Computer Programming

Simple Data Types

Overview

- Standard data types
  - char, int, double, etc.
  - logical values
- Define new data types
  - enumerated types
- Passing functions as a parameter to subprogram

Representation of Numeric Types

- Why more than one numeric type?
  - integers
    - faster
    - less space
    - precise
  - double
    - larger interval (large and small values)

type `int` format

binary number

type `double` format

mantissa	exponent

Print Ranges for Positive Numeric Data

```c
1.  /*
2.   * Find implementation's ranges for positive numeric data
3.   */
4.
5.  #include <stdio.h>
6.  #include <limits.h> /* definition of INT_MAX                    */
7.  #include <float.h>  /* definitions of DBL_MIN, DBL_MAX           */
8.
9.  int
10. main(void)
11. {
12.       printf("Range of positive values of type int: 1 . . %d\n",
13.              INT_MAX);
14.       printf("Range of positive values of type double: %e . . %e\n",
15.              DBL_MIN, DBL_MAX);
16.
17.       return (0);
18. }
```

Numerical Inaccuracies

Errors in representing real numbers using double

- representational error
  - round-off error
  - magnified through repeated calculation
  - use as a loop control
- cancelation error
  - manipulating very small and very large real numbers
- arithmetic underflow
  - too small to represent
- arithmetic overflow
  - too large to represent

Type Conversion

- Automatic conversion
  - arithmetic operations
  - assignment
  - parameter passing
- Explicit conversion
  - casting
    - frac = n1 / d1;
    - frac = (double) (n1 / d1);
    - frac = (double) n1 / d1;

Representation and Conversion of char

- **ASCII**
  - numeric code (32 to 126 printable and others control char)
- **constant:**
  - 'a'
- **variable:**
  - char letter;
- **assignment:**
  - letter = 'A';
- **Comparison: == , != , < , > , <= , >=**
  - if (letter > 'A')
- **Relation with integer**
  - compare
  - convert

```
1.   /*
2.    * Prints part of the collating sequence
3.    */
4.
5.   #include <stdio.h>
6.
7.   #define START_CHAR ' '
8.   #define END_CHAR    'Z'
9.
10.  int
11.  main(void)
12.  {
13.          int char_code; /* numeric code of each character printed */
14.
15.          for  (char_code = (int)START_CHAR;
16.                char_code <= (int)END_CHAR;
17.                char_code = char_code + 1)
18.              printf("%c", (char)char_code);
19.          printf("\n");
20.
21.          return (0);
22.  }
```

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Enumerated Types

- **Defines new data type**

```
typedef enum
   { sunday, monday, tuesday, wednesday,
   thursday, friday, saturday}
   day_t;
```

- **day_t is a new type**
  - has seven possible values
- **sunday is an enumeration constant represented by 0**
  - similarly, monday = 1, tuesday = 2, etc.

Enumerated Types

```
typedef enum
    { sunday, monday, tuesday, wednesday,
    tuesday, friday, saturday}
day_t;


day_t today;
```

- today is of type day_t
  - manipulated as other integers

```
today = sunday
today < monday
```

General syntax:

```
typedef enum
  { identifier_list }
enum_type;


enum_type variable_identifier;
```

Enumerated Types

```
typedef enum
  { sunday, monday, tuesday, wednesday,
  tuesday, friday, saturday}
day_t;

day_t today;

if (today == saturday)
        tomorrow = sunday
else
        tomorrow = (day_t)(today + 1)

today = friday + 3;
```

Enumerated Type for Budget Expenses

```
1.  /*   Program demonstrating the use of an enumerated type */
2.
3.  #include <stdio.h>
4.
5.  typedef enum
6.        {entertainment, rent, utilities, food, clothing,
7.         automobile, insurance, miscellaneous}
8.  expense_t;
9.
10. void print_expense(expense_t expense_kind);
11.
```

(continued)

GIT – Computer Engineering Department

```
12.   int
13.   main(void)
14.   {
15.         expense_t expense_kind;
16.
17.         scanf("%d", &expense_kind);
18.         printf("Expense code represents ");
19.         print_expense(expense_kind);
20.         printf(".\n");
21.
22.         return (0);
23.   }
24.
```

```
25.    /*
26.     * Display string corresponding to a value of type expense_t
27.     */
28.    void
29.    print_expense(expense_t expense_kind)
30.    {
31.          switch (expense_kind) {
32.          case entertainment:
33.                printf("entertainment");
34.                break;
35.
36.          case rent:
37.                printf("rent");
38.                break;
39.
40.          case utilities:
41.                printf("utilities");
42.                break;
43.
44.          case food:
45.                printf("food");
46.                break;
47.
48.          case clothing:
49.                printf("clothing");
50.                break;
51.
52.          case automobile:
53.                printf("automobile");
54.                break;
55.
56.          case insurance:
57.                printf("insurance");
58.                break;
59.
60.          case miscellaneous:
61.                printf("miscellaneous");
62.                break;
63.
64.          default:
65.                printf("\n*** INVALID CODE ***\n");
66.          }
67.    }
```

```
 4.
 5.  typedef enum
 6.         {monday, tuesday, wednesday, thursday, friday,
 7.          saturday, sunday}
 8.  day_t;
 9.
10.  void print_day(day_t day);
11.
12.  int
13.  main(void)
14.  {
15.        double week_hours, day_hours;
16.        day_t today;
17.
18.        week_hours = 0.0;
19.        for  (today = monday;  today <= friday;  ++today) {
20.            printf("Enter hours for ");
21.            print_day(today);
22.            printf("> ");
23.            scanf("%lf", &day_hours);
24.            week_hours += day_hours;
25.        }
26.
27.        printf("\nTotal weekly hours are %.2f\n", week_hours);
28.
29.        return (0);
30.  }
```

- Equation: f(x)
- Root: value k where f(k)=0
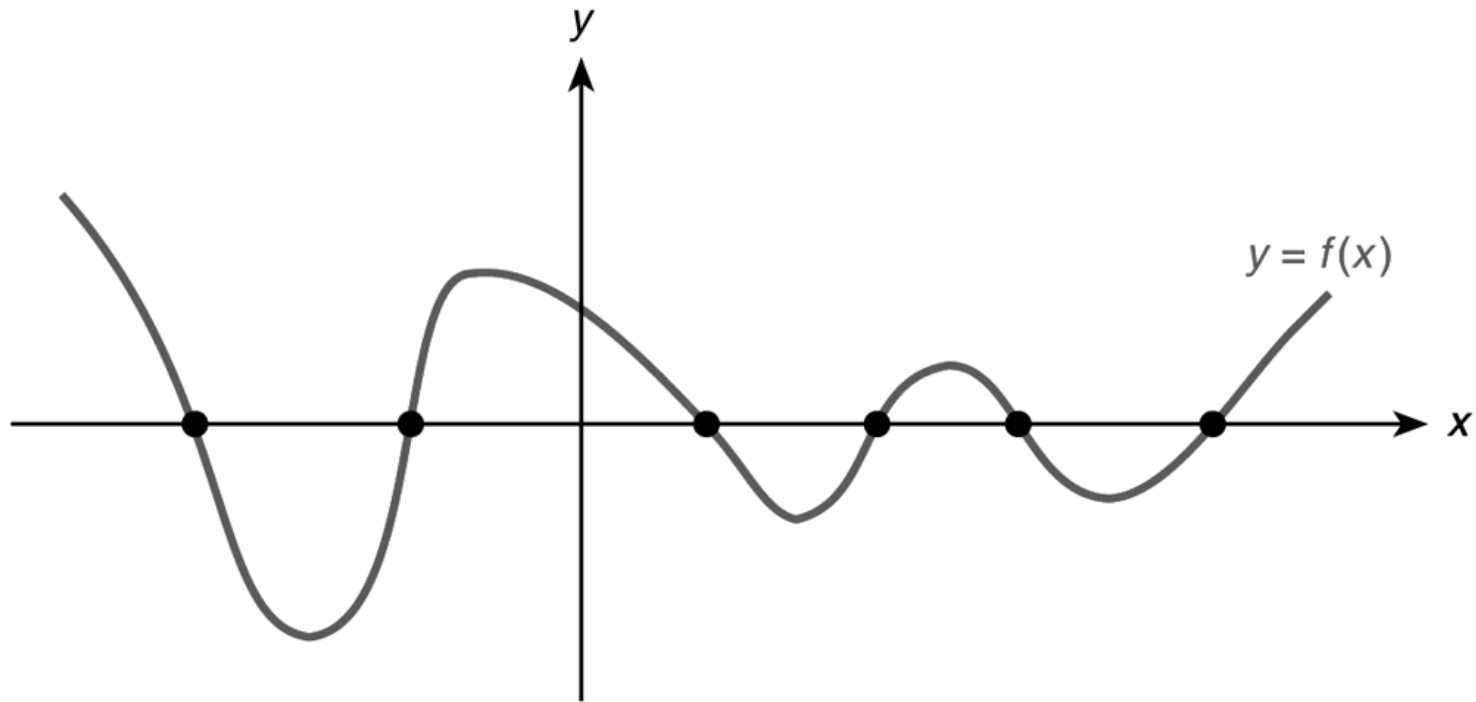
Finding Roots of Equations

- Bisection method:
  - Generate approximate roots until true root is found
    - or the difference is small (less than epsilon 0.00001)
- Bisection function is more usable if can find the root of any function
  - Function should be a parameter to the bisection function

Function Parameter

- Declaring a function parameter
  - including function prototype in the parameter list
 - as in the following evaluate function

```
1.  /*
2.   * Evaluate a function at three points, displaying results.
3.   */
4.  void
5.  evaluate(double f(double f_arg), double pt1, double pt2, double pt3)
6.  {
7.      printf("f(%.5f) = %.5f\n", pt1, f(pt1));
8.      printf("f(%.5f) = %.5f\n", pt2, f(pt2));
9.      printf("f(%.5f) = %.5f\n", pt3, f(pt3));
10. }
```

- Calling the function
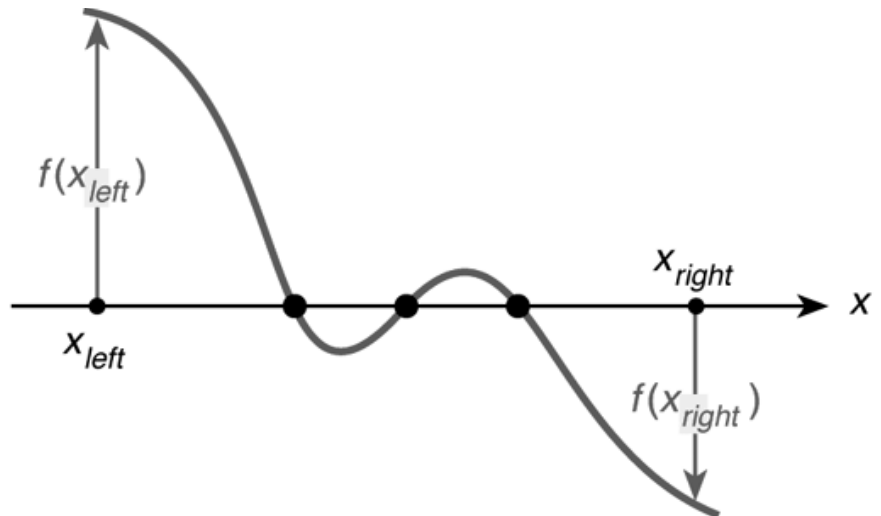
      evaluate(sqrt, 0.25, 25.0, 100);

Case Study: Bisection Method

- Problem: Find approximate root of a function on an interval that contains an odd number of roots

- Analysis
  - Change of sign in the interval: odd number of roots
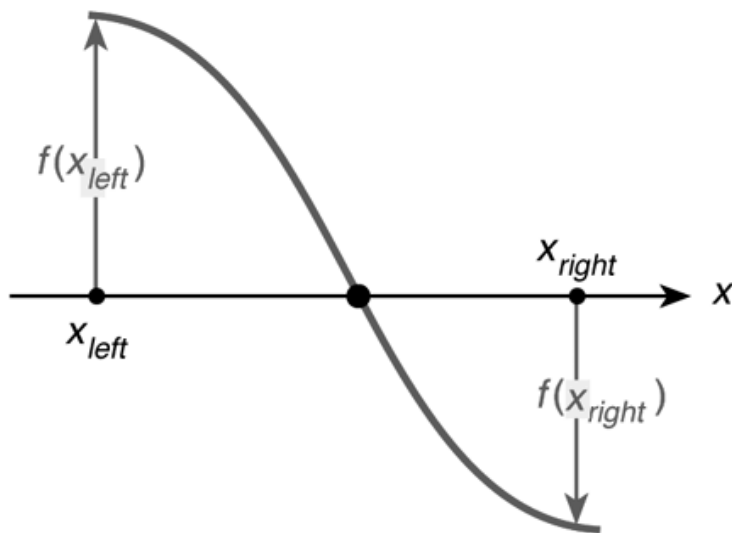

(a)
One root

(b)
Three roots

Case Study: Bisection Method

- Problem: Find approximate root of a function on an interval that contains an odd number of roots

- Analysis
  - Assume change of sign in the interval [x_left, x_right]
    - only one root
  - Assume f(x) is continous on the interval
  - Let x_mid = (x_left + x_right) / 2.0

- Three possibilities
  - root is in the lower half
  - root is in the upper half
  - f(x_mid) = 0
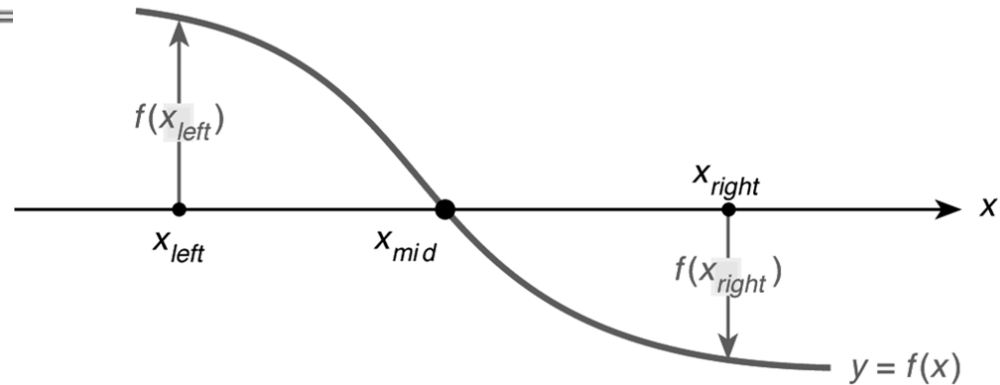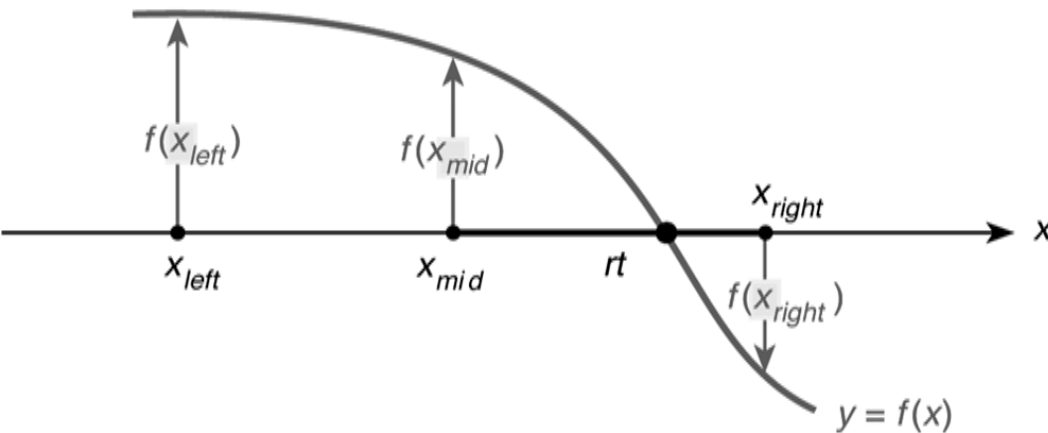
Three possibilities
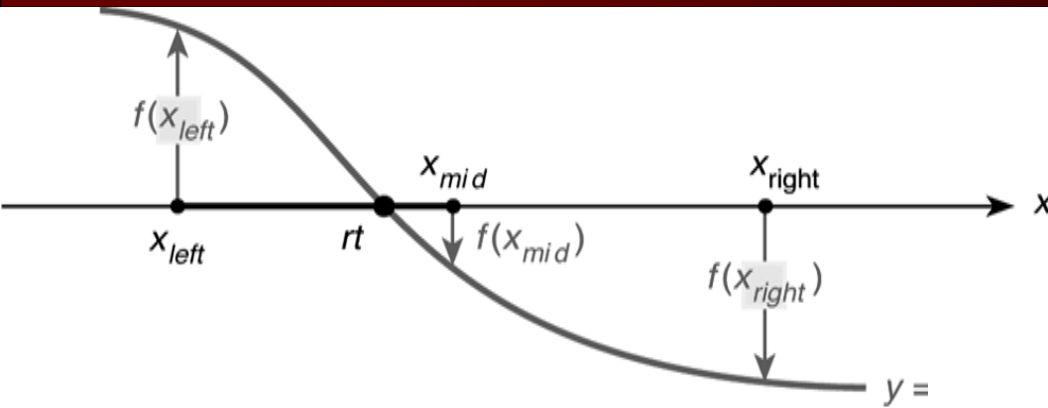
(a)
The root *rt* is in the half interval $[x_{left}, x_{mid}]$.

(b)
The root *rt* is in the half interval $[x_{mid}, x_{right}]$.

(c)
$f(x_{mid}) = 0.0$

Case Study: Bisection Method

- Inputs:
  - x_left - double
  - x_right - double
  - epsilon - double
  - funtion – double f (double farg)

- Outputs
  - root - double
  - error – int (indicating possiblem error in computation)

if the interval contains even number of roots

    set error flag

else

    clear error flag

    repeat until interval is larger than epsilon and root not found

        compute function value at the midpoint

        if the function value is zero

            the midpoint is the root

        if the root is in left half

            change right end to midpoint

        else

            change left end to midpoint

    return the midpoint as the root

```
7.   #include <stdio.h>
8.   #include <math.h>
9.
10.  #define FALSE 0
11.  #define TRUE 1
12.
13.  double bisect(double x_left, double x_right, double epsilon,
14.                double f(double farg), int *errp);
15.  double g(double x);
16.  double h(double x);
17.
18.  int
19.  main(void)
20.  {
21.       double x_left, x_right, /* left, right endpoints of interval  */
22.              epsilon,         /* error tolerance         */
23.              root;
24.       int    error;
25.
26.       /*  Get endpoints and error tolerance from user          */
27.       printf("\nEnter interval endpoints> ");
28.       scanf("%lf%lf", &x_left, &x_right);
29.       printf("\nEnter tolerance> ");
30.       scanf("%lf", &epsilon);
31.
32.       /*  Use bisect function to look for roots of g and h       */
33.       printf("\n\nFunction g");
34.       root = bisect(x_left, x_right, epsilon, g, &error);
35.       if (!error)
36.            printf("\n   g(%.7f) = %e\n", root, g(root));
37.
38.       printf("\n\nFunction h");
39.       root = bisect(x_left, x_right, epsilon, h, &error);
40.       if (!error)
41.            printf("\n   h(%.7f) = %e\n", root, h(root));
42.       return (0);
43.  }
```

```
45.  /*
46.   *   Implements the bisection method for finding a root of a function f.
47.   *   Finds a root (and sets output parameter error flag to FALSE) if
48.   *   signs of fp(x_left) and fp(x_right) are different. Otherwise sets
49.   *   output parameter error flag to TRUE.
50.   */
51.  double
52.  bisect(double x_left,          /* input  - endpoints of interval in */
53.         double x_right,         /*                which to look for a root */
54.         double epsilon,         /* input  - error tolerance            */
55.         double f(double farg),  /* input  - the function               */
56.         int    *errp)           /* output - error flag                 */
57.  {
58.       double x_mid,      /* midpoint of interval */
59.              f_left,     /* f(x_left)             */
60.              f_mid,      /* f(x_mid)              */
61.              f_right;    /* f(x_right)            */
62.       int    root_found = FALSE;
63.
64.       /* Computes function values at initial endpoints of interval  */
65.       f_left = f(x_left);
66.       f_right = f(x_right);
67.
68.       /* If no change of sign occurs on the interval there is not a
69.          unique root. Searches for the unique root if there is one.*/
70.       if (f_left * f_right > 0) {   /* same sign */
71.            *errp = TRUE;
72.            printf("\nMay be no root in [%.7f, %.7f]", x_left, x_right);
73.       } else {
74.            *errp = FALSE;
75.
```

```c
75.
76.                 /*  Searches as long as interval size is large enough
77.                     and no root has been found                         */
78.             while (fabs(x_right - x_left) > epsilon  &&  !root_found) {
79.
80.                 /* Computes midpoint and function value at midpoint */
81.                 x_mid = (x_left + x_right) / 2.0;
82.                 f_mid = f(x_mid);
83.                 if (f_mid == 0.0)  {                /* Here's the root    */
84.                         root_found = TRUE;
85.                 } else if (f_left * f_mid < 0.0) {/* Root in [x_left,x_mid]*/
86.                         x_right = x_mid;
87.                 } else {                           /* Root in [x_mid,x_right]*/
88.                         x_left = x_mid;
89.                         f_left = f_mid;
90.                 }
91.
92.                 /* Prints root and interval or new interval */
93.                 if (root_found)
94.                         printf("\nRoot found at x = %.7f, midpoint of [%.7f,
95.                                 %.7f]",
96.                                 x_mid, x_left, x_right);
97.                 else
98.                         printf("\nNew interval is [%.7f, %.7f]",
99.                                 x_left, x_right);
100.            }
101.        }
102.
103.    /*  If there is a root, it is the midpoint of [x_left, x_right]     */
104.    return ((x_left + x_right) / 2.0);
105. }
106.
```

```
107.  /*   Functions for which roots are sought                        */
108.
109.  /*     3      2
110.   *  5x  - 2x  + 3
111.   */
112.  double
113.  g(double x)
114.  {
115.       return (5 * pow(x, 3.0) - 2 * pow(x, 2.0) + 3);
116.  }
117.
118.  /*     4      2
119.   *  x  - 3x  - 8
120.   */
121.  double
122.  h(double x)
123.  {
124.       return (pow(x, 4.0) - 3 * pow(x, 2.0) - 8);
125.  }
```

Sample Run of Bisection Program

```
Enter interval endpoints> -1.0   1.0
Enter tolerance> 0.001

Function g
New interval is [-1.0000000, 0.0000000]
New interval is [-1.0000000, -0.5000000]
New interval is [-0.7500000, -0.5000000]
New interval is [-0.7500000, -0.6250000]
New interval is [-0.7500000, -0.6875000]
New interval is [-0.7500000, -0.7187500]
New interval is [-0.7343750, -0.7187500]
New interval is [-0.7343750, -0.7265625]
New interval is [-0.7304688, -0.7265625]
New interval is [-0.7304688, -0.7285156]
New interval is [-0.7294922, -0.7285156]
    g(-0.7290039) = -2.697494e-05

Function h
May be no root in [-1.0000000, 1.0000000]
```