# Fall 2006 Handout 6

**The world’s only ternary operator: K&R pp. 51−52; King pp. 72−73**

```
    if (a == b) {
        i = j;
    } else {
        i = k;
    }

    i = (a == b ? j : k);        /* parentheses unnecessary here */
```

$$\boxed{\boxed{\boxed{\texttt{a}}\ \texttt{==}\ \boxed{\texttt{b}}}\ \texttt{?}\ \boxed{\texttt{j}}\ \texttt{:}\ \boxed{\texttt{k}}}$$

```
    if (d == 1) {
        printf("%d dollar\n", dollar);
    } else {
        printf("%d dollars\n", dollar);
    }

    printf("%d dollar%s", d, d == 1 ? "" : "s");
```

```
    if (day == 1) {
        printf("A partridge in a pair tree.\n");
    } else {
        printf("And a partridge in a pair tree.\n");
    }

    printf(day == 1 ? "A partridge in a pair tree.\n" :
        "And a partridge in a pair tree.\n");

    printf("%s partridge in a pear tree.\n", day == 1 ? "A" : "And a");
```

The **`?:`** operator was invented primarily for use in **`#define`**’s. It lets you pack an **`if-then-else`** into a single expression:

```
/* Absolute value of a number. */
#define ABS(a)       ((a) >= 0 ? (a) : -(a))

/* Maximum of two numbers.  See K&R p. 89. */
#define MAX(a, b)    ((a) > (b) ? (a) : (b))

/* Nested ?:’s have left-to-right associativity. */
#define SIGNUM(i)    ((i) < 0 ? -1 : (i) > 0 ? 1 : 0)

/* It’s always redundant to say logical_expression ? 1 : 0 */
```

```
#define SIGNUM(i)    ((i) < 0 ? -1 : (i) > 0)

/* The corresponding uppercase letter if c is lowercase.  See K&R p. 249;
    King pp. 528−529. */
#define toupper(c)   (’a’ <= (c) && (c) <= ’z’ ? (c) - 32 : (c))
```

It’s better to write **’a’-’A’** instead of **32**, and **’A’-’a’** instead of **-32**:

```
#include <stdio.h>
#include <stdlib.h>

#define toupper(c)   (’a’ <= (c) && (c) <= ’z’ ? (c) + ’A’ - ’a’ : (c))

int main()
{
    int c;

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

    return EXIT_SUCCESS;
}
```

**▼ Homework 6.1: macros containing ?:**

Hand in only the **#define** lines, not an entire C program. You get no credit if you forget to parenthesize each argument in the replacement text, or if you forget to parenthesize the entire replacement text.

(1) Write the **#define** for a macro called **MIN** with two arguments. The value of the macro will be the value of the smaller of the two arguments.

(2) Write the **#define** for a macro called **tolower** with one argument. If the argument is the ASCII code of an uppercase letter, the value of the macro will be the ASCII code of the corresponding lowercase letter. Otherwise the value of the macro will be the same as the value of the argument. You get no credit if the number **32** appears in your **#define**.

(3) Write the **#define** for a macro called **INTBUFF** with no arguments. If **sizeof int** is less than or equal to 2, the value of the macro will be 7. Otherwise, if **sizeof int** is 4, the value of the macro will be 12. Otherwise, the value of the macro will be 21. Thus an array of **INTBUFF** characters would be big enough to hold the longest strings representing **int** values (**"-32768"**, **"-2147483648"**, and **"-9223372036854775808"**), including the terminating **’\0’**.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define  INTBUFF    Your homework is to complete this #define.

int main()
{
    char buffer[INTBUFF];

    srand(time(NULL));

    /* Deposit characters in memory with trailing ’\0’. */
    sprintf(buffer, "%d", rand());

    printf("%s\n", buffer);
16      return EXIT_SUCCESS;
17  }
```

▲

**Initialize a pointer**

One way to initialize a pointer is to let it be the address of another variable:

```
 1      char c = 'A';
 2      char *p = &c;       /* Put the address of the variable c into p. */
```

You can split line 2 into

```
        char *p;            /* Create a variable p containing garbage. */
        p = &c;             /* Replace the garbage by the address of c. */
```

—but why would you want to?

Another way is shown below. It looks like line 6 puts a string into **p**, but in reality it merely put the address of the first character of the string into **p**. Line 6 loads six **char**'s (including the terminating **'\0'**) into consecutive memory locations, and then puts the address of the letter ‘‘h’’ into **p**. The six bytes of memory are not necessarily located anywhere near **p**.

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  int main()
 5  {
 6      char *p = "hello";
 7      int i;
 8      char *q;
 9
10      printf("%p\n", p);
11      printf("%s\n", p);
12
13      /* Print each character individually. */
14      for (i = 0; p[i] != '\0'; ++i) {
15          printf("%c", p[i]);                    /* or putchar(p[i]); */
16      }
17      printf("\n");                              /* or putchar('\n'); */
18
19      for (q = p; *q != '\0'; ++q) {
20          printf("%c", *q);                      /* or putchar(*q); */
21      }
22      printf("\n");                              /* or putchar('\n'); */
23
24      return EXIT_SUCCESS;
25  }
```

```
1000         line 10
hello        line 11
hello        lines 14−17
hello        lines 19−22
```

You can split line 6 into

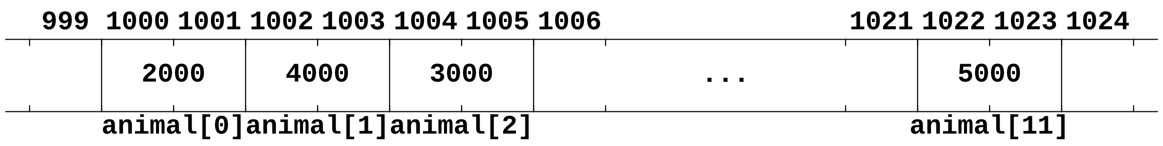
```
        char *p;            /* Ceate a variable p containing garbage. */
        p = "hello";        /* Replace the garbage by the address of the "h". */
```

—but why would you want to?

**An array of pointers: K&R pp. 107−110, 113; King pp. 262−263**

It looks like each element of the following array holds a word, but in reality each one merely holds the address of the first character of the word. See the example on K&R p. 104; King pp. 262−263 and the initialized array on K&R pp. 113−114; King p. 262.

The following diagram of the **`animal[]`** array assumes that the address of the array is 1000 and that **`sizeof(char *)`** is 2, i.e., that each element of the array occupies two bytes.



The strings are not necessarily adjacent to the **`animal`** array or to each other, and are not necessarily stored in the order in which they initialize the array. See the diagrams on K&R pp. 107, 114; King p. 263.

| 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 |
|---|---|---|---|---|---|---|
| ’m’ | ’o’ | ’n’ | ’k’ | ’e’ | ’y’ | ’\0’ |

| 4000 | 4001 | 4002 | 4003 | 4004 | 4005 | 4006 | 4007 |
|---|---|---|---|---|---|---|---|
| ’r’ | ’o’ | ’o’ | ’s’ | ’t’ | ’e’ | ’r’ | ’\0’ |

| 3000 | 3001 | 3002 | 3003 |
|---|---|---|---|
| ’d’ | ’o’ | ’g’ | ’\0’ |

| 5000 | 5001 | 5002 | 5003 | 5004 | 5005 |
|---|---|---|---|---|---|
| ’s’ | ’h’ | ’e’ | ’e’ | ’p’ | ’\0’ |

By storing the strings outside the body of the array, no space is wasted if the strings are of unequal length. A string can be lengthened without running into the next string. And two members of the array can be swapped by merely moving two pointers.

—On the Web at
**`http://i5.nyu.edu/~mm64/x52.9232/src/monkey.c`**

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *animal[] = {
        "monkey",    /*  0 */
        "rooster",   /*  1 */
        "dog",       /*  2 */
        "pig",       /*  3 */
        "rat",       /*  4 */
        "ox",       /*  5 */
        "tiger",    /*  6 */
        "hare",     /*  7 */
        "dragon",   /*  8 */
        "snake",    /*  9 */
        "horse",    /* 10 */
        "sheep"     /* 11 */
    };
#define N   (sizeof animal / sizeof animal[0])

    int year;

    printf("Please type a year and press RETURN: ");
    scanf("%d", &year);

    printf("%d is the year of the %s.\n", year, animal[year % N]);
    return EXIT_SUCCESS;
}
```

```
Please type a year and press RETURN: 2006
2006 is the year of the dog.
```

**How not to program**

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int year;

    printf("Please type a year and press RETURN.\n");
    scanf("%d", &year);

    if (year % 12 == 0) {
        printf("%d is the year of the dragon.\n", year);
    } else if (year % 12 == 1) {
        printf("%d is the year of the rooster.\n", year);
    } else if (year % 12 == 2) {
        printf("%d is the year of the dog.\n", year);
    } else if   /* etc. */
```

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int year;

    printf("Please type a year and press RETURN.\n");
    scanf("%d", &year);
    printf("%d is the year of the ", year);

    if (year % 12 == 0) {
        printf("dragon.\n");
    } else if (year % 12 == 1) {
        printf("rooster.\n");
    } else if (year % 12 == 2) {
        printf("dog.\n");
    } else if   /* etc. */
```

### ▼ Homework 6.2: select the correct anniversary present

```
How many years have you been married? 10
Then this is your tin anniversary!
```

First make sure that the user's input is in the range 1 to 15 inclusive.

| | |
|---|---|
| 1 | *paper* |
| 2 | *cotton* |
| 3 | *leather* |
| 4 | *linnen* |
| 5 | *wood* |
| 6 | *iron* |
| 7 | *wool* |
| 8 | *bronze* |
| 9 | *pottery* |
| 10 | *tin* |
| 11 | *steel* |
| 12 | *silk* |
| 13 | *lace* |
| 14 | *ivory* |
| 15 | *crystal* |

▲

### A two-dimensional array of pointers

If you have provided an initial value for each element, you can remove the first dimension (the empty brackets in line 6) but not the other dimensions.

You can also remove the **{**curly braces**}** in lines 8−26, but not the ones in lines 6 and 27.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 int main()
 5 {
 6     char *anniversary[][2] = {
 7         /* traditional */     /* contemporary */
 8         {NULL,                NULL},      /* dummy so subscripts will start at 1 */
 9
10         {"paper",             "clocks"},              /*  1 */
11         {"cotton",            "china"},               /*  2 */
12         {"leather",           "crystal"},             /*  3 */
13         {"linnen",            "electrical"},          /*  4 */
14         {"wood",              "silverware"},          /*  5 */
15
16         {"iron",              "wood"},                /*  6 */
17         {"wool",            "desk set"},            /*  7 */
18         {"bronze",          "linnen"},              /*  8 */
19         {"pottery",         "leather"},             /*  9 */
20         {"tin",             "diamond jewelry"},     /* 10 */
21
22         {"steel",           "fashion jewelry"},     /* 11 */
23         {"silk",            "pearls"},              /* 12 */
24         {"lace",            "furs"},                /* 13 */
25         {"ivory",           "gold jewelry"},        /* 14 */
26         {"crystal",         "watches"}              /* 15 */
27     };
28
29 /* minus one because of the dummy element */
30 #define N (sizeof anniversary / sizeof anniversary[0] - 1)
31
32     int year;
33     int mode;
34
35     printf("How many years have you been married? ");
36     scanf("%d", &year);
37     if (year < 1 || year > N) {
38         printf("Sorry, year must be in range 1 to %d inclusive.\n", N);
39         return EXIT_FAILURE;
40     }
41
42     printf("Type 0 for traditional, 1 for contemporary: ");
43     scanf("%d", &mode);
44
45     printf("Then this is your %s anniversary!\n", anniversary[year][mode]);
46     return EXIT_SUCCESS;
47 }
```

**Loop through an array of pointers**

Line 23 below is our first example of a ‘‘pointer to a pointer’’. The expression **`p`** is of data type **`char **`**, i.e., ‘‘pointer to pointer to **`char`**’’. The expression **`*p`** is of data type **`char *`**, i.e., ‘‘pointer to **`char`**’’. And the expression **`**p`** is of data type **`char`**.

During the first iteration of the loop in lines 29−31, the value of **`p`** is 1000, the value of **`*p`** is 2000, and the value of **`**p`** would be 109 (the ASCII code of ‘‘m’’). During the second iteration, the value of **`p`** is 1002, the value of **`*p`** is 4000, and the value of **`**p`** would be 114 (the ASCII code of ‘‘r’’).

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 char *animal[] = {
 5     "monkey",    /*  0 */
 6     "rooster",   /*  1 */
 7     "dog",       /*  2 */
 8     "pig",       /*  3 */
 9     "rat",       /*  4 */
10     "ox",        /*  5 */
11     "tiger",     /*  6 */
12     "hare",      /*  7 */
13     "dragon",    /*  8 */
14     "snake",     /*  9 */
```

```
    "horse",     /* 10 */
    "sheep"      /* 11 */
};
#define N   (sizeof animal / sizeof animal[0])

int main()
{
    int i;
    char **p;

    for (i = 0; i < N; ++i) {
        printf("%s is at address %p.\n", animal[i], animal[i]);
        printf("It starts with the three characters %c, %c, and %c.\n\m",
            animal[i][0], animal[i][1], animal[i][2]);
    }

    for (p = animal; p < animal + N; ++p) {
        printf("%s is at address %p.\n", *p, *p);
        printf("It starts with the three characters %c, %c, and %c.\n\n",
            (*p)[0], (*p)[1], (*p)[2]);
    }

    return EXIT_SUCCESS;
}
```

```
monkey is at address 2000.
It starts with the three characters m, o, n.

rooster is at address 4000.
It starts with the three characters r, o, o.
```

**▼ Homework 6.3: loop through an array of strings**

Write a C program to print one line of instructions and let the user type his or her weight. Then print the following table, with the decimal points aligned and two digits to the right of each decimal point:

```
Please type your weight: 150
On each planet, your weight would be
Mercury       40.50
Venus        127.50
Earth        150.00      etc.
```

Your program must have exactly three **`printf`**’s, one **`scanf`**, one **`for`**, and no **`if`**’s or **`while`**’s. It must have exactly two initialized arrays, two pointers, and a variable named **`weight`**:

```
char *name[] = {
double factor[] = {

#define N (sizeof name / sizeof name[0])

char **pn;                     /* for looping through the name array */
double *pf;                    /* for looping through the factor array */

double weight;                 /* hold the number that the user inputs */
```

The names of the planets and the factors by which to multiply your earth weight are

|  |  |
|---:|---|
| `.27` | *Mercury* |
| `.85` | *Venus* |
| `1.00` | *Earth* |
| `.38` | *Mars* |
| `2.33` | *Jupiter* |
| `.92` | *Saturn* |
| `.85` | *Uranus* |
| `1.12` | *Neptune* |
| `.44` | *Pluto* |

Put no comment alongside each string in the **`name`** array. Put the name of the corresponding planet in a comment alongside each number in the **`factor`** array.

We will rewrite this program later with one array of structures instead of two parallel arrays. Handout 5, p. 3 contains another example of parallel arrays. If you're more interested in finance, change the data to a list of foreign currencies and exchange rates.

▲

## ▼ Homework 6.4: retrofit an array of strings into an existing program

Retrofit an array of strings into one of the following three programs. Adding an array to the first two will let you remove some of the control statements that are otherwise needed. Hand in only one program: you get no credit if you hand in more than one.

(1) Add an array of strings to the last version of the **`mislead()`** function in Handout 7 (the version that returns an **`int`**). Write additional messages, striving for utter authenticity: **`"bus error"`**, **`"floating point exception"`**, **`"fork failed"`**. Unix people can say **`errno`**(2) or see **`/usr/include/sys/errno.h`** for error message ideas.

Make a **`sizeof/sizeof`** macro to measure the number of elements in the array (as in Handout 5, p. 12) and write the macro as the right operand of the **`%`** instead of writing **`3`**.

The array will let you eliminate the **`if`**'s. The finished function must have two **`printf`**'s and no **`if`**'s; no credit otherwise. Write a **`main()`** function to try it out.

(2) Add two arrays of strings to the answer to Homework 2.1 in Handout 3, pp. 12−13. Name the first array **`suffix`** and give it exactly five elements: an initial dummy followed by **`"st"`**, **`"nd"`**, **`"rd"`**, **`"th"`**. Use it to eliminate the **`switch`** in lines 12−28.

The fourth and fifth days will both use the suffix **`th`**. Instead of implementing this with an **`if`**, use the **`MIN`** macro as the subscript of the array:

```
suffix[MIN(day, 4)]
```

Your **`MIN`** macro must take two arguments as shown above, just like the **`MAX`** macro; no credit otherwise. Name the second array **`gift`** and give it exactly six elements: an initial dummy followed by

```
"A partridge in a pear tree",
"Two turtledoves",
"Three French hens",
"Four colly birds",
"Five gold rings"
```

Use **`int`** variables instead of pointer variables to loop through the arrays. Decide whether the elements should be in ascending order (as shown above) or descending order. Use this array to change the **`switch`** in lines 32−57 into a **`for`**.

Use **`NULL`** instead of **`0`** as the dummy element in an array of strings. **`NULL`** is a **`#define`** in **`stdio.h`**; write **`NULL`** in all uppercase.

The finished program must have exactly two **for**'s, two variables (both **int**'s; name the new one **d**), no **if**'s, no **switch**'s, and a small number of **printf**'s. The words ‘‘partridge’’, ‘‘turtledoves’’, ‘‘hens’’, etc., must each appear exactly once in the program. Forget about the ‘‘A/And a’’ issue. Make a **sizeof/sizeof** macro to measure the number of elements in the **gift** array.

(3) Add an array of strings giving the name of each month to the to non-extra credit answer to Homework 2.5 in Handout 4, pp. 15−16.

Let the user type in a month name instead of a month number:

```
1$ due
Please type the starting month and press RETURN.
December
Please type the starting day and press RETURN.
21
Please type the starting year and press RETURN.
2006
```

Add a

```
#define BUFFSIZE 256     /* number of bytes in input buffer */
char birthmonth[BUFFSIZE];
```

to receive the word that the user types in. **#include <string.h>** and write a **for** loop with a **strcmp** inside it. Output an error message and exit with **EXIT_FAILURE** if the user does not input a valid month name.

Instead of printing the due date as three numbers, the program will now print it as

```
December 21, 2006
```

Your program will have two arrays with the same number of elements. Call the new array **name**. If you want to add a dummy element at the start of **name**, initialize the dummy to **NULL** instead of to **0**.

Use **int** variables instead of pointer variables to loop through the arrays. Do not write 12 **if** statements. Ignore leap years: you will get no credit if your program handles leap years. The names and sizes of the months must appear nowhere in your program except in the two arrays. We will rewrite this program later with one array of structures instead of two parallel arrays.
□