Functions and Storage Classes

Unit 3 Objectives

- Define and call functions
  - — Pass arguments to functions
  - — Use return value of a function
  - — Pass array address to a function
  - — Define and use recursive functions
- Manage data with storage classes
  - — Automatic
  - — Register
  - — External
  - — Static
- Write and compile a C program contained in several source files

Functions

Functions - Overview

- C programs are collections of functions
- Functions and procedures
- Modular programming:
  - — Break large tasks into smaller ones
  - — Avoid duplication
  - — Programs easier to
    - Read
    - Write
    - Debug
    - Maintain

Function Definition

- Block of code with a name
- Information (arguments) can be passed to it
- One value can be returned
- No special order in source file

```
type identifier (argument list)
argument declarations
{
    declarations
    statements
}
```

Function Call

function-name(optional arguments)

- Execution transferred to function
- Statements in function executed
- Return is made to "calling function"

```
 1  #include    <stdio.h>
 2
 3  main()
 4  {
 5
 6      intro();    /* function call */
 7      ...
 8
 9  }
10
11  intro()
12  {
13      printf("Welcome to the Information ");
14      printf("Retrieval System.\n\n");
15      printf("For assistance, type help ");
16      printf("at any time\n");
17  }
```

Function Arguments

- Used to pass information to a function
- Function call should have correct number, order, type
- Values sent to function
- Local to the function

```
 1  #include   <stdio.h>
 2  main()
 3  {
 4      float   amount = 250,/* Initial investment */
 5              interest = .075;  /* 7.5% */
 6
 7      print_growth(amount,interest);
 8      printf("main: amount is $%.2f\n", amount);
 9  }
10
11
12  print_growth(val,rate) /* val=250; rate=.075 */
13  float   val, rate;
14  {
15      val = (1 + rate) * val;
16      printf("Value after 1 year: $%.2f\n",val);
17  }
```

```
$ a.out
Value after 1 year: $268.75
main: amount is $250.00
```

Casting

(type) expression

- Cast operator used for temporary conversion
- Often used to supply correct type to function

```
 1  main()
 2  {
 3      int     x;
 4      f((long)x);
 5  }
 6
 7  f(val)
 8  long    val;    /* a long is expected */
 9  {
10      ...
11
12  }
```

– Other uses of the cast operator:

```
int     x,y;
float   result;

result = (float)x / (float) y;
```

Exercise – Function Calls and Arguments

1. Consider the following function:

```
func(one, two, sum)
int one, two ;
double  sum;
{
        double      num;
        • • •
}
```

   a. How many arguments does this function expect?

   b. What are the data types of one, two, and sum?

   c. Write a function call that calls func(), passing it valid arguments.

   d. What is the difference between the variables sum and num?

   e. What do sum and num have in common?

2. What is wrong with this function definition?

```
compute(x, y)
{
        int         x ;
        double      y ;
        • • •
}
```

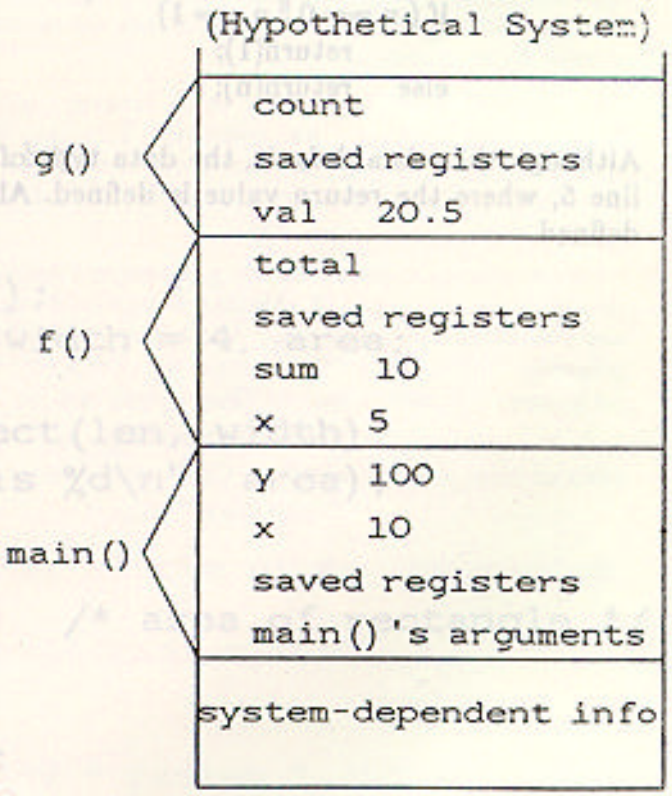
3. What will this program print?

```
main()
{
        int         x = 5 ;
        square(x) ;
        printf("main: x is %d\n", x) ;
}
square(val)
int     val;
{
        val = val * val ;
        printf("%d\n", val);
}
```

The Role of the Stack

- Memory used when a function is called
- Grows and shrinks as functions called, exited

```
main()
{
    int     x=10, y=100;

    f(5,10);

}

f(x, sum)
int x, sum;
{
    double  total;

    g(20.5);
     .
}

g(val)
double      val;
{
    int     count;
     .
}
```



return statement

- Returns execution to calling function
- Optional value returned

```
return;

return (expression);

return expression;
```

```
1  #include   <stdio.h>
2  main()
3  {
4      int area_rect();
5      int len = 50, width = 4, area;
6
7      area = area_rect(len, width);
8      printf("Area is %d\n", area);
9  }
10
11 int area_rect(l,w)   /* area of rectangle */
12 int l, w;
13 {
14     return(l * w);
15 }
```

Non-Integer Return

- If a function returns a non-integer:
  1. Declaration must be in/above calling function
  2. Definition must specify type
- If no type specified, int assumed
- **void** declares that a function returns no value

```
 1      #include   <stdio.h>
 2
 3      main()
 4      {
 5  ☞       double  area_circle(),
 6                  rad = 50.5, area;
 7
 8          area = area_circle(rad);
 9          printf("Area is %f\n",area);
10      }
11
12      /* Returns area of circle given the radius */
13
14  ☞   double  area_circle(radius)
15      double  radius;
16      {
17          return(3.14159 * radius * radius);
18      }
```

Conditional operator ?:

expression1? expression2 : expression3

- If *expression1* is true, value of expression is *expression2*, else *expression3*
- An expression, not a statement

```
/* Body of absolute */
/*  value function */

return (x >= 0 ? x : -x);
```

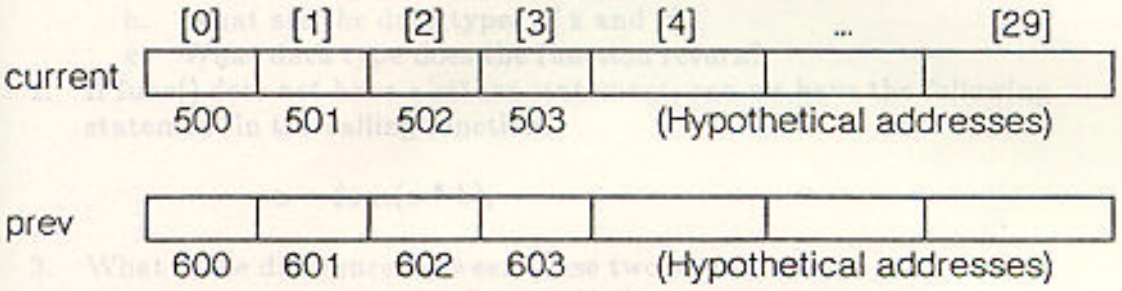
```
/* Alternative */

if (x >= 0)
     return(x);
else  return(-x);
```

- Other uses of conditional operator:

```
largest = x > max ? x : max;
```

Passing an Array Address to a Function

- Does not pass copy of entire array to a function
- Does pass starting address
- Function can access/change original array




```
 1  /* Copies array using a function */
 2  main()
 3  {
 4       void     stringcopy();
 5       char      current[30], prev[30];
 6       ...
 7       stringcopy(prev, current);
 8       ...
 9  }
10
11  /* Copies str2 into str1 */
12  void  stringcopy(str1, str2)     /* str1 = 600 */
13  char  str1[], str2[];            /* str2 = 500 */
14  {
15       int      i;
16       for (i = 0; str2[i] != '\0'; i++)
17                str1[i] = str2[i];  /*address[offset]*/
18       str1[i] = '\0';
19  }
```

Exercise - Function Arguments and Return Values

1. Consider the following function:

```
double   f(x,y)
int   x;
double     y;
{
```

   a. How many arguments does this function expect?
   b. What are the data types of x and y?
   c. What data type does the function return?

2. If func() does not have a return statement, can we have the following statement in the calling function?

```
x = func(a * b);
```

3. What is the difference between these two statements?

```
int  verify();

verify();
```

4. Write argument declaration statements next to the pointing hands (☞) that correspond with the function calls in main().

```
main()
{
      int         line[100];
      adjust(line[0]);
      sum(line);    /* or sum(&line[0]); */
}

adjust(arg)
☞
{
      ...
}

sum(arg)
☞
{
      ...
}
```

Storage Classes

Storage Classes Overview

- Alternatives for storing data
- Storage Classes:
  - Automatic
  - Register
  - External
  - Static
- Scope: Statements which can reference a variable
- Life: Length of time the contents are valid

Automatic Storage Class

- Scope: Local, block
- Life: Created when block entered
  Becomes non-existent when block exited
- Inner declarations supersede outer ones

```
 1  main()
 2  {
 3      void    sub();
 4      int     x;    /* Known only in main() */
 5
 6      x = getchar();
 7      if ( x == 'c' ) {
 8          float   x; /* Known in this block */
 9          x = .75;   /* refers to x in line 8 */
11          ...
12          }
13      sub(100);
14  }
15
16  void sub(x)
17  int  x;   /* Known only in sub() */
18  {
19      printf("%d\n", x * x * x);
20  }
```

Register Storage Class

- Speed and efficiency
- Machine registers used instead of stack
- Function parameters and automatics
- Scope and life: same as automatics
- Number available is system-dependent
- Data types: usually char, int, pointer

```
f(count, num)
register int    count, num;
{
        register int    i;
        ...
}
```

External Storage Class

- Global
- Variables defined outside a function, initialized to 0
- Scope: From definition to end of file
- Life: Life of program
- Advantage: Data sharing

```
 1        int      status;
 2        char     sysname[20];
 3
 4        main()
 5        {
 6                 fillbuf();
                   ...
21                 emptybuf();
                   ...
25        }
26
27        char     buf[1024];
28
29        fillbuf()
30        {
                   ...
38        }
39
40        emptybuf()
41        {
                   ...
49        }
```

Multiple Source Files

- C programs often kept in several files
- Keyword extern used to access external variables defined in another file

one.c

```
int    status;
char   sysname[20];


main()
{
        fillbuf();
        .
        if (expression)
                emptybuf();
}
```

two.c

```
extern int      status;
extern char     sysname[];
char    buf[1024];

fillbuf()
{

}

emptybuf()
{

        fillbuf();
}
```

```
$cc one.c two.c
```

Static Storage Class

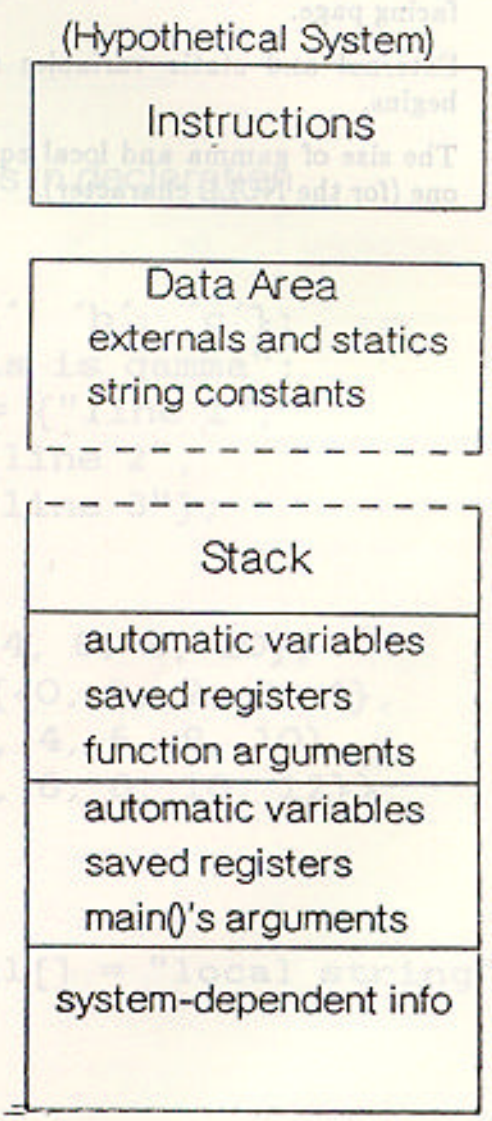
- Scope:
  - External: Declaration to end-of-file
    Not available to other files
  - Internal: Function, block
- Life: Life of program
- Value: Defaults to 0
- Advantages: Permanence and privacy

```
 1    static int   count;
 2
 3    g(amnt)
 4    int   amnt;
 5    {
 6            static int total = 1024;
 7            int     x = 0;
 8
 9            total -= amnt;
10
11    }
12
13    /* A static function is private */
14    /* to its source file */
1j    static  errmsg()
10    {
10
10    }
```

Where Variables are Stored

- External and Static Variables
  - Life of program
  - Values default to 0
  - Initialized before execution

- Automatic Variables
  - Temporary, block life
  - Default value undefined



Variable Initialization

- Automatic Variables

  Default value undefined

  Can not initialize arrays,structures in declaration

- External and Static Variables

  Default value is 0

  Can initialize arrays,structures in declaration

```
 1  char alpha [10];
 2  char beta [10] = {'a', 'b', 'c'};
 3  char gamma [] = "This is gamma";
 4  char delta [5][10] = {"line 1",
 5                        "line 2",
 6                        "line 3"};
 7
 8  int num1 [10];
 9  int num2 [10] = {2, 4, 6, 8, 10};
10  int num3 [5][10] = {{0, 1, 2, 3, 4},
11                      {2, 4, 6, 8, 10},
12                      {3, 6, 8, 10, 12}};
13
14  main()
15  {
16     static char local[] = "local string";
17
18  }
```

Storage Class Summary

Class	Scope	Life	Storage	Initialize Arrays, Structs	Default Value
automatic	block	block active	stack	no	undefined
register[1]	block	block active	machine register	no	undefined[2]
external[3]	declaration to end-of-file	permanent	data area	yes	0
static external[4]	declaration to end-of-file	permanent	data area	yes	0
static internal	block	permanent	data area	yes	0

1. Speed advantage
2. For function arguments, value passed
3. Can be accessed from other files
4. Can not be accessed from other files

Appendix

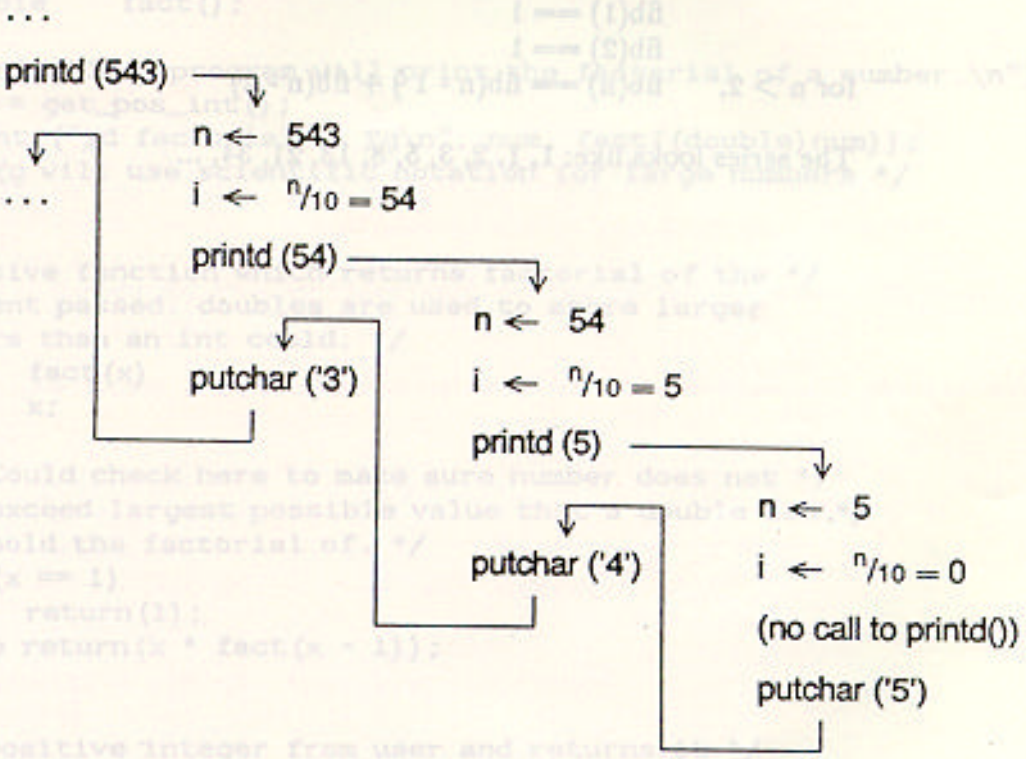
Recursive Functions

- A function may call itself
- Each invocation gets its own variables
- Warning: avoid infinite recursion

```
/* The %d part of printf() */

void printd(n)
int n;
{
    int     i;
    if (n < 0) {
            putchar('-');
            n = -n;
            }
    if ( (i = n/10) != 0)
            printd(i);
    putchar(n % 10 + '0');
}
```

Recursive Functions (con't)

Recursive flow through printd()



```
/* 3.Aa.c */
/* Prints a the factorial of a number supplied by the user */

#include  <stdio.h>

main()
{
     int  num;
     double    fact();

     printf("This program will print the factorial of a number.\n");
     num = get_pos_int();
     printf("%d factorial is %g\n", num, fact((double)num));
     /* %g will use scientific notation for large numbers */
}

/* Recursive function which returns factorial of the */
/* argument passed. doubles are used to store larger
/* numbers than an int could. */
double    fact(x)
double    x;
{
     /* Could check here to make sure number does not */
     /* exceed largest possible value that a double can */
     /* hold the factorial of. */
     if (x == 1)
          return(1);
     else return(x * fact(x - 1));
}

/* Gets positive integer from user and returns it */
get_pos_int()
{
     int  val;
     for (;;) {
          printf("Please enter a positive integer: ");
          if (scanf("%d",&val) != 1) {
               printf("\tError: non-integer\n");
               while (getchar() != '\n')
                    ;    /* Clear line */
               continue;
               }
```

```
/* 3.Ab.c */
/* Prints the n-th number in the Fibonnaci Series. n is supplied */
/* by the user. */

#include  <stdio.h>

main()
{
   int     num;
   double  fact();

   printf("This program will print a number ");
   printf("in the Fibonnaci Series.\n");
   printf("Which number in the series would you like to see?\n\n");
   num = get_pos_int();
   printf("Number %d in the Fibonnaci Series is %d\n", num, fib(num));
}

/* Recursive function which returns n-th number in Fibnnaci Series */
fib(n)
int  n;
{
     if (n == 1 || n == 2)
          return(1);
     else return(fib(n - 1) + fib(n - 2));
}

/* Gets positive integer from user and returns it */
get_pos_int()
{
     int  val;
     for (;;) {
          printf("Please enter a positive integer: ");
          if (scanf("%d",&val) != 1) {
               printf("\tError: non-integer\n");
               while (getchar() != '\n')
                    ;     /* Clear line */
               continue;
               }
          if (val <= 0) {
               printf("\tError: non-positive integer\n");
               continue;
```