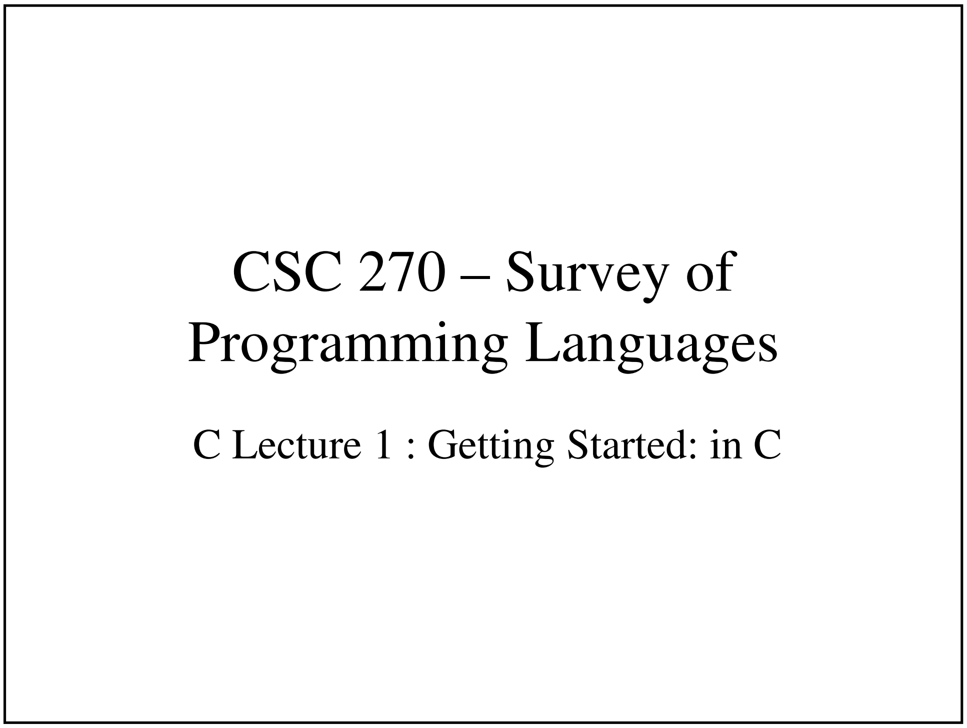
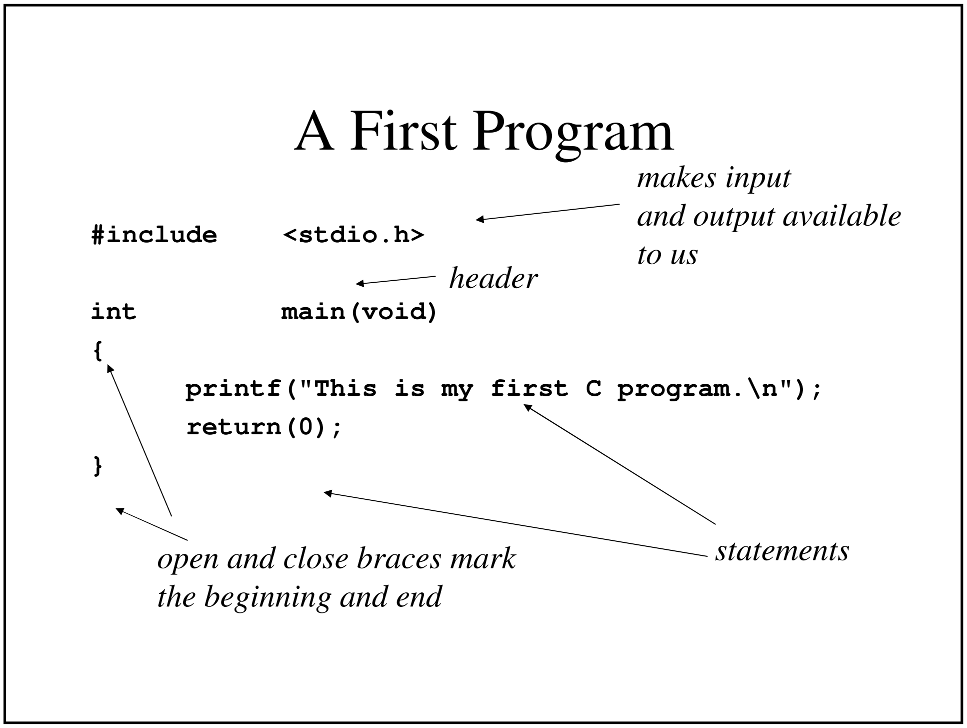


CSC 270 – Survey of Programming Languages

C Lecture 1 : Getting Started: in C



A First Program

```
#include    <stdio.h>

int         main(void)
{
      printf("This is my first C program.\n");
      return(0);
}
```

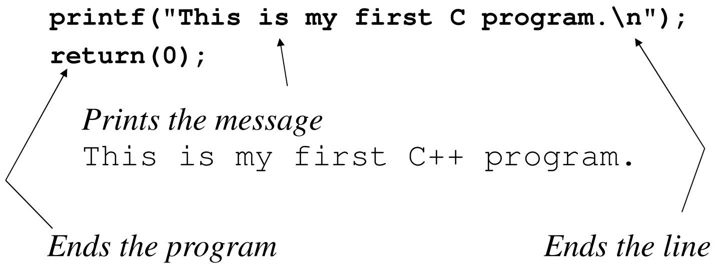
makes input and output available to us

header

statements

open and close braces mark the beginning and end

A First Program – What Does It Do?



```
printf("This is my first C program.\n");
return(0);
```

Prints the message

```
This is my first C++ program.
```

Ends the program

Ends the line

Identifier Rules

- An identifier must begin with a letter or an underscore _
- C is case sensitive upper case (capital) or lower case letters are considered different characters. **`Average`**, **`average`** and **`AVERAGE`** are three different identifiers.
- Numbers can also appear after the first character.

Identifier Rules (continued)

- Identifiers can be as long as you want but names that are too long usually are too cumbersome.
- However, C only considers the first 6 (external identifiers) or first 31 (internal identifiers) significant.
- Identifiers cannot be reserved words (special words like **`int`**, **`main`**, etc.)

`Average3.cpp`

```
#include    <stdio.h>

int         main(void)
{
      int         value1, value2, value3;
      float sum, average;

      printf("What is the first value? ");
      scanf("%d", &value1);
```

Read

```
      printf("What is the second value? ");
      scanf("%d", &value2);
```

Indicates that we are reading an integer

```
      printf("What is the third value? ");
      scanf("%d", &value3);

      sum = value1 + value2 + value3;
      average = sum / 3;

      printf("Average = %f\n", average);
      return(0);
}
```

`scanf` *needs the* `&` *before the identifier*

payroll.c

```
#include          <stdio.h>

int   main(void)
{
      float rate, hours, gross;

      printf("What is your hourly pay rate ? ");
      scanf("%f",&rate);

      printf("How many hours did you work ? ");
      scanf("%f", &hours);
```

```
     gross = rate * hours;
     printf("Your gross pay is $%f\n", gross);
     return(0);
}
```

Print a float value

Comments

- In C, anything beginning with **/*** and ending with ***/** is considered a comment.

payroll.c

```
#include          <stdio.h>
/*
 * This program calculates the gross pay for an
 * hourly worker.
 * Inputs - hourly pay rate and number of hours
 *          worked
 * Output - Gross pay
 */
int   main(void)
{
      float rate, hours, gross;

      /* Get the hourly rate */
      printf("What is your hourly pay rate ? ");
      scanf("%f",&rate);
```

```
      /* Get the number of hours worked */
      printf("How many hours did you work ? ");
      scanf("%f", &hours);

      /* Calculate and display the gross pay */
      gross = rate * hours;
      printf("Your gross pay is $%f\n", gross);
      return(0);
}
```

Character Data

- All of our programs so far have used variables to store numbers, not words.
- We can store one or more characters by writing:

  `char   x, s[10];`

  - **`x`** can hold one and only one character
  - **`s`** can up to nine characters.
- For now, we use character data for input and output only.

A program that uses a character variable

```
#include      <stdio.h>

/* A very polite program that greets you by name */
int    main(void)
{
   char       name[25];

   /* Ask the user his/her name */
   printf("What is your name ? ");
   scanf("%s", &name);

   /* Greet the user */
   printf("Glad to meet you, %s\n.", name);
   return(0);
}
```

if and *if-else (continued)*

- The general form is:

```
if (expression)
  statement;
```

or

```
if (expression)
  statement;
else
  statement;
```

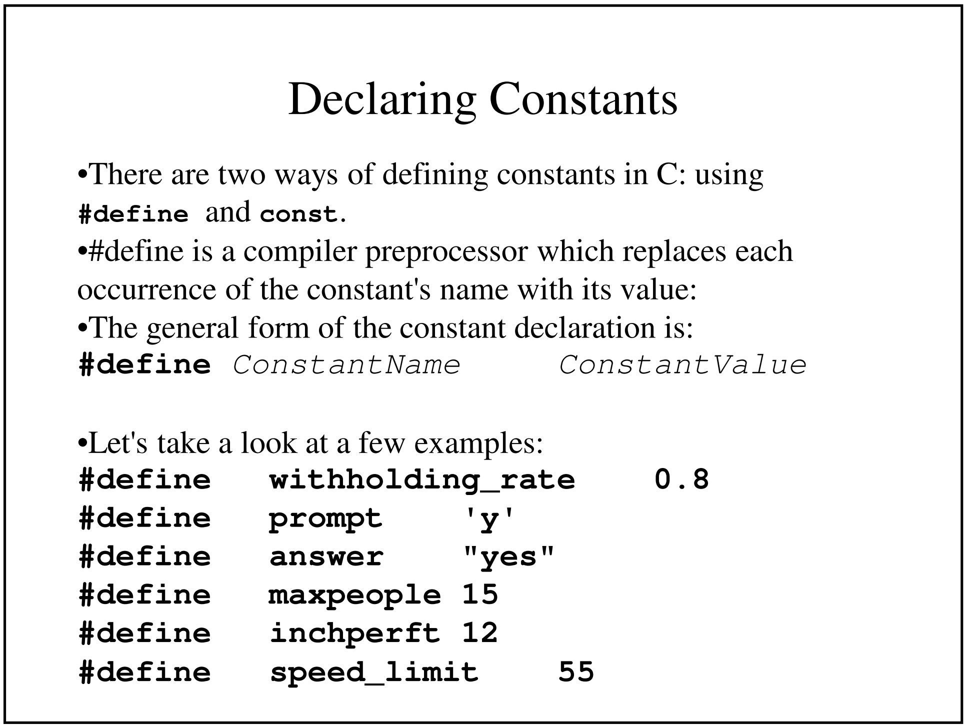
IsItNeg.c

```
#include      <stdio.h>
/*
 * Tell a user if a number is negative
 * or non-negative
 */
int    main(void)
{
       float  number;
      /* Ask the user for a number */
       printf("Please enter a number ? ");
       scanf("%f", &number);
```

```
     // Print whether the number is negative or not
     if (number < 0)
          printf("%f is a negative number\n",
                  number);
     else
          printf("%f is NOT a negative number\n",
                  number);

      return(0);
}
```

```
     // Print the warning if appropriate
     if (speed > 55)   {
          printf("**BE CAREFUL!**");
          printf("You are driving too fast!\n");
     }
     return(0);
}
```



Declaring Constants

•There are two ways of defining constants in C: using `#define` and `const`.

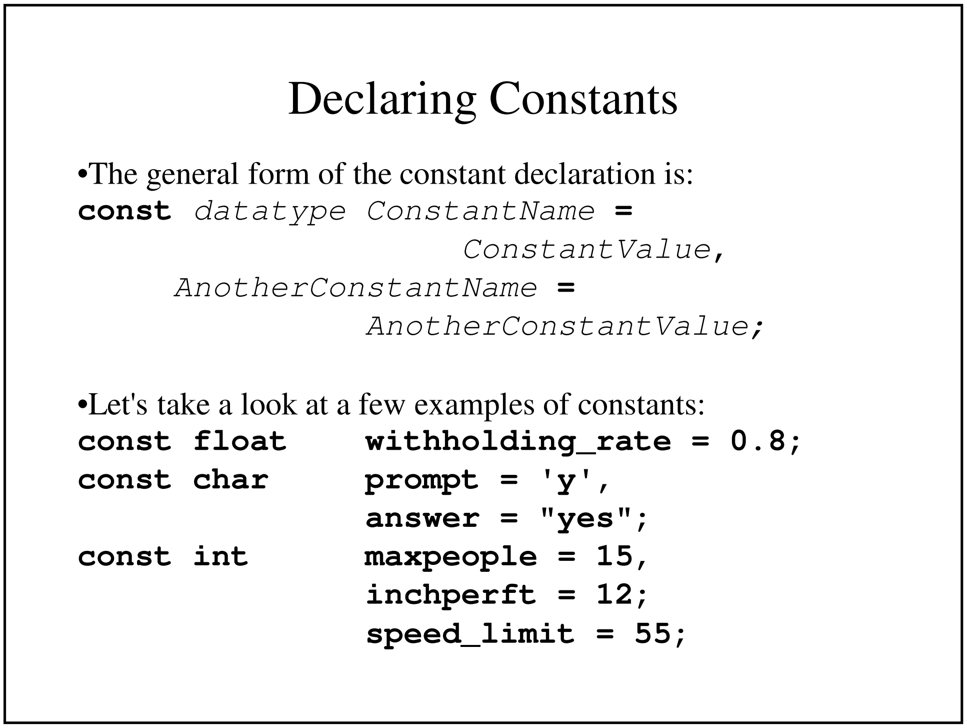
•#define is a compiler preprocessor which replaces each occurrence of the constant's name with its value:

•The general form of the constant declaration is:

```
#define ConstantName      ConstantValue
```

•Let's take a look at a few examples:

```
#define   withholding_rate    0.8
#define   prompt    'y'
#define   answer    "yes"
#define   maxpeople 15
#define   inchperft 12
#define   speed_limit    55
```



Declaring Constants

•The general form of the constant declaration is:

```
const datatype ConstantName =
                    ConstantValue,
     AnotherConstantName =
               AnotherConstantValue;
```

•Let's take a look at a few examples of constants:

```
const float    withholding_rate = 0.8;
const char     prompt = 'y',
               answer = "yes";
const int      maxpeople = 15,
               inchperft = 12;
               speed_limit = 55;
```

Counting Loops

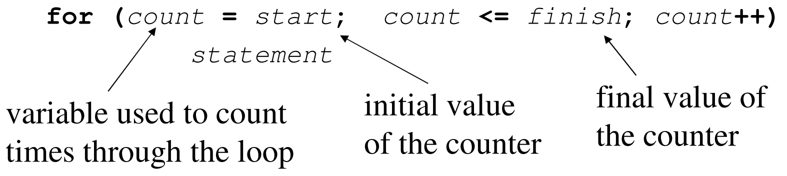
- We use a for loop to write counting loops
- In C, it looks like this:

```
for (count = start;  count <= finish; count++)
          statement
```

- or

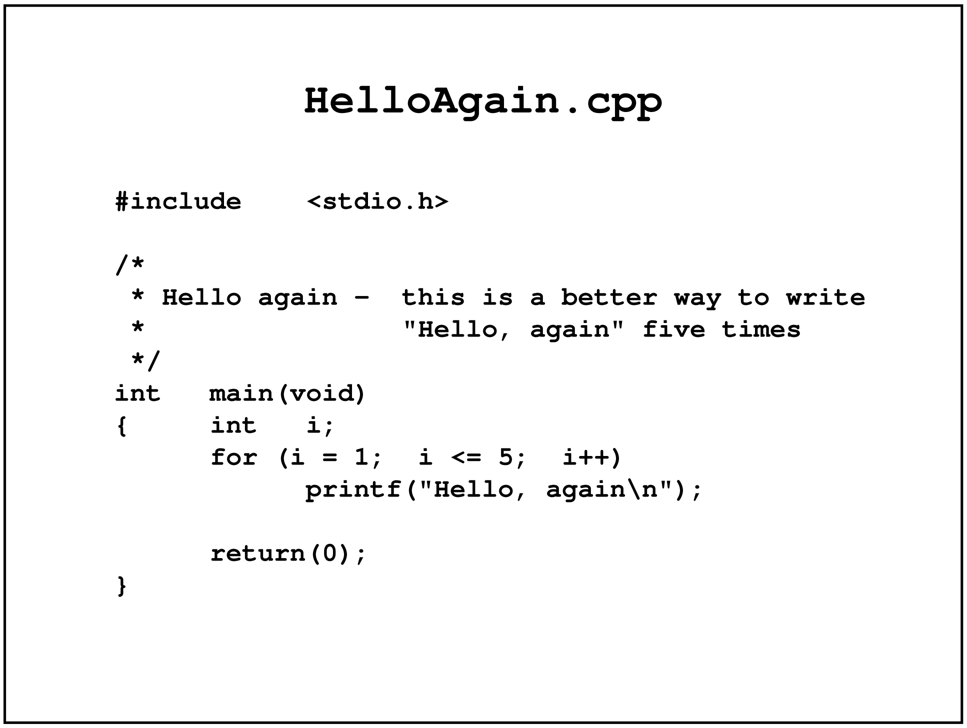
```
for (count = start;  count <= finish; count++)  {
          statements
}
```

Counting Loops (continued)



```
for (count = start;  count <= finish; count++)
          statement
```

variable used to count times through the loop
initial value of the counter
final value of the counter

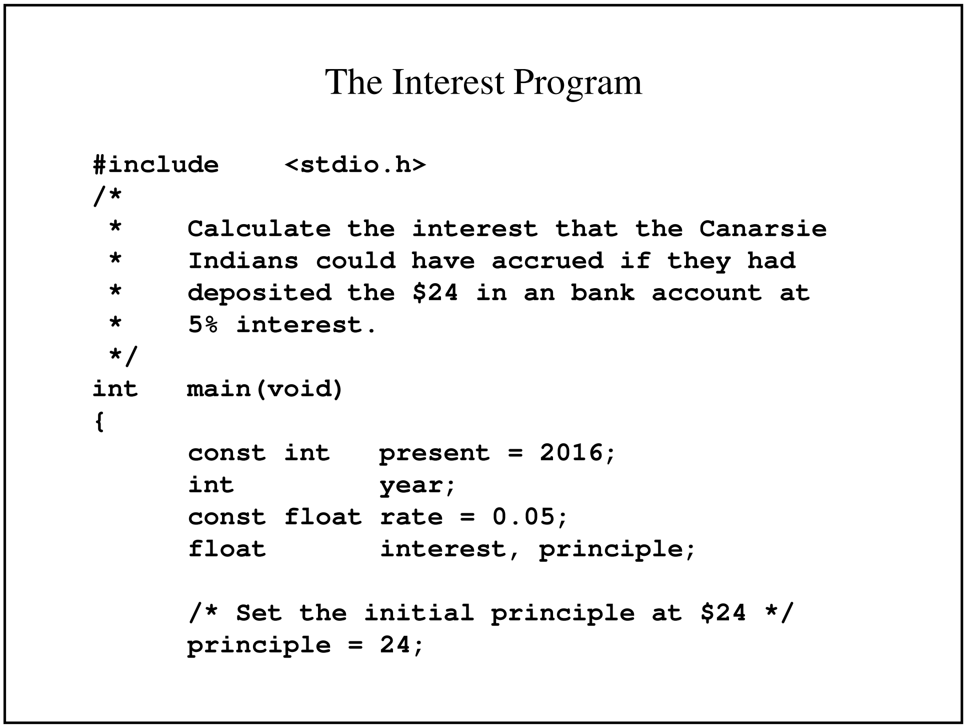


HelloAgain.cpp

```
#include    <stdio.h>

/*
 * Hello again -  this is a better way to write
 *                 "Hello, again" five times
 */
int   main(void)
{     int   i;
      for (i = 1;  i <= 5;  i++)
            printf("Hello, again\n");

      return(0);
}
```



The Interest Program

```
#include    <stdio.h>
/*
 *    Calculate the interest that the Canarsie
 *    Indians could have accrued if they had
 *    deposited the $24 in an bank account at
 *    5% interest.
 */
int   main(void)
{
      const int   present = 2016;
      int         year;
      const float rate = 0.05;
      float       interest, principle;

      /* Set the initial principle at $24 */
      principle = 24;
      /*
       * For every year since 1625, add 5% interest
       * to the principle and print out
       *    the principle
       */
      for  (year = 1625;  year < present;  year++)    {
            interest = rate * principle;
            principle = principle + interest;

            printf("year = %d\tprinciple = %f\n",
                  year, principle);
      }
      return(0);
}
```

Output from the Compound Interest Program

- What will our output look like?

```
year = 1625 principle = 25.200001
year = 1626 principle = 26.460001
year = 1627 principle = 27.783001
year = 1628 principle = 29.172152
      … … … … …
year = 2011     principle = 3806008832.000000
year = 2012     principle = 3996309248.000000
year = 2013     principle = 4196124672.000000
year = 2014     principle = 4405931008.000000
year = 2015     principle = 4626227712.000000
```

- This does not look the way we expect monetary amounts to be written!

%d and **%f**

- The specifiers **%d** and **%f** allow a programmer to specify how many spaces a number will occupy and (in the case of float values) how many decimal places will be used.
- **%*n*d** will use at least n spaces to display the integer value in decimal (base 10) format.
- **%*w.d*f** will use at least w spaces to display the value and will have exactly d decimal places.

Changing the width

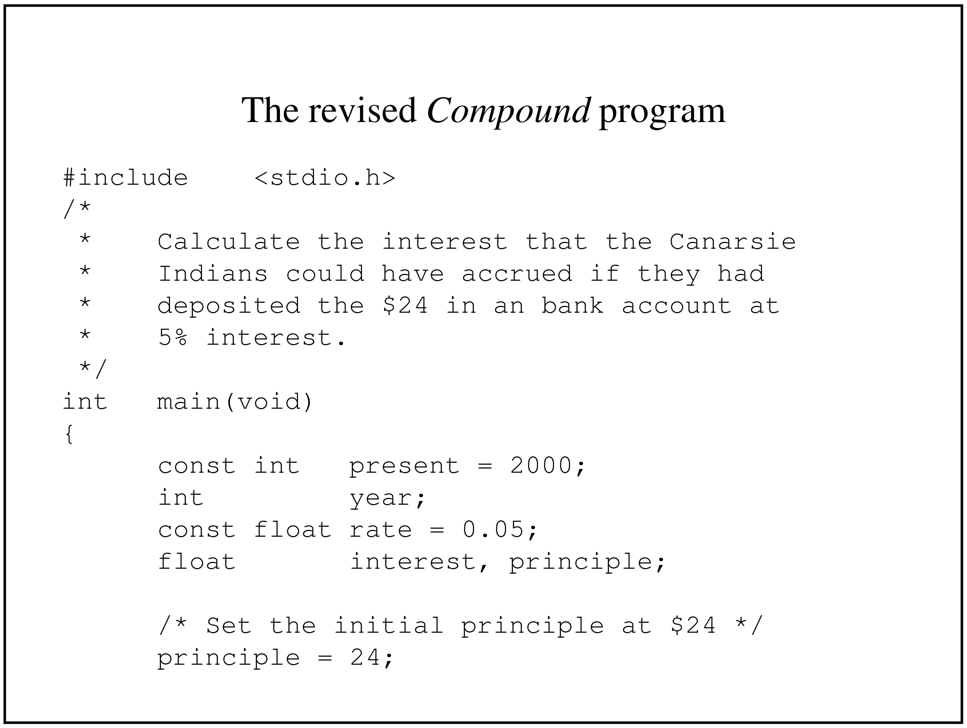
Number	Formatting	Print as:
182	**%2d**	182
182	**%3d**	182
182	**%5d**	``182
182	**%7d**	````182
-182	**%4d**	-182
-182	**%5d**	`-182
-182	**%7d**	```-182

Changing the width (continued)

Number	Formatting	Print as:
23	**%1d**	23
23	**%2d**	23
23	**%6d**	….23
23	**%8d**	……23
11023	**%4d**	11023
11023	**%6d**	.11023
-11023	**%6d**	-11023
-11023	**%10d**	…..-11023

Changing The Precision

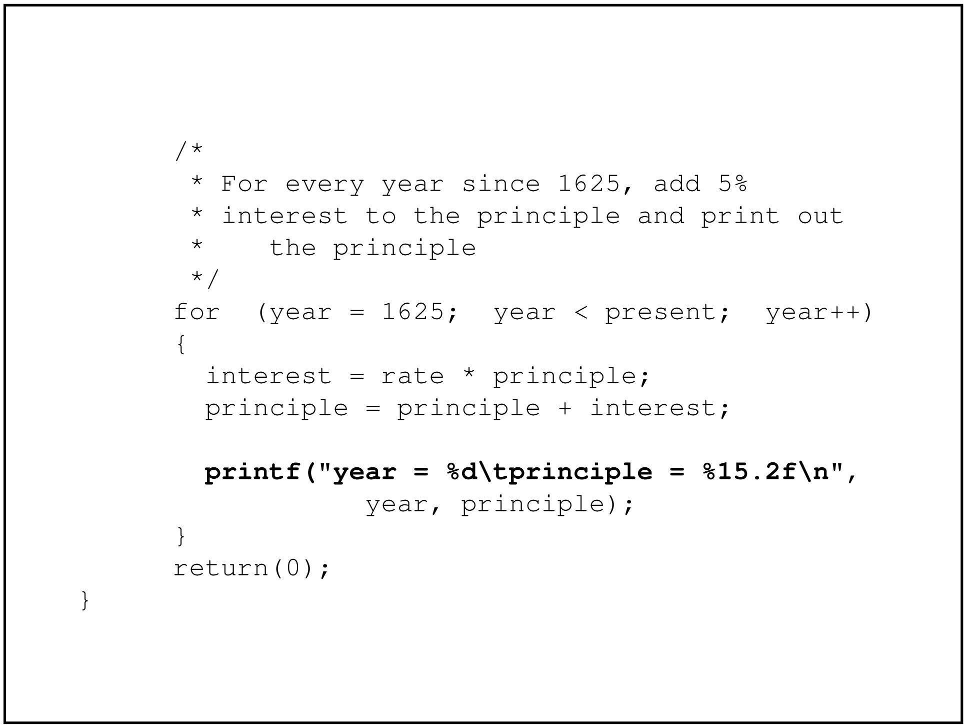
Number	Formatting	Prints as:
2.718281828	**%8.5f**	**`2.71828**
2.718281828	**%8.3f**	**```2.718**
2.718281828	**%8.2f**	**````2.72**
2.718281828	**%8.0f**	**````````3**
2.718281828	**%13.11f**	**2.71828182800**
2.718281828	**%13.12f**	**2.718281828000**



The revised *Compound* program

```
#include    <stdio.h>
/*
 *    Calculate the interest that the Canarsie
 *    Indians could have accrued if they had
 *    deposited the $24 in an bank account at
 *    5% interest.
 */
int   main(void)
{
      const int   present = 2000;
      int         year;
      const float rate = 0.05;
      float       interest, principle;

      /* Set the initial principle at $24 */
      principle = 24;
```



```
       /*
        * For every year since 1625, add 5%
        * interest to the principle and print out
        *    the principle
        */
       for  (year = 1625;  year < present;  year++)
       {
         interest = rate * principle;
         principle = principle + interest;

         printf("year = %d\tprinciple = %15.2f\n",
                   year, principle);
       }
       return(0);
 }
```

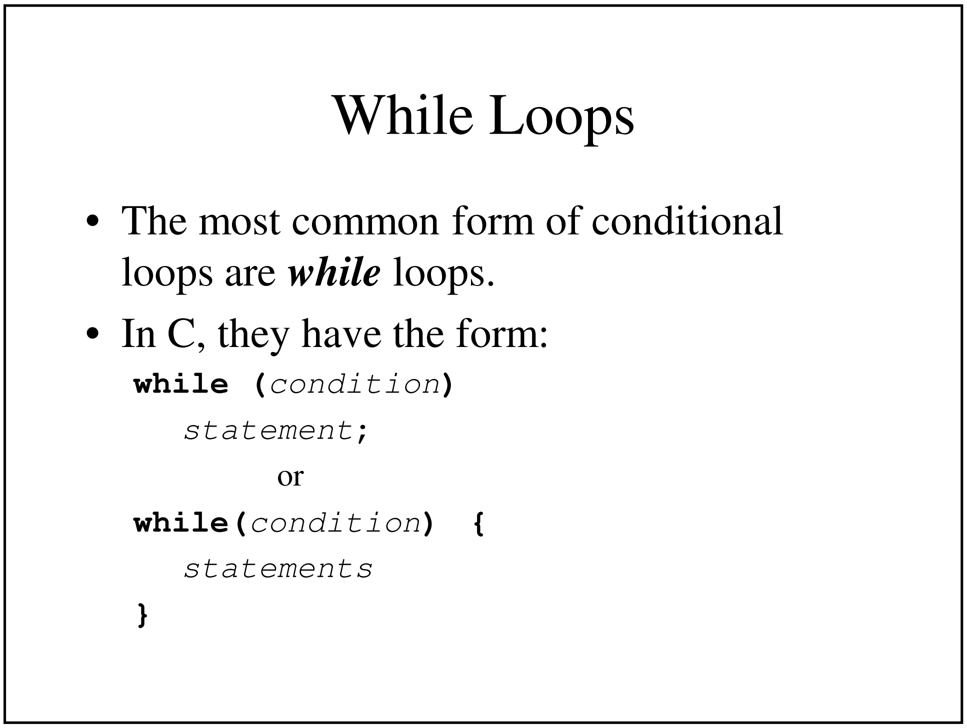
```
      for  (year = 1625;  year < present;  year++)    {
            interest = rate * principle;
            principle = principle + interest;

            printf("year = %d\tprinciple = %15.2f\n",
                  year, principle);
      }
      return(0);
}
```

The output from the Revised Compound Program

Our output now looks like this:

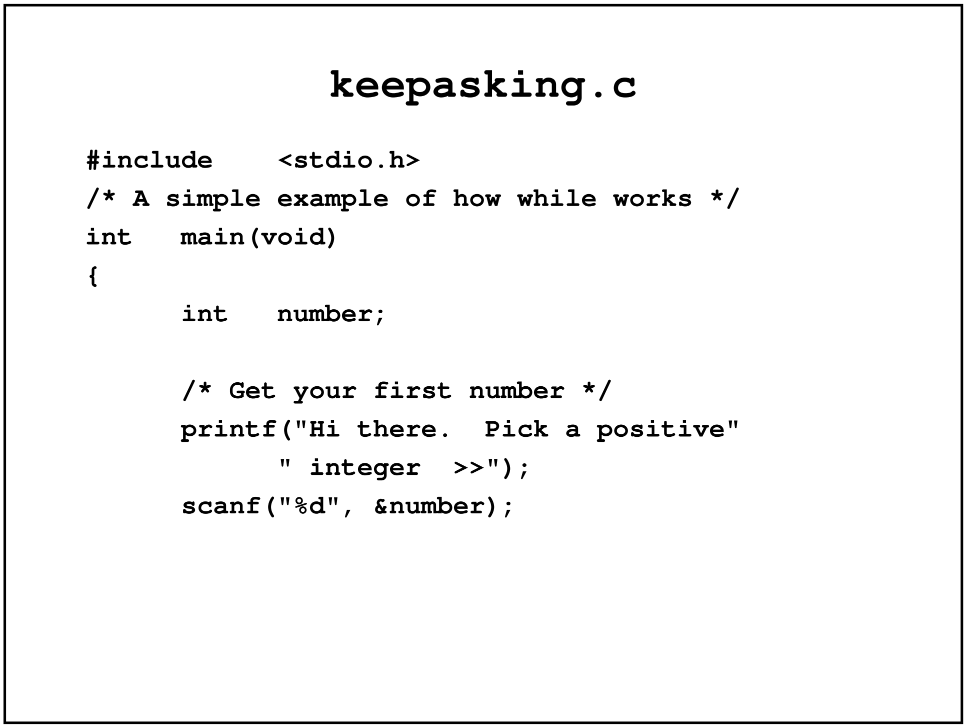
```
year = 1625       principle =          25.20
year = 1626       principle =          26.46
year = 1627       principle =          37.78
year = 1628       principle =          29.17
… … … … …         … … … … … … … … … …
year = 1996       principle =  1830755328.00
year = 1997       principle =  1922293120.00
year = 1998       principle =  2018407808.00
year = 1999       principle =  2119328256.00
```



While Loops

- The most common form of conditional loops are ***while*** loops.
- In C, they have the form:

```
while (condition)
   statement;
         or
while(condition)  {
   statements
}
```



keepasking.c

```
#include    <stdio.h>
/* A simple example of how while works */
int   main(void)
{
      int   number;

      /* Get your first number */
      printf("Hi there.  Pick a positive"
            " integer  >>");
      scanf("%d", &number);
      /* Keep reading number as long as
                  they are positive */
      while (number > 0)      {
            printf("Pick another positive"
                  " integer>>");
            scanf("%d", &number);
      }
      printf("%d is not a positive integer\n",
             number);
      return(0);
}
```

Magic Number Problem

- The magic number game involves guessing a number and with each wrong guess, the player is told “too high” or “ too low”. The goal is to guess the number in the smallest number of tries.
- We need a method for having the computer pick a number at random for the player to guess.
- We will need to learn about how to use “library functions” to provide us with the magic number.

do.. while loops

- You may have noticed that we asked the user twice for same information - the number (s)he is guessing.
- Some loops really require that the condition be at the end - not at the beginning.
- In Java, we have the do.. while loop, whose syntax is:

```
do      {
    statement(s)
 }  (condition)
```

The Magic Number Program

- The main loop in the magic number program is:

```
do{
  /* Let the user make a guess */
  printf("Guess: ");
  scanf("%d", &guess);

  /* Let the user make another guess */
  if (guess > magic)
    printf(".. Wrong .. Too high\n\n");
  else if (guess < magic
    printf(".. Wrong .. Too low\N\n");
  tries++;
} while (guess != magic);
```

`exit()`

- **`exit()`** allows the user to let a program terminate if the program detects an unrecoverable error.
- The statement

  ```
  #include <stdlib.h>
  ```

  has to be included.
- A non-zero status value should be returned when the program terminates abnormally.