Introducing C

1. Please get logged in.

2. Open a new terminal window Applications/Accessories/Terminal

3. Open a web browser Applications/Internet/Iceweasel Web Browser

4. Go to http://www.cse.msu.edu/~cse251

5. Open Step 2: Introducing C

Hello World

```
#include <stdio.h>

/*
 * This is my first program in CSE 251
 */

int main()
{
   printf("Hello, and welcome to CSE 251!!!\n");
}
```

Hello World

```
#include <stdio.h>

/*
 * This is my first progr
 */

int main()
{
   printf("Hello, and we
}
```

#include statement

Includes other source code into your program. One use is to tell your program about libraries you are using. We are using the Standard Input/Output library right now: stdio. The file stdio.h is the header that defines the Standard I/O Library.

Hello World

```
#include <stdio.h>

/*
 * This is my first program in CSE 251
 */

int main()
{
   printf("Hello, and w
}
```

Comment

A Comment is a note to yourself. In C we start a comment with /* and end it with */. Everything in between is ignored.

Programs can be cryptic and hard to read. You need to remind yourself what you were thinking when you wrote something. That's what comments are for.

Hello World

The main function

```
#include <stdio.h>

/*
 * This is my first progra
 */

int main()
{
   printf("Hello, and welcome to CSE 251!!!\n");
}
```

main() is the starting point for a C program. It's where the computer begins execution. Note the *syntax*: the code is contained in curly braces.

Hello World

A printf function call

```
#include <stdio.h>

/*
 * This is my first progr
 */

int main()
{
   printf("Hello, and welcome to CSE 251!!!\n");
}
```

printf() is a library function that prints. It's the main way we will get output from our C programs.

printf statement in more detail

A function call (what we are doing)
Always a name followed by (, optional arguments,), and ;.

A string – Content between " and " is our way of telling the computer about text we want to print.

```
{
    printf("Hello, and welcome to CSE 251!!!\n");
}
```

\n means “newline”.

C statements end with a ;

Escape characters

Characters that are hard to express:

- \n newline
- \t tab
- \’ print a single quote
- \\ print a backslash
- many others

Variable Declarations

Here is how C deals with memory:

- Imagine the system memory as a nice, flat stretch of beach
- You want a variable, you need to dig a hole in the sand and dump the value in
- How big a hole?

The holes in C are very specialized. Once dug, they can only hold one type of thing. Our hole might hold integers, floating point values, or characters, for example.

Declare variable before use

- When you declare a variable, you are telling the compiler the kind of value the variable may hold (its type)
- You **cannot** change the type of value a variable can hold once declared (well, pretty much anyway)
- In fact, *everything needs a type in C* and it must be declared before use!

Common types, “regular” C

- int : an integer, usually 4 bytes
- float: float, usually 4 bytes
- double : float, usually 8 bytes
- char : single char, value in *single quotes*

Must declare before use

- Every variable must be declared before it can be used (its type must be indicated)
- Syntax:

(optional)

<variable_type> <variable_name> [=<initial_value>];

- Example: int length;

Must declare before use

- Every variable must be declared before it can be used (its type must be indicated)
- Syntax:

(optional)

<variable_type> <variable_name> [=<initial_value>] ;

- Example: double width = 10;

Rules for Variable Names

- Must begin with a letter
- Any combination of letters, digits and underscore
- Up to 31 characters long
- Cannot match with a C keyword
  - E.g., int ~~int~~; int ~~long~~;

Some C Keywords

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Some Variable Declarations

```
int numLots = 7;
double width1;
double width2;
float carHP = 420.7;
int a, b, c;
```

printf: a new feature

You can use printf to output variables as well as strings. Put a "descriptor" in the string you print:

```
int numLots = 10;
double totalArea = 100;
printf("There are %d lots\n", numLots);
printf("The %d lots have a total area of %f\n", numLots, totalArea);
```

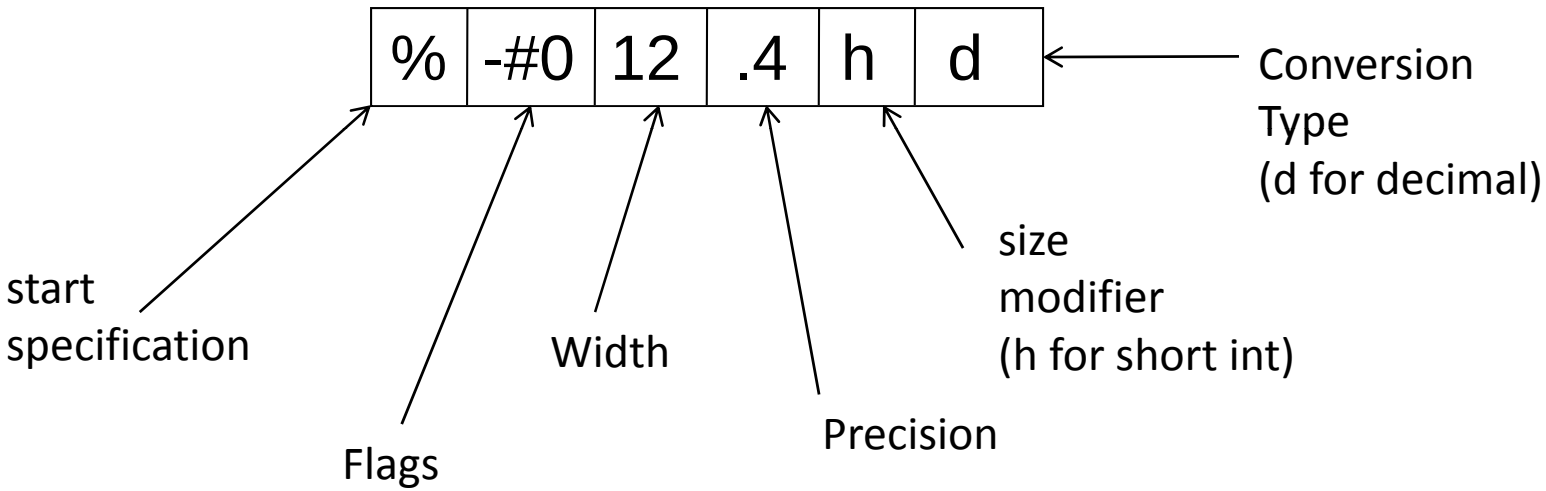
Follow the string with variables.
Each *descriptor* is replaced with a variable's value.

Many descriptors

- %s : string
- %d : decimal (integer)
- %e : floating point exponent
- %f : floating point decimal
- %u : unsigned integer
- and others

Full Format string

- The format string contains a set of format descriptors that describe how an object is to be printed



Examples

- printf("%f\n",M_PI);
  - 3.141593
- printf("%.4f\n",M_PI);
  - 3.1416 (4 decimal points of precision, with rounding)
- printf("%10.2f\n",M_PI);
  - 3.14 (10 spaces total including the number and the decimal point)
- printf("%-10.2f is PI\n",M_PI);
  - 3.14 is PI (10 spaces, but left justified)

Examples

```
int numLots = 10;
double totalArea = 100.7883;
float current = 22.7;

printf("There are %d lots with a total area of %f\n", numLots, totalArea);
/* Output: There are 10 lots with a total area of 100.788300 */

printf("The area is %6.2f\n", totalArea);
/* Output: The area is 100.79 */

printf("The meter reads %.1f amps\n", current);
/* Output: The meter reads 22.7 amps */
```

Expression

An expression is written in pretty standard mathematical notation:

```
wins + losses
width * height
area / width
x * y * z – q
M_PI * 2

printf("We played %d games\n", wins + losses);
```

We can put expressions where we have been putting variables.

Types determine results

- For integers: +,-,*,/ all yield integers. Thus division can lead to truncation (2/3 has value 0). % gives the remainder

- For floats: +,-,*,/ all work as advertised. No remainder.

Mixed computation

- As with most languages, C expects to work with like types. 1 + 1, 3.14 + 4.56
- When mixing, errors are common except where C can "help"
- It will promote a value to a more "detailed" type when required
- 1 + 3.14 yields a float (1 promoted to 1.0)

Assignment

An assignment puts a value in a variable. It is of the form:

variable = expression ;

Do everything on the right hand side, get a value.
Dump the value into the variable.

```
double area, circumference;
area = width * height;
circumference = radius * 2 * M_PI;
```

Declare first, assign later!
Don't forget the semicolon.

Expression and Assignment Examples

```
double volume, diameter, hypot;
int games;

volume = radius * radius * M_PI;
diameter = radius * 2;
games = wins + losses;
hypot = sqrt(near * near + pow(far, 2));
```

To use M_PI or math functions: #include <math.h> and compile with –lm switch.

scanf

```
printf("Please enter the yards of pipe used: ");
scanf(”%f",&yardsOfPipe);
```

Scanf is an input routine

- Useful for reading in string input and doing conversion to the correct type, all at once
- Syntax is “kind of like” printf
- Note the use of the & operator!!!

Basic form

- To understand input, it is probably better to start with an example.

```
scanf("%d, %f", &myInt, &myFloat);
```

int float

…is waiting for input of the **exact** form

25, 3.14159

Use %f for float and %lf for double. (That's the letter l, not the number 1)

format string the same

- What is typed in the format string is the same as what the input expects, in this case:
  - a decimal number
  - a comma
  - a floating point number

The &

- Hard to explain at the moment, but any variable that gets read in needs that ampersand character
- It is the address of the variable
- more on that later

scanf examples

```
printf("Input the radius: ");
scanf("%lf", &radius);

printf("Input the height: ");
scanf("%lf", &height);

printf("Input the number of cylinders: ");
scanf("%d", &numCylinders);
```