=y */
        r[x] = *t++; /* read byte from t into r */
    r[x] = '\0';      /* tack on NULL */

    return(r); /* return a pointer to the internal array */
}
```

27

## Comments

```
// Ignore all the signals that are caught to prevent
// clean_up routine from being interrupted.
// ignore the SIGHUP signal
(void)signal(SIGHUP,SIG_IGN);
// ignore the SIGTERM signal
(void)signal(SIGTERM,SIG_IGN);
// ignore the SIGPIPE signal
(void)signal(SIGPIPE,SIG_IGN);
// ignore the SIGINT signal
(void)signal(SIGINT,SIG_IGN);
// ignore the SIGQUIT signal
(void)signal(SIGQUIT,SIG_IGN);
```

28

## Comments (2)

```
/*  
 * default  
 */  
default:  
    break;
```

29

## Comments (3)

```
/* return SUCCESS */  
return (SUCCESS);
```

30

## Comments (4)

```
return;          /* back to caller */
```

31

## Comments (5)

```
C INCREMENT ZERO ENTRY COUNTER  
  ZEROCOUNT = ZEROCOUNT + 1
```

32

## Comments (6)

```
/******  
/* Update total lines */  
lines = lines + 1;  
/******
```

33

## Design and organization

- interface issues
  - what are the pieces
    - libraries, objects, components, ...
    - re-use versus reinvention (make versus buy)
  - what do they do
  - what do they hide
  - how do they work together

34

## Reuse instead of reinvention

a sample from one part of a big system:

```
void bytcpy(uchar *dptr, uchar *sptr, short n)
bytcpy(uchar *dst, uchar *src, long n)
void copy (uchar *entries, uchar *destptr, short n)
void copy(long nin, long nout, char *in, char *out)
copy(char *from, char *to, short n)
short copy(short *from, short *to, short n)
...
```

all are handled by

```
void *memmove(void *dst, const void*src, size_t n)
```

35

## Consistency? (from stdio.h)

```
int fprintf(FILE *fp, const char *fmt, ...);
int fputs(const char *buf, FILE *fp);

char *fgets(char *buf, int, FILE *fp);
int fscanf(FILE *fp, const char *fmt, ...);
```

36

## Consistency? (from winbase.h)

```
DWORD GetCurrentDirectoryA(
    DWORD nBufferLength, LPSTR lpBuffer);

UINT GetSystemDirectoryA(
    LPSTR lpBuffer, UINT uSize);

n = GetCurrentDirectory(200, s1);

n = GetSystemDirectory(s2, 200);
```

37

## Control flow or data?

```
if (!(new = append_char(ch++,50,5))) /* height will be 10 */
    return (false);
if (!(new = append_char(ch++,50,10))) /* force height=10 */
    return (false);
/* line 2 is 10 high, 3 chars, ...: */
if (!(new = append_char(ch++,1,1))) return (false);
if (!(new = append_char(ch++,98,1))) return (false);
if (!(new = append_char(ch++,1,10))) return (false);
if (!(new = append_char(ch++,25,1))) return (false);
if (!(new = append_char(ch++,25,1))) return (false);
if (!(new = append_char(ch++,25,1))) return (false);
if (!(new = append_char(ch++,25,1))) return (false);
if (!(new = append_char(ch++,1,10))) return (false);
return (true);
```

38

## Put regularity in control flow, irregularity in data

```
static int xy[]={
    50, 5, /* height will be 10 */
    50, 10, /* force height=10 */
    1, 1, /* line 2 is 10 high, 3 chars, ... */
    98, 1,
    1, 10,
    25, 1,
    25, 1,
    25, 1,
    25, 1,
    1, 10,
    -1, -1,
};
...
for (i = 0; xy[i] != -1; i += 2)
    if ((new = append_char(ch++, xy[i], xy[i+1])) == 0)
        return false;
return true;
```

39

```
case NCP100: /* DSDC */
    ccode1 = 0x10;
    ccode2 = 0x0C;
    service = DSD;
    rec_flg.fields.spec_flg = regptr->oc.ocmp.setb;
    if ((regptr->oc.ocmp.fred_ind == 1) && (regptr->rrannex.dsd.fred_rcv
/* A800 Suppl Data */
        cix = CIFRED;
    }
    else {
        if (regptr->oc.ocmp.qtime == 0 && regptr->oc.ocmp.sid_ind == 0
            && regptr->oc.ocmp.uui == 0
            && regptr->oc.ocmp.data_rate == 0) {
            cix = CIDSQ;
        }
        else {
            cix = CIDSQ;
        }
    }
    break;
```

40

```

case NCP324:          /* I800 Inbound    */
    ccode1 = 0x32;
    ccode2 = 0x4C;
    service = DSD;
    rec_flg.fields.spec_flg = regptr->oc.ocmp.setb;
    if ((regptr->oc.ocmp.fred_ind == 1) && (regptr->rrannex.dsd.fred_rcv
/* A800 Suppl Data */
        cix = CIFRED;
    }
    else {
        if (regptr->oc.ocmp.qtime == 0 && regptr->oc.ocmp.sid_ind == 0
            && regptr->oc.ocmp.uui == 0
            && regptr->oc.ocmp.data_rate == 0) {
                cix = CIDSD;
            }
            else {
                cix = CIDSQ;
            }
        }
    }
    break;

```

41

```

case NCP325:          /* Megacom 800    */
    ccode1 = 0x32;
    ccode2 = 0x5C;
    service = DSD;
    rec_flg.fields.spec_flg = regptr->oc.ocmp.setb;
    if ((regptr->oc.ocmp.fred_ind == 1) && (regptr->rrannex.dsd.fred_rcv
/* A800 Suppl Data */
        cix = CIFRED;
    }
    else {
        if (regptr->oc.ocmp.qtime == 0 && regptr->oc.ocmp.sid_ind == 0
            && regptr->oc.ocmp.uui == 0
            && regptr->oc.ocmp.data_rate == 0) {
                cix = CIDSD;
            }
            else {
                cix = CIDSQ;
            }
        }
    }
    break;

```

42

```

case NCP326:      /* I800 Outbound */
    ccode1 = 0x32;
    ccode2 = 0x6C;
    service = DSD;
    rec_flg.fields.spec_flg = regptr->oc.ocmp.setb;
    if ((regptr->oc.ocmp.fred_ind == 1) && (regptr->rrannex.dsdc.fred_rcv
/* A800 Suppl Data */
        cix = CIFRED;
    }
    else {
        if (regptr->oc.ocmp.qtime == 0 && regptr->oc.ocmp.sid_ind == 0
            && regptr->oc.ocmp.uui == 0
            && regptr->oc.ocmp.data_rate == 0) {
            cix = CIDSQ;
        }
        else {
            cix = CIDSQ;
        }
    }
}
break;

```

43

```

void dictionary::insert( char* w ) {
// Returns 1 if w in dictionary otherwise returns 0
    unsigned int majorkey, minorkey, tablevalue, len;
    if (strlen(w) != 0) {
        len = strlen(w);
        if (len <= 3) { /* Table lookup */
            if (len == 1) small[w[0] - c4] = 1;
            else if (len == 2)
                small[w[0] - c4 + c2 * (w[1]-c4)] = 1;
            else
                small[w[0] - c4 + c2 * (w[1]-c4) + c3 * (w[2]-c4)] = 1;
        } else { /* Virtual hashing */
            compute(w, majorkey, minorkey);

            tablevalue = hash[majorkey];
            while ( (tablevalue != 0) && (tablevalue != minorkey) ) {
                majorkey = (majorkey+1) % hashsize;
                tablevalue = hash[majorkey];
            }
            if (tablevalue == minorkey) return;
            else
                hash[majorkey] = minorkey;
        }
    }
}

```

44

```

int dictionary::check( char* w ) {
// Returns 1 if w in dictionary otherwise returns 0
  unsigned int majorkey, minorkey, tablevalue, len;
  if (strlen(w) != 0) {
    len = strlen (w);
    if (len <= 3) { /* Direct table lookup for small words */
      if (len == 1) return(small[w[0] - c4]);
      else if (len == 2)
        return(small[w[0] - c4 + c2 * (w[1]-c4)]);
      else
        return(small[w[0] - c4 + c2 * (w[1]-c4) + c3 * (w[2]-c4)]);
    } else { /* Virtual hashing */
      compute(w, majorkey, minorkey);
      majorkey = majorkey % hashsize;
      tablevalue = hash[majorkey];
      while ( (tablevalue != 0) && (tablevalue != minorkey) ) {
        majorkey = (majorkey+1) % hashsize;
        tablevalue = hash[majorkey];
      }
      return(tablevalue == minorkey);
    }
  }
}

```

45

## Why bother about style?

- who cares about style if the program works?
- it take too much time to fix it up
- style rules interfere with programmer freedom
- the rules are arbitrary anyway
- etc., etc., etc.

programming style matters:

if the little things are bad,  
the big ones will be bad too

46