BIRZEIT UNIVERSITY

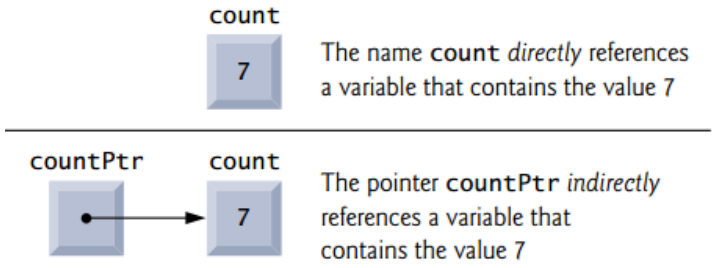
Pointers & Modular Programming

Computer Science Department

Pointers

- **Pointers : variables whose values are memory addresses.**
- A pointer contains an address of a variable that contains a specific value

- **<u>Directly and indirectly referencing a variable</u>:**



Pointer declaration

float *p;

Identifies p as a pointer variable of type "pointer to float ."

This means that we can store the memory address of a type float variable in p .

- **Pointer Type Declaration:**

SYNTAX: ***type * variable ;***
EXAMPLE: **float *p;**

The value of the pointer variable p is a memory address

Pointer Operators

1. The Address (&) Operator

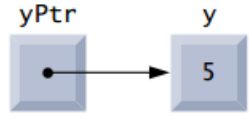
The &, or address operator, is a unary operator that returns the address of its operand. Assuming the definitions:

```
int y = 5;
int *yPtr;
```

the statement

```
yPtr = &y;
```

assigns the address of the variable y to pointer variable yPtr. Variable yPtrs then said to "point to" y.



Pointer Operators – Cont.

2. The Indirection (*) Operator

The unary * operator, commonly referred to as the indirection operator or dereferencing operator, returns the value of the object to which its operand (i.e., a pointer) points. For example, the statement:

```
printf("%d", *yPtr);
```

prints the value of variable y (5).

Using * in this manner is called **dereferencing a pointer**

Demo

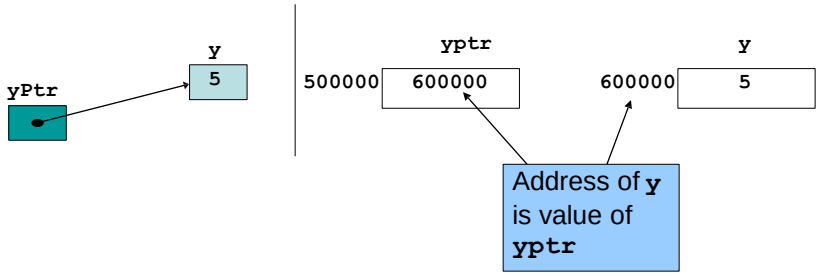
```
#include <stdio.h>

int main(void)
{
   int a = 7;
   int *aPtr; ; // set aPtr to the address of a
   aPtr = &a;

   printf("The address of a is %p"
          "\nThe value of aPtr is %p", &a, aPtr);

   printf("\n\nThe value of a is %d"
          "\nThe value of *aPtr is %d", a, *aPtr);

   printf("\n\nShowing that * and & are complements of "
          "each other\n&*aPtr = %p"
          "\n*&aPtr = %p\n", &*aPtr, *&aPtr);
}
```

```
The address of a is 0028FEC0
The value of aPtr is 0028FEC0

The value of a is 7
The value of *aPtr is 7

Showing that * and & are complements of each other
&*aPtr = 0028FEC0
*&aPtr = 0028FEC0
```

Example (1)

```
int y = 5;
int *yPtr;
yPtr = &y;   //yPtr gets address of y
```

yPtr "points to" y



Example (2)

```
int i = 5;
int *ptr;                    /* declare a pointer variable */
ptr = &i;                    /* store address-of i to ptr */
printf("*ptr = %d\n", *ptr);   /* refer to referee of ptr */
```

output
*ptr = 5

Example (3):

What actually *ptr* is?

- **ptr** is a variable storing **an address**
- ptr is **NOT** storing the actual value of i

```
Address of i= 0022FF18    Address of ptr=0022FF1C
```

```
int i = 5;
int *ptr;
ptr = &i;
printf("i = %d\n", i);
printf("*ptr = %d\n", *ptr);
printf("ptr = %p\n", ptr);
printf("address of i = %p\n", &i);
printf("address of ptr = %p\n", &ptr);
```

ptr | address of i

i | 5

```
Output:
i = 5
*ptr = 5
ptr =0022FF18
address of i = 0022FF18
address of ptr = 0022FF1C
```

value of ptr = address of i in memory

Example (4)

```c
#include <stdio.h>
int main()
{
    int x, *p;
    p = &x;
    *p = 0;
    printf("x is %d\n", x);
    printf("*p is %d\n", *p);
    *p += 1;
    printf("x is %d\n", x);
    (*p)++;
    printf("x is %d\n", x);
    return 0;
}
```

```
Output:
x is 0
*p is 0
x is 1
x is 2
```

Example (5)

Trace the execution of the following fragment

```
int m = 10, n = 5;
int *mp, *np;

mp = &m;
np = &n;
*mp = *mp + *np;
*np = *mp – *np;
printf("%d %d\n%d %d\n", m, *mp, n, *np);
```

Output:

```
15 15
10 10
```

Passing Arguments to Functions

- All arguments in C are passed by **value**.
- Functions often require the capability to modify variables in the caller
- So, *receive a pointer* to a large data object to avoid the overhead of receiving the object by value

- **return** may be used to return one value

- Pointers can be used to enable a function to "return" multiple values to its caller by modifying variables in the caller

- There are two ways to pass arguments to a function—pass-by-value and pass-by-reference.

Pass-By-Value

```c
#include <stdio.h>

int cubeByValue(int n); // prototype

int main(void)
{
   int number = 5; // initialize number

   printf("The original value of number is %d", number);

   // pass number by value to cubeByValue
   number = cubeByValue(number);

   printf("\nThe new value of number is %d\n", number);
}

// calculate and return cube of integer argument
int cubeByValue(int n)
{
   return n * n * n; // cube local variable n and return result
}
```

```
The original value of number is 5
The new value of number is 125
```

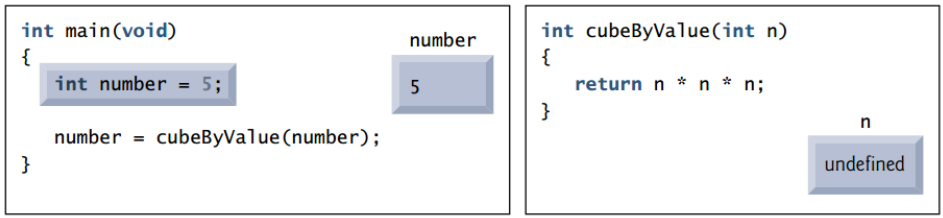
Pass-By-Reference

```c
#include <stdio.h>

void cubeByReference(int *nPtr); // function prototype

int main(void)
{
   int number = 5; // initialize number

   printf("The original value of number is %d", number);

   // pass address of number to cubeByReference
   cubeByReference(&number);

   printf("\nThe new value of number is %d\n", number);
}

// calculate cube of *nPtr; actually modifies number in main
void cubeByReference(int *nPtr)
{
   *nPtr = *nPtr * *nPtr * *nPtr; // cube *nPtr
}
```

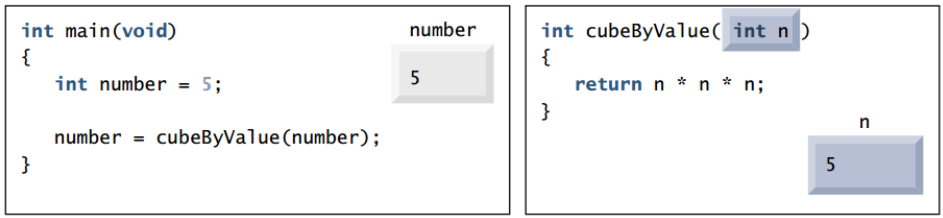
```
The original value of number is 5
The new value of number is 125
```

Call-by-Value - Example

Step 1: Before `main` calls `cubeByValue`:

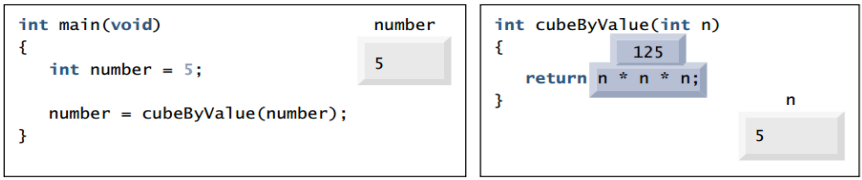


Step 2: After `cubeByValue` receives the call:

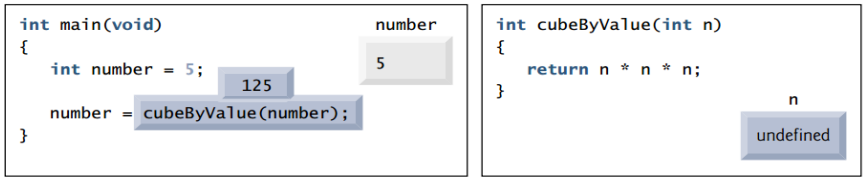


Call-by-Value – Example – cont.

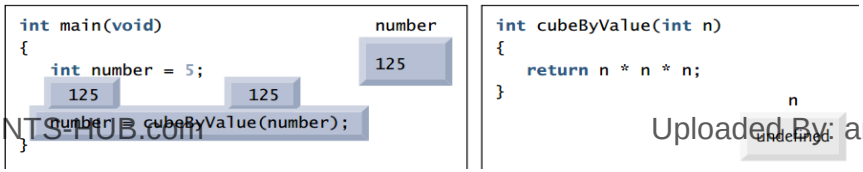
Step 3: After `cubeByValue` cubes parameter `n` and before `cubeByValue` returns to `main`:



Step 4: After `cubeByValue` returns to `main` and before assigning the result to `number`:

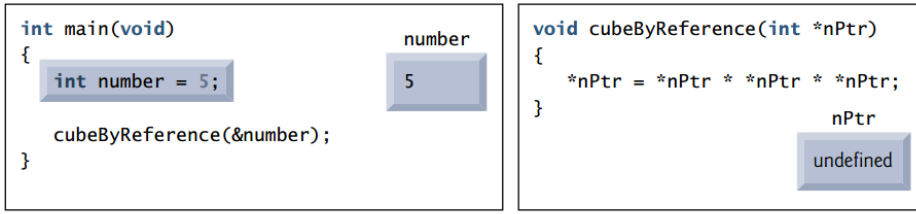


Step 5: After `main` completes the assignment to `number`:

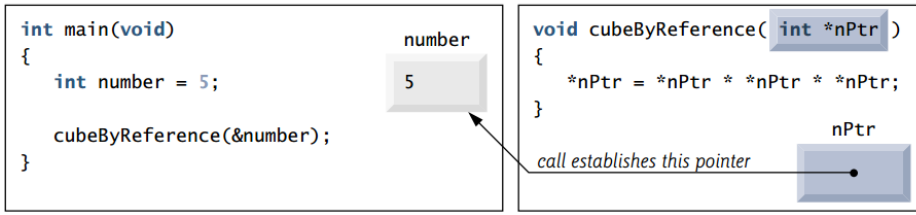


Call-by-Reference - Example

Step 1: Before `main` calls `cubeByReference`:

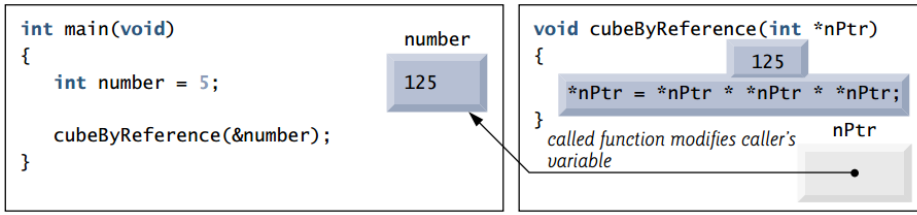


Step 2: After `cubeByReference` receives the call and before `*nPtr` is cubed:



Call-by-Reference – Cont.

Step 3: After `*nPtr` is cubed and before program control returns to `main`:



Function - Redefined

```c
#include <stdio.h>
int sum(int,int);
int main()
{
    int num1=4,num2=5;
    int result;
    result=sum(num1,num2);
    printf("The result is %d",result);

    return 0;
}
int sum(int x,int y)
{
    return (x+y);
}
```

```c
#include <stdio.h>
void sum(int*,int,int);
int main()
{
    int num1=4,num2=5;
    int result;
    sum(&result,num1,num2);
    printf("The result is %d",result);

    return 0;
}
void sum(int*res,int x,int y)
{
    *res=x+y;
}
```

Functions with Output Parameters

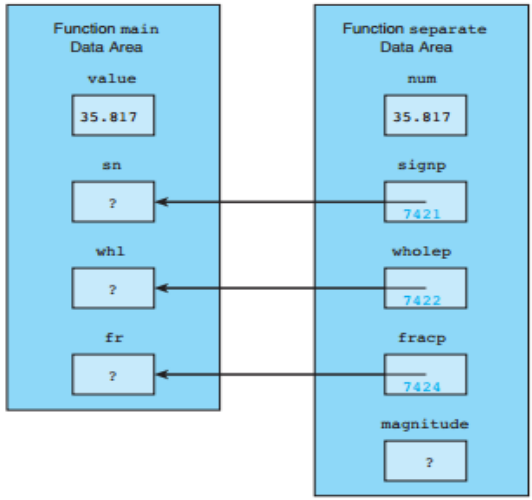
Write function to find the sum and the difference between two numbers.

```c
#include <stdio.h>
int sum_difference (int,int,int*);
int main()
{
    int num1,num2,sum,diff;
    printf("Please enter two numbers: ");
    scanf("%d%d",&num1,&num2);
    diff=sum_difference (num1,num2,&sum);
    printf("Sum= %d and difference=%d",sum,diff);
    return 0;
}

int sum_difference (int x,int y,int* sum)
{
    *sum=x+y;
    return (x-y);
}
```

Example

Write a function to separate a number into three parts:

- a sign (+, -, or blank),
- a whole number magnitude
- a fractional part.



Parameter Correspondence for separate(value, &sn, &whl, &fr);

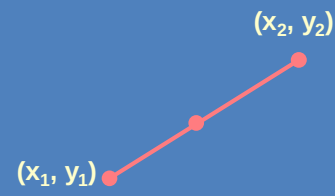
Function: separate

```
void
separate(double  num,    /* input - value to be split                    */
         char   *signp,  /* output - sign of num                         */
         int    *wholep, /* output - whole number magnitude of num       */
         double *fracp)  /* output - fractional part of num              */
{
      double magnitude; /* local variable - magnitude of num            */

      /* Determines sign of num */
      if (num < 0)
           *signp = '-';
      else if (num == 0)
           *signp = ' ';
      else
           *signp = '+';

      /* Finds magnitude of num (its absolute value) and
         separates it into whole and fractional parts                    */
      magnitude = fabs(num);
      *wholep = floor(magnitude);
      *fracp = magnitude - *wholep;
```

```
#include <stdio.h>
#include <math.h>
void separate(double num, char *signp, int *wholep, double *fracp);

int main(void)
{
    double value; /* input - number to analyze                    */
    char sn;      /* output - sign of value                       */
    int whl;      /* output - whole number magnitude of value     */
    double fr;    /* output - fractional part of value            */

    /* Gets data                                                   */
    printf("Enter a value to analyze> ");
    scanf("%lf", &value);

    /* Separates data value into three parts                       */
    separate(value, &sn, &whl, &fr);

    /* Prints results                                              */
    printf("Parts of %.4f\n sign: %c\n", value, sn);
    printf(" whole number magnitude: %d\n", whl);
    printf(" fractional part: %.4f\n", fr);

    return (0);
}
```

M-24

Example: Midpoint Of A Line



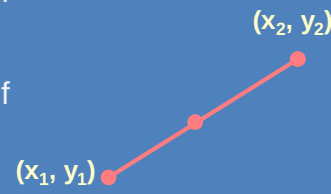
M-25

Example: Midpoint Of A Line

Problem: Find the midpoint of a line segment.

Algorithm: find the average of the coordinates of the endpoints:

```
xmid = (x1+x2)/2.0;
ymid = (y1+y2)/2.0;
```



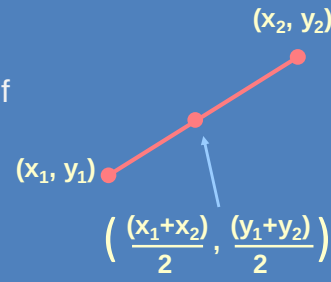
M-26

Example: Midpoint Of A Line

Problem: Find the midpoint of a line segment.

Algorithm: find the average of the coordinates of the endpoints:

```
xmid = (x1+x2)/2.0;
ymid = (y1+y2)/2.0;
```



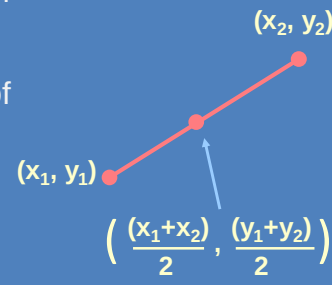
M-27

Example: Midpoint Of A Line

Problem: Find the midpoint of a line segment.

Algorithm: find the average of the coordinates of the endpoints:

```
xmid = (x1+x2)/2.0;
ymid = (y1+y2)/2.0;
```



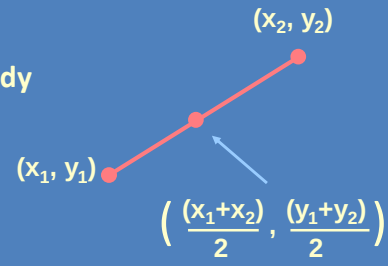
Programming approach: We'd like to package this in a function

M-28

Function Specification

Function specification: given endpoints (x1,y1) and (x2,y2) of a line segment, store the coordinates of the midpoint in (midx, midy)

Parameters:
x1, y1, x2, y2, midx, and midy

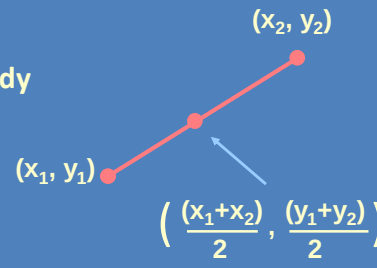


Function Specification

Function specification: given endpoints (x1,y1) and (x2,y2) of a line segment, store the coordinates of the midpoint in (midx, midy)

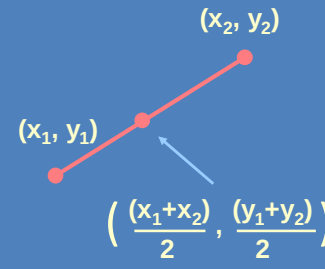
Parameters:
x1, y1, x2, y2, midx, and midy

The (midx,midy) parameters are being altered, so they need to be pointers



Midpoint Function: Code

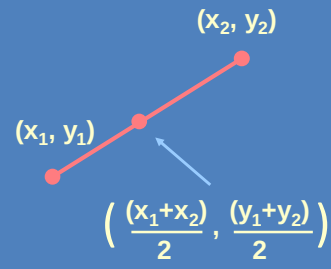
```
void set_midpoint( double x1, double y1,
                   double x2, double y2,
                   double * midx_p, double * midy_p ) {

    *midx_p = (x1 + x2) / 2.0;
    *midy_p = (y1 + y2) / 2.0;
}
```



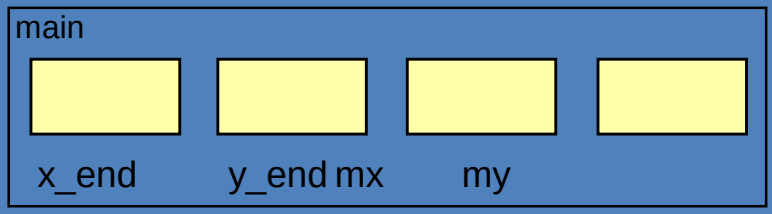
Midpoint Function: Code

```
void set_midpoint( double x1, double y1,
                   double x2, double y2,
                   double * midx_p, double * midy_p ) {

    *midx_p = (x1 + x2) / 2.0;
    *midy_p = (y1 + y2) / 2.0;
}
```

```
double x_end, y_end, mx, my;
x_end = 250.0; y_end = 100.0;
set_midpoint(0.0, 0.0,
             x_end, y_end,
             &mx, &my);
```

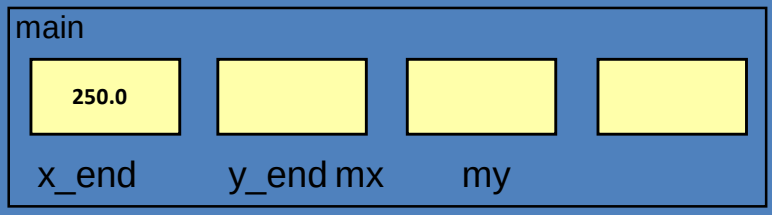


Trace



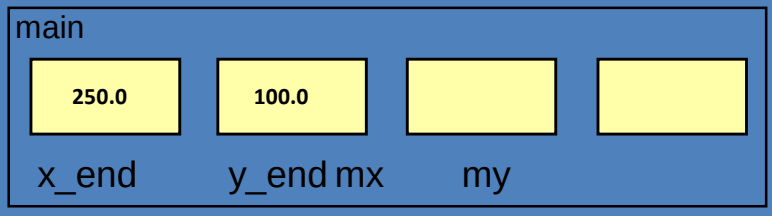
M-33

Trace



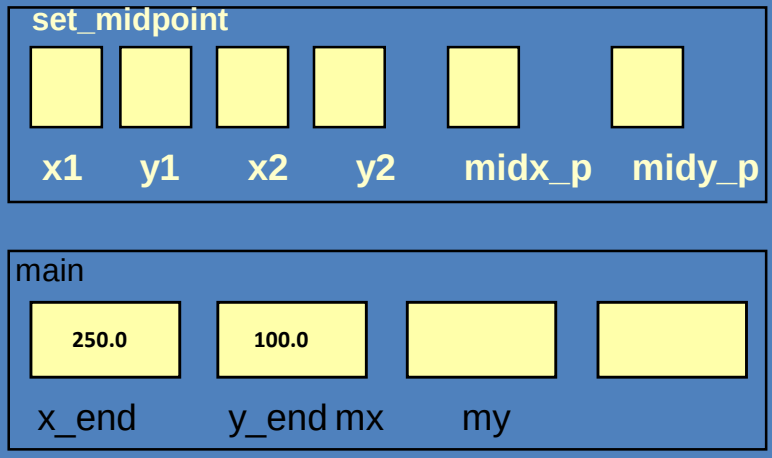
M-34

Trace



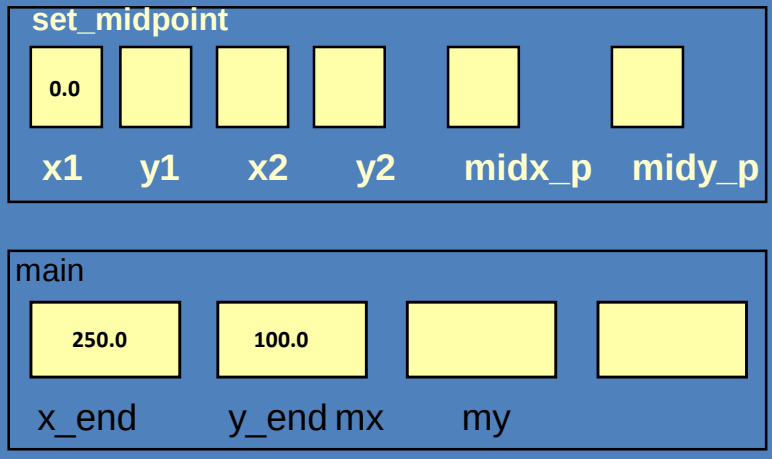
M-35

Trace



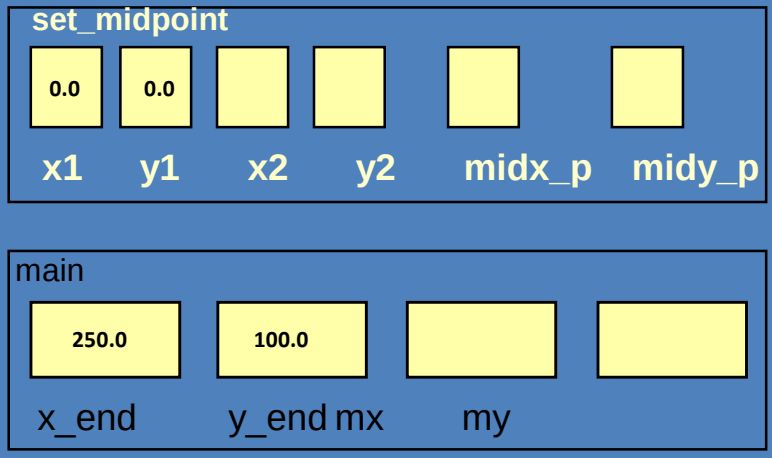
M-36

Trace



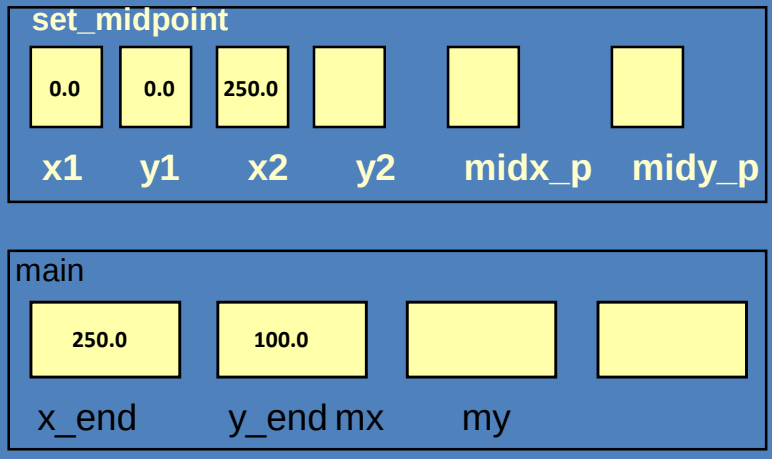
M-37

Trace



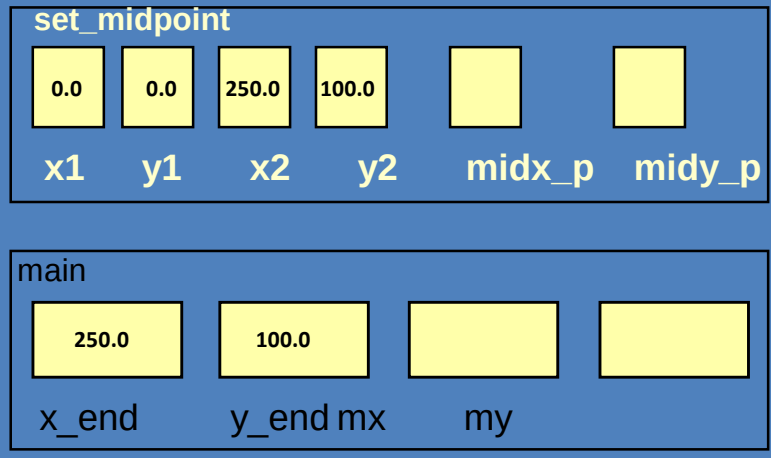
M-38

Trace



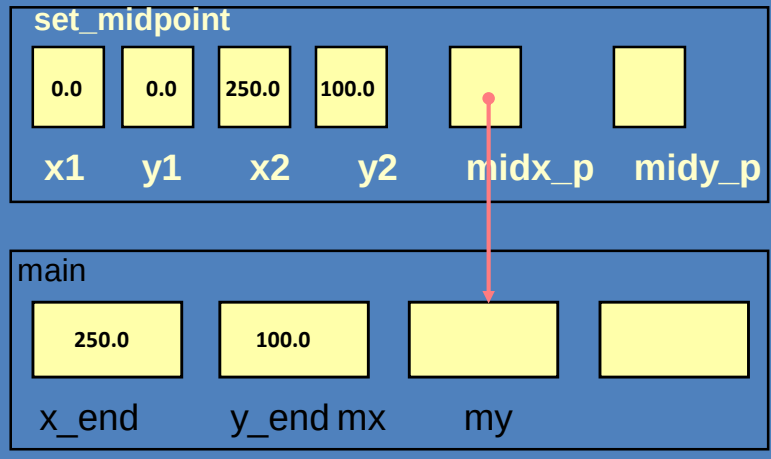
M-39

Trace



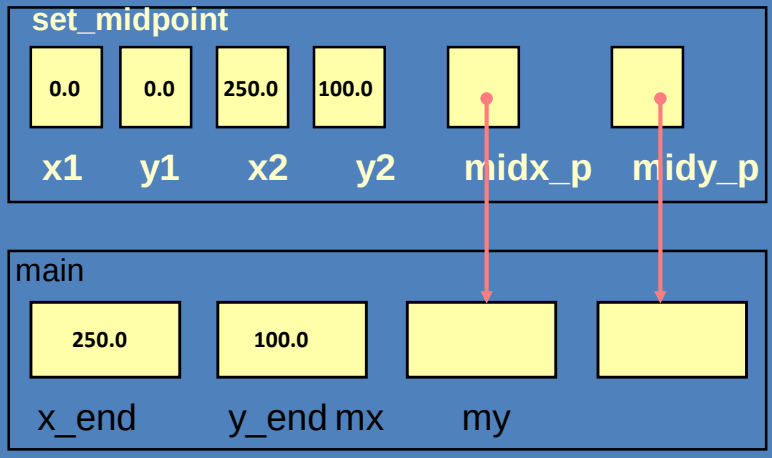
M-40

Trace



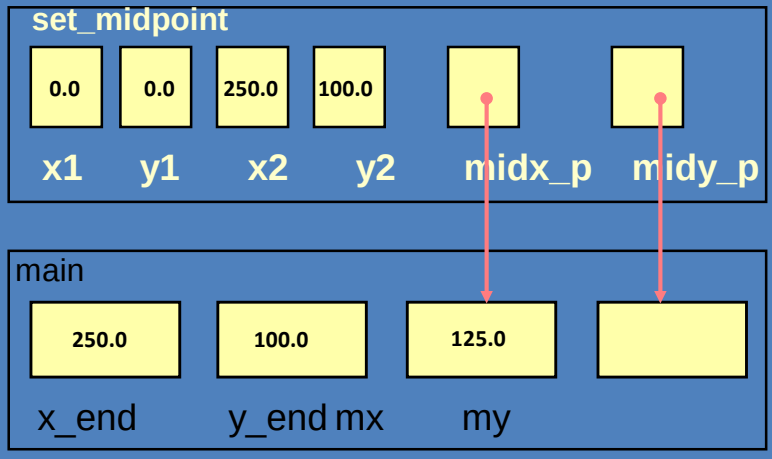
M-41

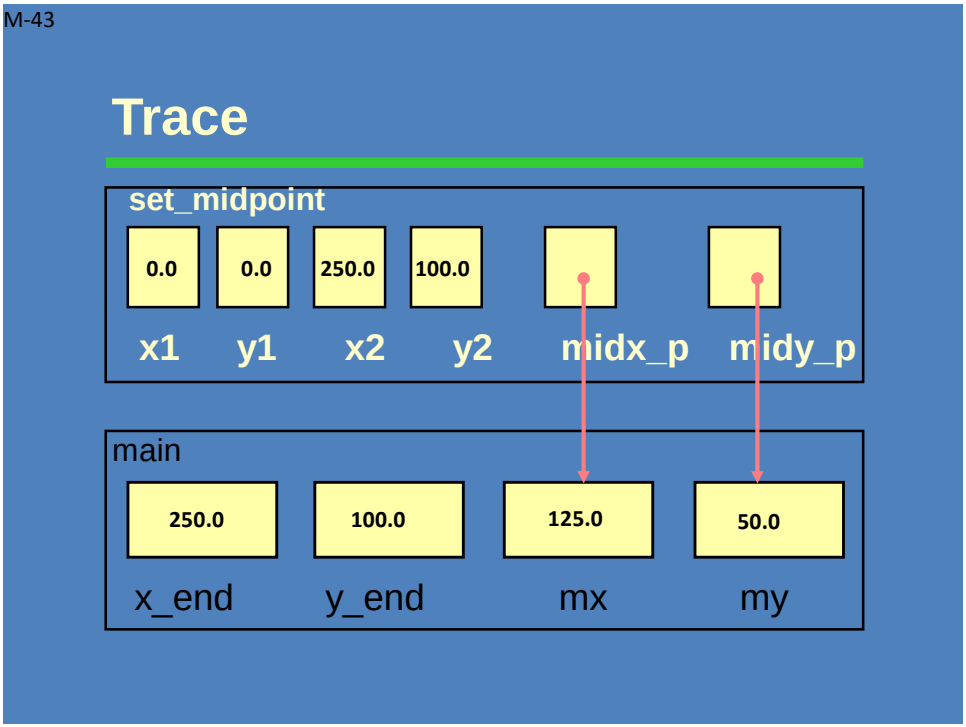
Trace



M-42

Trace





Multiple Calls to a Function with Input/output Parameters

```
#include <stdio.h>

void order(double *smp, double *lgp);

int
main(void)
{
        double num1, num2, num3; /* three numbers to put in order        */

        /* Gets test data                                                 */
        printf("Enter three numbers separated by blanks> ");
        scanf("%lf%lf%lf", &num1, &num2, &num3);

        /* Orders the three numbers                                       */
        order(&num1, &num2);
        order(&num1, &num3);
        order(&num2, &num3);

        /* Displays results                                               */
        printf("The numbers in ascending order are: %.2f %.2f %.2f\n",
               num1, num2, num3);

        return (0);
}
```

Multiple Calls to a Function with Input/output Parameters - Sorting

```
/*
 * Arranges arguments in ascending order.
 * Pre:   smp and lgp are addresses of defined type double variables
 * Post:  variable pointed to by smp contains the smaller of the type
 *        double values; variable pointed to by lgp contains the larger
 */

void
order(double *smp, double *lgp)    /* input/output */
{
       double temp; /* temporary variable to hold one number during swap    */

        if (*smp > *lgp) {
                temp = *smp;

                *smp = *lgp;
                *lgp = temp;
        }
}
```

```
Enter three numbers separated by blanks> 7.5 9.6 5.5
The numbers in ascending order are: 5.50 7.50 9.60
```

Trace of Program to Sort Three Numbers

Statement	num1	num2	num3	Effect
`scanf("...", &num1, &num2, &num3);`	7.5	9.6	5.5	Enters data
`order(&num1, &num2);`				No change
`order(&num1, &num3);`	5.5	9.6	7.5	Switches num1 and num3
`order(&num2, &num3);`	5.5	7.5	9.6	Switches num2 and num3
`printf("...", num1, num2, num3);`				Displays 5.5 7.5 9.6

Different Kinds of Function Subprograms

Purpose	Function Type	Parameters	To Return Result
To compute or obtain as input a single numeric or character value.	Same as type of value to be computed or obtained.	Input parameters hold copies of data provided by calling function.	Function code includes a `return` statement with an expression whose value is the result.
To produce printed output containing values of numeric or character arguments.	`void`	Input parameters hold copies of data provided by calling function.	No result is returned.
To compute multiple numeric or character results.	`void`	Input parameters hold copies of data provided by calling function. Output parameters are pointers to actual arguments.	Results are stored in the calling function's data area by indirect assignment through output parameters. No `return` statement is required.
To modify argument values.	`void`	Input/output parameters are pointers to actual arguments. Input data is accessed by indirect reference through parameters.	Results are stored in the calling function's data area by indirect assignment through output parameters. No `return` statement is required.

Scope of Names

```
#define MAX 950
#define LIMIT 200

void one(int anarg, double second);     /* prototype 1 */

int fun_two(int one, char anarg);       /* prototype 2 */

int
main(void)
{
        int localvar;

        . . .
} /* end main */


void
one(int anarg, double second)           /* header 1    */
{
        int onelocal;                   /* local 1     */
        . . .
} /* end one */


int
fun_two(int one, char anarg)            /* header 2    */
{
        int localvar;                   /* local 2     */
        . . .
} /* end fun_two */
```

Name	Visible in one	Visible in fun_two	Visible in main
MAX	yes	yes	yes
LIMIT	yes	yes	yes
main	yes	yes	yes
localvar (in main)	no	no	yes
one (the function)	yes	no	yes
anarg (int)	yes	no	no
second	yes	no	no
onelocal	yes	no	no
fun_two	yes	yes	yes
one (formal parameter)	no	yes	no
anarg (char)	no	yes	no
localvar (in fun_two)	no	yes	no

Example (7)

Write a function to :

1. **Find the number of digits in a given number**
2. **Sum of digits**
3. **Reverse a number**

Example:
Please enter a number: 123
number of digits = 3
sum of digits=6
reverse=321

Code

```
#include <stdio.h>
int sum_reverse (int,int*,int*);
int main()
{
    int num,numberOfDigits;
    int sumOfDigits,reverseDigidts;

    printf("Please enter  a number: ");
    scanf("%d",&num);
    numberOfDigits=
sum_reverse(num,&sumOfDigits,&reverseDigidts);

  printf(" Number of digits= %d\n Sum= %d\n
reverse= %d", numberOfDigits, sumOfDigits,
reverseDigidts);

return 0;
}
```

```
int sum_reverse (int
num,int *sum,int *rev )
{
    *sum=0;
    *rev=0;
    int counter=0;
    while (num>0)
    {
        ++counter;
        *sum+=num%10;
        *rev=*rev*10;
        *rev=*rev+num%10;
         num=num/10;
    }
    return counter;
}
```

Example (11)

C program to find square and cube of given number

```c
#include <stdio.h>
int square_cube(int,int*);
int main()
{
    int num,square,cube;
    printf("Please enter a number : ");
    scanf("%d",&num);
    square=square_cube(num,&cube);
    printf("square=%d\ncube=%d",square,cube);
    return 0;
}
int square_cube(int num,int*cube)
{
   int square;
   square=num*num;
   *cube=num*num*num;
   return square;
}
```

```
Please enter a number : 2
square=4
cube=8
```

Example (8)

Exchanges the values of the two integer variables

```c
#include<stdio.h>
void interchange(int*,int*);
int main() {

    int num1,num2;
    printf("Enter num1 and num2: ");
    scanf("%d%d",&num1,&num2);
    interchange(&num1,&num2);

    printf("\nNumber 1 : %d",num1);
    printf("\nNumber 2 : %d",num2);

    return(0);
}
void interchange(int *num1,int *num2)
{
    int temp;
    temp  = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```

Example (9)

Identify and correct the errors in the following code fragment, given the correct output (%p is used to print a pointer):

```
int y = 3;
int *yptr;
yptr = &y;
printf("The value of y is %d\n", *yptr);
printf("The address of y is %p\n",*yptr);
```

Change "*yptr" in the above statement to "yptr" or "&y"

Output:

The value of y is 3

The address of y is 2063865468

Example (10): Output

```
#include <stdio.h>
int main()
{
    int i = 0, j = 5;
    int *y;
    y=&j;
    for( i = 0; i <= 4; i++ )
    {
        *y  = *y + i;
    }
    printf( "The final value of j is %d.\n", j );
    return 0;
}
```

The final value of j is 15.