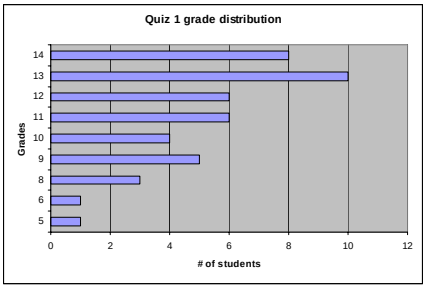
CS 201
Functions

Debzani Deb

Quiz 1 Grade Distribution



Overview of Today's Lecture

- Building Program from Existing Information
- Library Functions
- Functions without Arguments

Existing Information

- Programmers seldom start from scratch when writing a program.
- Typically, you will reuse work that has been done by yourself or others
  - For example, using `printf` and `scanf`
- You start with your algorithm, and then implement it piece by piece
  - When implementing these pieces, you can save effort by reusing functionality.

Library Functions

- Predefined Functions and Code Reuse
- C Library Functions
- A Look at Where We Are Heading

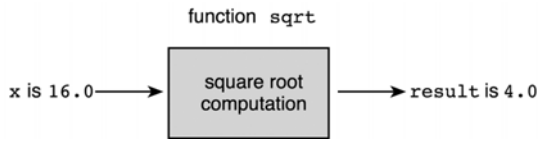
Predefined Functions and Code Reuse

- The primary goal of software engineering is to write error-free code.
- Reusing code that has already been written & tested is one way to achieve this.
  - "Why reinvent the wheel?"
- C promotes reuse by providing many predefined functions. e.g.
  - Mathematical computations.
  - Input/Output: e.g. `printf`, `scanf`
- C's standard math library defines a function named `sqrt` that performs the square root computation. It is called like:

```
y = sqrt(x)
```

- This passes the argument `x` to the function `sqrt`. After the function executes, the result is assigned to the left hand side variable `y`.

Function sqrt.



Function sqrt as a black box.

```
first_sqrt = sqrt(25.0);
second_sqrt = sqrt(second);
third_sqrt = sqrt(first+second);
Z = 5.7 + sqrt(num);
```


C Library Functions

- The next slide lists some commonly used mathematical functions (Table 3.1 in the text)
- In order to use them you must use `#include` with the appropriate library.
  - Example, to use function `sqrt` you must include `math.h`.
- If one of the functions in the next slide is called with a numeric argument that is not of the argument type listed, the argument value is converted to the required type before it is used.
  - Conversion of type `int` to type `double` cause no problems
  - Conversion of type `double` to type `int` leads to the loss of any fractional part.
- Make sure you look at documentation for the function so you use it correctly.

Some Mathematical Library Functions

Function	Header File	Purpose	Arguments	Result
abs(x)	<stdlib.h>	Returns the absolute value of its integer argument x.	int	int
sin(x),cos(x), tan(x)	<math.h>	Returns the sine, cosine, or tangent of angle x.	double (in radians)	double
log(x)	<math.h>	Returns the natural log of x.	double (must be positive)	double
pow(x,y)	<math.h>	Returns x^y	double, double	double
sqrt(x)	<math.h>	$\sqrt{x}$	double (must be positive)	double

Function we have seen so far

- We've seen a few other I/O library functions
  - printf, scanf
  - fprintf, fscanf
  - fopen, fclose
  - To use them, have to use `#include <stdio.h>`
- Mathematical Functions
  - sqrt, pow, sin, cos etc.
  - To use them, have to use `#include <math.h>`
- We use C's predefined functions as building blocks to construct a new program.

Where We are Heading?

- C also allows us to write **our own functions**.
- We could write our own functions to find area and find circumference of a circle.
  - Function find_area(r) returns the area of a circle with radius r.
  - Function find_circum(r) returns the circumference of a circle with radius r.
  - The following statements can be used to find these values.

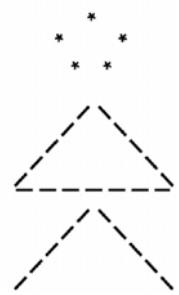
```
area = find_area(r);
circum = find_circum(r);
```


Top Down Design

- Use the top-down approach for analyzing all complex problems.
- The solution to any complex problem is conceptually simpler if viewed hierarchically as a tree of subproblems.
- It is more convenient to design your solution first with rough blocks, and then refine them gradually.
- You first break a problem up into its major subproblems and then solve those subproblems to derive the solution to the original problem.

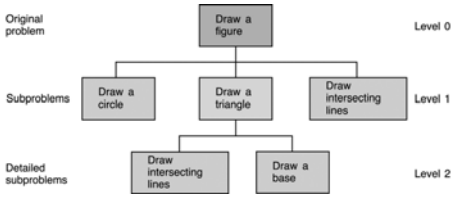
Example: Top-down approach

- Drawing a Stick Figure in the screen as an example of problem solving with Top-down design approach.
- We can draw this figure with the basic three components
  - Circle
  - Intersecting lines
  - Base line

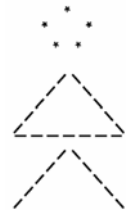


Structure Chart for Drawing Stick Figure

- Structure chart is an software engineering documentation tool.



Function `main` for Stick Figure



```
/*
 * Draws a stick figure
 */

#include <stdio.h>

/* function prototypes                                  */

void draw_circle(void);      /* Draws a circle          */
void draw_intersect(void);   /* Draws intersecting lines */
void draw_base(void);        /* Draws a base line       */
void draw_triangle(void);    /* Draws a triangle        */

int
main(void)
{
      /* Draw a circle.  */
      draw_circle();

      /* Draw a triangle.  */
      draw_triangle();

      /* Draw intersecting lines.  */
      draw_intersect();

      return (0);
}
```


Void Functions without Arguments

- Functions that do not have arguments and return no values.
  - Output is normally placed in some place else (e.g. screen)
- Why would you want to do these?
  - They can help with top down design of your program.
  - Instead of writing all of your code in your `main` function, separate it into separate functions for each subproblem.

Void Functions Without Arguments

- Function Prototypes
- Function Definitions
- Local variables.
- Placement of Functions in a Program
- Program Style
- Advantages of Using Function Subprograms
  - **Procedural Abstraction**
  - **Reuse of Functions.**

Function Prototype (1)

```
/* This program draws a circle in the screen */
#include <stdio.h>
/* Function prototypes */
void draw_circle(void); /* Draws a circle */

int main(void)
{
   draw_circle();
   return (0);
}
/* Draws a circle */
void draw_circle(void) {
   printf("  * *\n");
   printf(" *   *\n");
   printf("  * *\n");
}
```

Function Prototype (2)

- Like other identifiers in C, a function must be declared before it can be referenced.
- To do this, you can add a **function prototype** before `main` to tell the compiler what functions you are planning to use.
- A function prototype tells the C compiler:
  1. The data type the function will return
     - For example, the `sqrt` function returns a type of double.
  2. The function name
  3. Information about the arguments that the function expects.
     - The `sqrt` function expects a double argument.
- So the function prototype for `sqrt` would be:

```
double sqrt(double);
```


More on void Functions

- `void draw_circle(void);` is a void function
  - **Void function -** does not return a value
    - The function just does something without communicating anything back to its caller.
  - If the arguments are void as well, it means the function doesn't take any arguments.
- Now, we can understand what our main function means:

```
int main(void)
```

- This means that the function `main` takes no arguments, and returns an `int`

Function Definition (1)

```
/* This program draws a circle in the screen */
#include <stdio.h>
/* Function prototypes */
void draw_circle(void); /* Draws a circle */

int main(void)
{
   draw_circle();
   return (0);
}
/* Draws a circle */
void draw_circle(void) {
   printf("  * *\n");
   printf(" *   *\n");
   printf("  * *\n");
}
```


Function Definition (2)

- The prototype tells the compiler what arguments the function takes and what it returns, but not what it does.
- We define our own functions just like we do the `main` function
  - **Function Header** – The same as the prototype, except it is not ended by the symbol ;
  - **Function Body –** A code block enclosed by {}, containing variable declarations and executable statements.
- In the function body, we define what actually the function does
  - In this case, we call `printf` 3 times to draw a circle.
  - Because it is a void function, we can omit the return statement.
- Control returns to `main` after the circle has been drawn.

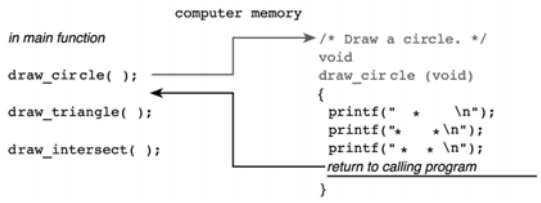
Placement of Functions in a program

- In general, we will declare all of our function prototypes at the beginning (after `#include` or `#define`)
- This is followed by the `main` function
- After that, we define all of our functions.
- However, this is just a convention.
- As long as a function's prototype appears before it is used, it doesn't matter where in the file it is defined.
- The order we define them in does not have any impact on how they are executed

Execution Order of Functions

- Execution order of functions is determined by the order of execution of the function call statements.
- Because the prototypes for the function subprograms appear before the `main` function, the compiler processes the function prototypes before it translates the `main` function.
- The information in each prototype enables the compiler to correctly translate a call to that function.
- After compiling the `main` function, the compiler translates each function subprogram.
- At the end of a function, control always returns to the point where it was called.

Figure 3.15 Flow of Control Between the main Function and a Function Subprogram



Program Style

- Each function should begin with a comment that describes its purpose.
- If the function subprograms were more complex, we would include comments on each major algorithm step just as we do in function `main`.
- It is recommended that you put prototypes for all functions at the top, and then define them all after `main`.

Advantages of Using Function Subprograms

- There are two major reasons:

1. A large problem can be solved easily by breaking it up into several small problems and giving the responsibility of a set of functions to a specific programmer.
   - It is easer to write two 10 line functions than one 20 line one and two smaller functions will be easier to read than one long one.
2. They can simplify programming tasks because existing functions can be reused as the building blocks for new programs.
   - Really useful functions can be bundled into libraries.

Procedural Abstraction

- **Procedural Abstraction –** A programming technique in which a `main` function consists of a sequence of function calls and each function is implemented separately.
- All of the details of the implementation to a particular subproblem is placed in a separate function.
- The main functions becomes a more abstract outline of what the program does.
  - When you begin writing your program, just write out your algorithm in your main function.
  - Take each step of the algorithm and write a function that performs it for you.
- Focusing on one function at a time is much easier than trying to write the complete program at once.

Reuse of Function Subprograms

- Functions can be executed more than once in a program.
  - Reduces the overall length of the program and the chance of error.
- Once you have written and tested a function, you can use it in other programs or functions.

A good use of void functions – A separate function to display instructions for the user.

```
1.  /*
2.   * Displays instructions to a user of program to compute
3.   * the area and circumference of a circle.
4.   */
5.  void
6.  instruct(void)
7.  {
8.        printf("This program computes the area\n");
9.        printf("and circumference of a circle.\n\n");
10.       printf("To use this program, enter the radius of\n");
11.       printf("the circle after the prompt: Enter radius>\n");
12. }

This program computes the area
and circumference of a circle.

To use this program, enter the radius of
the circle after the prompt: Enter radius>
```