CS251 Programming Languages **Handout # 29**
Prof. Lyn Turbak **April 21, 2004**
Wellesley College

C Examples

1 Multiplication Table

```
    // multable.c
    // Print an nxn multiplication table
    int main () {
      int n = 10;
      int i, j;
      // Print row labels
      for (i=1; i<=n; i++) {
        for (j=1; j<=n; j++) {
          printf("\t%d",i*j);
        }
        printf("\n");
      }
    }
```

```
[fturbak@jaguar c] gcc -o multable multable.c
[fturbak@jaguar c] ./multable
        1       2       3       4       5       6       7       8       9       10
        2       4       6       8       10      12      14      16      18      20
        3       6       9       12      15      18      21      24      27      30
        4       8       12      16      20      24      28      32      36      40
        5       10      15      20      25      30      35      40      45      50
        6       12      18      24      30      36      42      48      54      60
        7       14      21      28      35      42      49      56      63      70
        8       16      24      32      40      48      56      64      72      80
        9       18      27      36      45      54      63      72      81      90
        10      20      30      40      50      60      70      80      90      100
```

2 Swapping via Integer Pointers

```
    // intptr.c
    // Illustrate integer pointers
    void printab (int x, int y) {
      printf("a=%d; b=%d\n", x, y);
    }

    void swap (int* x, int* y) {
      int temp;
      printf("x=%u; *x=%d; y=%u; *y=%d\n", x, *x, y, *y); // %u for unsigned int
      temp = *x;
      *x = *y;
      *y = temp;
      printf("x=%u; *x=%d; y=%u; *y=%d\n", x, *x, y, *y);
    }

    int main () {
      int a = 1;
      int b = 2;
      printab(a,b);
      swap(&a,&b);
      // Can also swap array slots: e.g. swap(&c[i], &d[j])
      printab(a,b);
    }
```

```
[fturbak@jaguar c] gcc -o intptr intptr.c
[fturbak@jaguar c] ./intptr
a=1; b=2
x=3221223268; *x=1; y=3221223264; *y=2
x=3221223268; *x=2; y=3221223264; *y=1
a=2; b=1
```

Here are some examples of `swap` in other languages:

- Pascal supports both call-by-value and call-by-reference parameters:

```
    program TestSwap;
      procedure swap (var x : int, var y : int);
        begin
          var temp:integer := x;
          x := y;
          y := temp;
        end;
      begin
        var a:integer := 1;
        var b:integer := 2;
        swap(a,b); {a now contains 2 and b contains 1}
        {Can also call swap on array slots: e.g. swap(c[i],d[j]).}
      end;
    end.
```

- C++ supports call-by-reference parameters:

```
void swap (int &x, int &y) {
  int temp = x;
  x = y;
  y = temp;
}

int main () {
  int a = 1;
  int b = 2;
  swap(a,b); // a now contains 2 and b contains 1
  // Can also swap array slots: e.g. swap(c[i], d[j])
}
```

3 Parameter Passing

Consider the following Pascal program:

```
program ParamTest (input,output);
  var a, b: integer;
  procedure p (x:integer, var y:integer);
    begin
      x = x + y;
      y = x * y;
    end;
  begin
    a := 3;
    b := 4;
    p(a,b);
    writeln('a=', a);  {a is still 3}
    writeln('b=', b);  {b is now 28}
  end;
end.
```

How can we encode this in C?

```
void p (int x, int* y) {
  x = x + *y;
  *y = x * *y;
}

int main () {
  int a = 3;
  int b = 4;
  p(a,&b);
  printf("a=%d\nb=%d\n", a, b);
}
```

```
[fturbak@jaguar c] gcc -o paramtest paramtest.c
[fturbak@jaguar c] paramtest
a=3
b=28
```

4 Reading Standard Input

```
    // plus.c
    // Read two numbers from user and add them
    int main () {
      int a; // storage for first input
      int b; // storage for second input
      printf("a=");
      scanf("%d", &a); // read integer into a
      printf("b=");
      scanf("%d", &b); // read integer into b
      printf("%d+%d=%d\n",a,b,a + b);
    }
```

```
[fturbak@jaguar c] gcc -o plus plus.c
[fturbak@jaguar c] ./plus
a=3
b=4
3+4=7
```

5 Command Line Arguments

```
    // mainargs.c
    // Illustrates command line arguments
    int main (int argc, char** argv) {
      int i;
      printf("%d\n", argc);
      for (i=0; i<argc; i++) {
        printf("%s\n", argv[i]);
      }
    }

[fturbak@jaguar c] gcc -o mainargs mainargs.c
[fturbak@jaguar c] ./mainargs foo bar baz
4
./mainargs
foo
bar
baz
```

```
    // plusargs.c
    // Adds up all the numbers in the command line arguments
    int main (int argc, char** argv) {
      int i;
      int sum = 0;
      for (i=1; i<argc; i++) {
        sum += atoi(argv[i]); // atoi converts integer to string
      }
      printf("sum=%d\n", sum);
    }

[fturbak@jaguar c] gcc -o plusargs plusargs.c
[fturbak@jaguar c] ./plusargs 3 42 17
sum=62
```

6 Reading From a File

```
// readlines.c
// reads and displays lines from a file
#include <stdio.h>

int main (int argc, char** argv) {
  int i = 0;
  int line = 1;
  char c;
  char buff [128];
  FILE* f = fopen(argv[1],"r"); // open file named in argv[1] for reading
  while ((c = fgetc(f)) != EOF) { // EOF is "end of file" marker
    if (i >= 128) {
      printf("buffer overflow!\n");
      exit(0); // abort program if buffer overflow;
               // there are security problems if this not done!
    } else if (c == '\n') {
      buff[i] = 0;
      printf("%d:\t%s\n", line, buff);
      i = 0;
      line++;
    } else {
      buff[i++] = c;
    }
  }
}
```

```
[fturbak@jaguar c] gcc -o readlines readlines.c
[fturbak@jaguar c] ./readlines tiny-sorted.txt
1:      aback
2:      babe
3:      cab
4:      dad
5:      each
6:      fable
7:      gab
8:      ha
9:      ibex
10:     jab
11:     kanji
12:     lab
13:     mace
14:     nab
15:     oaf
16:     pace
```

7 Dangling Pointers

```
// dangling-pointer.c
// Illustrates problems with dangling pointers
void printarray(char* s, int* a, int n) {
  int i;
  for (i = 0; i < n; i++) {
    printf("%s[%d] = %d\t", s, i, a[i]);
  }
  printf("\n");
}

int* elts (int c, int n) {
  int a[n]; // Stack allocated array
  // Heap allocated array:
  // int* a = (int *) malloc(n*sizeof(int));
  int i;
  for (i = 0; i < n; i++) {
    a[i] = c*i;
  }
  printarray("a",a,n);
  return a;
}

int main () {
  int* b;
  int* c;
  b = elts(1,5);
  printarray("b",b,5);
  c = elts(2,5);
  printarray("b",b,5);
  printarray("c",c,5);
}
```

```
[fturbak@jaguar c] gcc -o dangling-pointer dangling-pointer.c
dangling-pointer.c: In function ‘elts’:
dangling-pointer.c:17: warning: function returns address of local variable
[fturbak@jaguar c] ./dangling-pointer
a[0] = 0        a[1] = 1        a[2] = 2        a[3] = 3        a[4] = 4
b[0] = 0        b[1] = 134514120        b[2] = -1073744224        b[3] = 3        b[4] = 4
a[0] = 0        a[1] = 2        a[2] = 4        a[3] = 6        a[4] = 8
b[0] = 0        b[1] = 134514120        b[2] = -1073744224        b[3] = 6        b[4] = 8
c[0] = 1108531968        c[1] = 134514120        c[2] = -1073744224        c[3] = 6        c[4] = 8
```

We can fix the problem by heap allocated the array `a` within `elts` using `malloc`. After this change, we get:

```
[fturbak@jaguar c] gcc -o dangling-pointer dangling-pointer.c
[fturbak@jaguar c] ./dangling-pointer
a[0] = 0        a[1] = 1        a[2] = 2        a[3] = 3        a[4] = 4
b[0] = 0        b[1] = 1        b[2] = 2        b[3] = 3        b[4] = 4
a[0] = 0        a[1] = 2        a[2] = 4        a[3] = 6        a[4] = 8
b[0] = 0        b[1] = 1        b[2] = 2        b[3] = 3        b[4] = 4
c[0] = 0        c[1] = 2        c[2] = 4        c[3] = 6        c[4] = 8
[fturbak@jaguar c]
```

8 Points as Structures

```
    // points-struct.c
    typedef struct P {int x; int y;} point;

    point scaledCopy (int s, point p) {
      point q;
      q.x = s * p.x;
      q.y = s * p.y;
      return q;
    }

    void scale1 (int s, point p) {
      p.x = s * p.x;
      p.y = s * p.y;
    }

    void scale2 (int s, point* p) {
      (*p).x = s * (*p).x;
      (*p).y = s * (*p).y;
    }

    void printPoint (point p) {
      printf("x=%d;y=%d\n", p.x, p.y);
    }

    int main () {
      point a,b; a.x = 1; a.y = 2;
      b = scaledCopy(3,a);
      printPoint(a); printPoint(b);
      scale1(4,a); scale2(5,&b);
      printPoint(a); printPoint(b);
    }
```

```
[fturbak@jaguar c] gcc -o points-struct points-struct.c
[fturbak@jaguar c] points-struct
x=1;y=2
x=3;y=6
x=1;y=2
x=15;y=30
```

9 Points as Arrays

```
// points-array.c
/* Represent a point as a 2-slot integer array,
   with x in slot 0 and y in slot 1. */

typedef int point[2];

/* Not possible with arrays b/c attempts to return array.
  point scaledCopy (int s, point p) {
    point q;
    q[0] = s * p[0];
    q[1] = s * p[1];
    return q;
    }
*/

void scaledCopy (int s, point p, point q) {
    q[0] = s * p[0];
    q[1] = s * p[1];
}

void scale1 (int s, point p) {
  p[0] = s * p[0];
  p[1] = s * p[1];
}

void scale2 (int s, point* p) {
  (*p)[0] = s * (*p)[0];
  (*p)[1] = s * (*p)[1];
}

void printPoint (point p) {
  printf("x=%d;y=%d\n", p[0], p[1]);
}

int main () {
  point a,b; a[0] = 1; a[1] = 2;
  scaledCopy(3,a,b);
  printPoint(a); printPoint(b);
  scale1(4,a);
  scale2(5,&b);
  printPoint(a); printPoint(b);
}
```

```
[fturbak@jaguar c] gcc -o points-array points-array.c
[fturbak@jaguar c] points-array
x=1;y=2
x=3;y=6
x=4;y=8
x=15;y=30
```

10 Integer Lists

```
    // sumlist.c
    #include <stddef.h>

    typedef struct IL {int head; struct IL *tail;} intlist;

    int sumlist (intlist* lst) {
      if (lst == NULL)
        return 0;
      else
        return (*lst).head + sumlist((*lst).tail);
    }

    intlist* fromTo (int lo, int hi) {
      intlist* result;
      if (lo > hi)
        return NULL;
      else {
        result  = (intlist*) malloc(sizeof(intlist));
        (*result).head = lo;
        (*result).tail = fromTo(lo + 1, hi);
        return result;
      }
    }

    int main () {
      printf("sumlist(fromTo(1,10))=%d\n", sumlist(fromTo(1,10)));
    }
```

```
[fturbak@jaguar c] gcc -o sumlist sumlist.c
[fturbak@jaguar c] sumlist
sumlist(fromTo(1,10))=55
[fturbak@jaguar c]
```