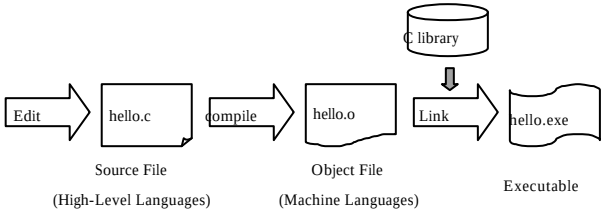
Introduction to C Programming

Creating Programs



Simple Program

```
#include <stdio.h>

/* My first C program */

int main (void)
{
   printf ("Hello World!\n");
   return 0;
}
```

Pre-processor / Linker

Comments

main() means "start here"

Library command

Return 0 from main means our program finished without errors

Brackets define code blocks

Basic C Programming Concepts

- Comment
  - To explain what the program is for and why it is the way it is.
  - Text surrounded by **/*** and ***/** is ignored by computer
- Return
  - A way to exit a function, return 0 often means that the program terminated normally
- Statement
  - A line of code that tells computer to perform some action. Always ended with a semicolon **;** .

Basic C Programming Concepts

- Function
  - A module often consisting of a set of statements
- Linker
  - When a function is called, linker locates it in the library. If function name is misspelled, the linker will produce an error because it will not be able to find function in the library
  - **<stdio.h>** allows standard input/output operations

Common Keywords of C

- Flow control (6) – **`if, else, return, switch, case, default`**
- Loops (5) – **`for, do, while, break, continue`**
- Common types (5) – **`int, float, double, char, void`**
- Structures (3) – **`struct, typedef, union`**
- Counting and sizing things (2) – **`enum, sizeof`**
- Rare but still useful types (7) – **`extern, signed, unsigned, long, short, static, const`**
- Evil keywords which we avoid (1) – **`goto`**
- Wierdies (3) – **`auto, register, volatile`**

Printing a Line of Text

```
/* 1A
   A first program in C */
#include <stdio.h>
#include <conio.h>
int main( void )
{
   printf( "Welcome to C!\n" );
   getch();
   return 0;
}
```


Output Result

Explanation

- **#include <stdio.h>**
  - Preprocessor directive
  - Tells computer to load contents of a certain file
  - <stdio.h> allows standard input/output operations

Explanation

- **int main()**
  - C programs contain one or more functions, exactly one of which must be main
  - Parenthesis used to indicate a function
  - int means that main "returns" an integer value
  - Braces (**{** and **}**) indicate a block
  - The bodies of all functions must be contained in braces

Explanation

- **printf("Welcome to C!\n");**
  - Instructs computer to perform an action
  - Specifically, prints the string of characters within quotes **“ ”**
  - Entire line called a statement
  - All statements must end with a semicolon **;**
  - Escape character \
  - Indicates that printf should do something out of the ordinary
  - **\n** is the newline character

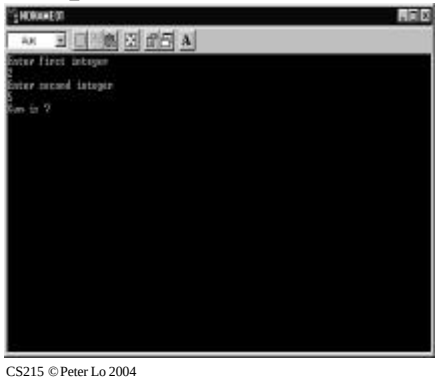
Explanation

- **return 0;**
  - A way to exit a function
  - return 0, in this case, means that the program terminated normally
- **}** (Right brace)
  - Indicates end of main has been reached
- Linker
  - When a function is called, linker locates it in the library
  - If function name is misspelled, the linker will produce an error because it will not be able to find function in the library

Adding Two Integers

```
/* 1B Adding 2 integers */
#include <stdio.h>
#include <conio.h>
int main()
{
  int integer1, integer2, sum;      /* declaration */
  printf( "Enter first integer\n" ); /* prompt */
  scanf( "%d", &integer1 );         /* read an integer */
  printf( "Enter second integer\n" ); /* prompt */
  scanf( "%d", &integer2 );         /* read an integer */
  sum = integer1 + integer2;        /* assignment of sum */
  printf( "Sum is %d\n", sum );     /* print sum */
  getch();
  return 0;  /* indicate that program ended successfully */
}
```


Output Result



Explanation

- **int integer1, integer2, sum;**
  - Declaration of variables
    - Variables: locations in memory where a value can be stored
  - **int** means the variables can hold integers (-1, 3, 0, 47)
  - Variable names (identifiers)
    - *integer1, integer2, sum* are Identifiers: consist of alphanumeric (can't begin with digit) and underscores _ . Case sensitive!
  - Declarations appear before executable statements
    - If an executable statement references and undeclared variable it will produce a syntax (compiler) error

Explanation

- **scanf("%d", &integer1);**
  - Obtains a value from the user
    - **scanf** uses standard input (usually keyboard)
  - This **scanf** statement has two arguments
    - **%d** - indicates data should be a decimal integer
    - **&integer1** - location in memory to store variable
    - **&** is confusing in beginning – for now, just remember to include it with the variable name in **scanf** statements
  - When executing the program the user responds to the **scanf** statement by typing in a number, then pressing the enter (return) key

Explanation

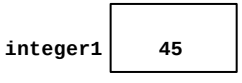
- = (assignment operator)
  - Assigns a value to a variable
  - Is a binary operator (has two operands)
    - *sum = variable1 + variable2;*
    - *sum gets variable1 + variable2;*
  - Variable receiving value on left

Explanation

- **printf("Sum is %d\n", sum);**
  - Similar to **scanf**
    - **%d** means decimal integer will be printed
    - **sum** specifies what integer will be printed
  - Calculations can be performed inside **printf** statements
    - **printf("Sum is %d\n", integer1 + integer2);**

Memory Concepts

- Variables
  - Variable names correspond to locations in the computer's memory
  - Every variable has a name, a type, a size and a value
  - Whenever a new value is placed into a variable (through **scanf**, for example), it replaces (and destroys) the previous value
  - Reading variables from memory does not change them



Types of variable

- We must *declare* the *type* of every variable we use in C.
- Every variable has a *type* (e.g. `int`) and a *name*.
- We already saw `int`, `double` and `float`.
- This prevents some bugs caused by spelling errors (misspelling variable names).
- Declarations of types should always be together at the top of main or a function.
- Other types are `char`, `signed`, `unsigned`, `long`, `short` and `const`.

Naming variables

- Variables in C can be given any name made from numbers, letters and underlines which is not a keyword and does not begin with a number.
- A good name for your variables is important

```
int a,b;
double d;
/* This is
a bit cryptic */
```

```
int start_time;
int no_students;
double course_mark;
/* This is a bit better */
```

- Ideally, a comment with each variable name helps people know what they do.

Arithmetic Operators

- To form expressions
- Many statements are merely expressions.
- Normal order of operations is followed.
- Parentheses can be used.

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder)
++	Increment
--	Decrement

Arithmetic Operators

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y	`x / y`
Modulus	%	$r \bmod s$	`r % s`

Operator Precedence

Operator(s)	Operation(s)	Order of evaluation (precedence)
`( )`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
`*`, `/`, or `%`	Multiplication, Division, Modulus	Evaluated second. If there are several, they are evaluated left to right.
`+` or `-`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Example

- Integer division truncates remainder
  - 7 / 5 evaluates to 1
- Modulus operator (%) returns the remainder
  - 7 % 5 evaluates to 2
- To find the average of three variables a, b and c
  - Do not use: a + b + c / 3
  - Use: (a + b + c) / 3

Printing Integers

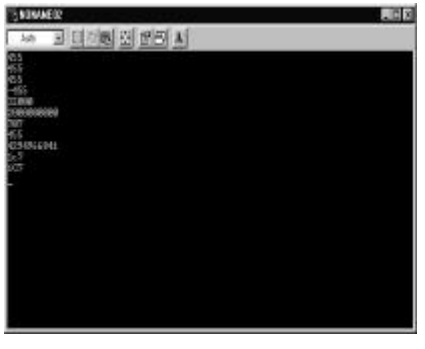
- Whole number (no decimal point): 25, 0, -9
- Positive, negative, or zero
- Only minus sign prints by default

Conversion Specifier	Description
`d`	Display a signed decimal integer.
`i`	Display a signed decimal integer. (*Note:* The `i` and `d` specifiers are different when used with `scanf`.)
`o`	Display an unsigned octal integer.
`u`	Display an unsigned decimal integer.
`x or X`	Display an unsigned hexadecimal integer. `X` causes the digits `0-9` and the letters `A-F` to be displayed and `x` causes the digits `0-9` and `a-f` to be displayed.
`h` or `l` (letter `l`)	Place before any integer conversion specifier to indicate that a `short` or `long` integer is displayed respectively. Letters `h` and `l` are more precisely called *length modifiers*.

Example

```
/* 1 C: Using the integer conversion specifiers */
#include<stdio.h>
#include<conio.h>
int main()
{
   printf( "%d\n", 455 );
   printf( "%i\n", 455 );  /* i same as d in printf */
   printf( "%d\n", +455 );
   printf( "%d\n", -455 );
   printf( "%hd\n", 32000 );
   printf( "%ld\n", 2000000000 );
   printf( "%o\n", 455 );
   printf( "%u\n", 455 );
   printf( "%u\n", -455 );
   printf( "%x\n", 455 );
   printf( "%X\n", 455 );
   getch ();
   return 0;
}
```


Output Result



Printing Floating-Point Numbers

- Have a decimal point (33.5)
- Exponential notation (computer's version of scientific notation)
  - 150.3 is 1.503 x 10^2 in scientific
  - 150.3 is 1.503E+02 in exponential (E stands for exponent)
  - use **e** or **E**
- **f** – print floating point with at least one digit to left of decimal
- **g** (or **G**) - prints in for e with no trailing zeros (1.2300 becomes 1.23)
- Use exponential if exponent less than -4, or greater than or equal to precision (6 digits by default)

Example

```
/* 1D: Printing floating-point numbers with floating-point conversion specifiers */

#include <stdio.h>
#include <conio.h>
int main()
{
   printf( "%e\n", 1234567.89 );
   printf( "%e\n", +1234567.89 );
   printf( "%e\n", -1234567.89 );
   printf( "%E\n", 1234567.89 );
   printf( "%f\n", 1234567.89 );
   printf( "%g\n", 1234567.89 );
   printf( "%G\n", 1234567.89 );
   getch();
   return 0;
}
```


Output Result

Printing Strings and Characters

- **c**
  - Prints char argument
  - Cannot be used to print the first character of a string
- **s**
  - Requires a pointer to char as an argument
  - Prints characters until **NULL** ('**\0**') encountered
  - Cannot print a char argument
- Remember
  - Single quotes for character constants (**'z'**)
  - Double quotes for strings "**z**" (which actually contains two characters, '**z**' and '**\0**')

Printing Strings and Characters

```
/* 1E: Printing strings and characters */
#include <stdio.h>
#include <conio.h>
int main()
{
   char character = 'A';
   char string[] = "This is a string";
   const char *stringPtr  = "This is also a string";
   printf( "%c\n", character );
   printf( "%s\n", "This is a string" );
   printf( "%s\n", string );
   printf( "%s\n", stringPtr  );
   getch();
   return 0;
}
```


Output Result

A Complex Example

```
/* 1 F: Using precision while printing integers, floating-point numbers, and strings */
#include<stdio.h>
#include<conio.h>
int main()
{
   int i = 873;
   double f = 123.94536;
   char s[] = "Happy Birthday";

   printf( "Using precision for integers\n" );
   printf( "\t%.4d\n\t%.9d\n\n", i, i );
   printf( "Using precision for floating-point numbers\n" );
   printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );
   printf( "Using precision for strings\n" );
   printf( "\t%.11s\n", s );
   getch ();
   return 0;
}
```


Output Result

Formatting Input with Scanf

Conversion specifier	Description
Integers	
d	Read an optionally signed decimal integer. The corresponding argument is a pointer to integer.
i	Read an optionally signed decimal, octal, or hexadecimal integer. The corresponding argument is a pointer to integer.
o	Read an octal integer. The corresponding argument is a pointer to unsigned integer.
u	Read an unsigned decimal integer. The corresponding argument is a pointer to unsigned integer.
x or **X**	Read a hexadecimal integer. The corresponding argumentis a pointer to unsigned integer.
h or **l**	Place before any of the integer conversion specifiers to indicate that a **short** or **long** integer is to be input.
Floating-point numbers	
e, **E**, **f**, **g** or **G**	Read a floating-point value. The corresponding argument is a pointer to a floating-point variable.
l or **L**	Place before any of the floating-point conversion specifiers to indicate that a **double** or **long double** value is to be input.

Formatting Input with Scanf

Conversion Specifier	Description
Characters and strings	
c	Read a character. The corresponding argument is a pointer to **char**, no null (**'\0'**) is added.
s	Read a string. The corresponding argument is a pointer to an array of type **char** that is large enough to hold the string and a terminating null (**'\0'**) character—which is automatically added.
Scan set	
[scan characters	Scan a string for a set of characters that are stored in an array.
Miscellaneous	
p	Read an address of the same form produced when an address is output with **%p** in a **printf** statement.
n	Store the number of characters input so far in this **scanf**. The corresponding argument is a pointer to integer
%	Skip a percent sign (**%**) in the input.

Example

```
/* 1G: Reading characters and strings */
#include <stdio.h>
#include <conio.h>
int main()
{
   char x, y[ 9 ];

   printf( "Enter a string: " );
   scanf( "%c%s", &x, y );

   printf( "The input was:\n" );
   printf( "the character \"%c\" ", x );
   printf( "and the string \"%s\"\n", y );
   getch();
   return 0;
}
```


Output Result

Decision Making: Equality and Relational Operators

Standard algebraic equality operator or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality Operators			
=	==	x == y	x is equal to y
not =	! =	x != y	x is not equal to y
Relational Operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
>=	>=	x >= y	x is greater than or equal to y
<=	<=	x <= y	x is less than or equal to y

Example

```
/* 1H:  Using if statements, relational  operators, and equality operators */
#include <stdio.h>
#include <conio.h>
main()
{
   int num1, num2;

   printf( "Enter two integers, and I will tell you\n" );
   printf( "the relationships they satisfy: " );
   scanf( "%d%d", &num1, &num2 );   /* read two integers */

   if ( num1 == num2 )
      printf (  "%d is equal to %d\n", num1, num2 );

   if ( num1 != num2 )
      printf (  " %d is not equal to %d\n ", num1, num2 );
```


Example (cont')

```
   if ( num1 < num2 )
      printf ( "%d is less than %d\n", num1, num2 );

   if ( num1 > num2 )
      printf ( "%d is greater than %d\n", num1, num2 );

   if ( num1 <= num2 )
      printf ( "%d is less than or equal to %d\n",
            num1, num2 );

   if ( num1 >= num2 )
      printf ( "%d is greater than or equal to %d\n",
            num1, num2 );
   getch();
   return 0;   /* indicate program ended successfully */
}
```


Output Result