Computer Science 1510

Lecture 21

Lecture Outline

- Introduction to C programming
- Data types and variables
- Arithmetic, relational, and logical operators

The C Programming Language

- C was originally developed as a system programming language for the UNIX operating system.

- However, it has since become one of the most common programming languages for many different applications.

- In this course we will follow a similar route through C programming as we did for Fortran programming.

- Many of the programming ideas are the same in both languages.

- For example, variables, data structures, assignment statements, control constructs (selection and repetition), subprograms, I/O, and dynamic allocation are all present both languages

Your First C Program

- The following simple C program will be used to illustrate the structure of a C program.

```
#include <stdio.h>
int main(int argc, char *argv[])
{
   printf("Hello, world\n");
   return 0;
}
```

- The first line above is a pre-processor directive. We will see more about the C pre-processor soon.

- Each C program must have a `main` function, the beginning of which is indicated by the line

```
int main(int argc, char *argv[])
```

  This is similar to the main `PROGRAM` in Fortran.

- The contents of the main function are enclosed in curly brackets, also known as braces.

Your First C Program

- The `printf` line is used to print the string "Hello, world" to the screen. We will examine the syntax of the `printf` statement in more detail.

- It is important to note that C is case sensitive. Variables `sum` and `Sum` are different variables in C.

- Thus, unlike Fortran programs, which can be written in any case (to be converted to uppercase during compilation), keywords and standard library function calls must be written in lowercase in C.

- Each statement in C must be terminated with a semicolon. Thus, a single statement can run over multiple lines.

The C Library

- Unlike Fortran, C has very few built-in functions. It instead uses a collection of library functions that are accessible via *header files*.

- Header files (.h extension) define and include information about what is contained in a given library.

- For example, the `stdio` library contains all of the functions that pertain to input and output.

- To use the functions contained within a given library, we must *include* that library by using a `#include` statement.

- For example, in our first program we had,

  ```
  #include <stdio.h>
  ```

  Note that the name of the library is contained within angle brackets (indicating that it is found in the standard C library directory), and that `#include` lines do not have semicolons at the end.

The C Compilation Process

- The compilation process in C is a little different than what we have seen in Fortran.

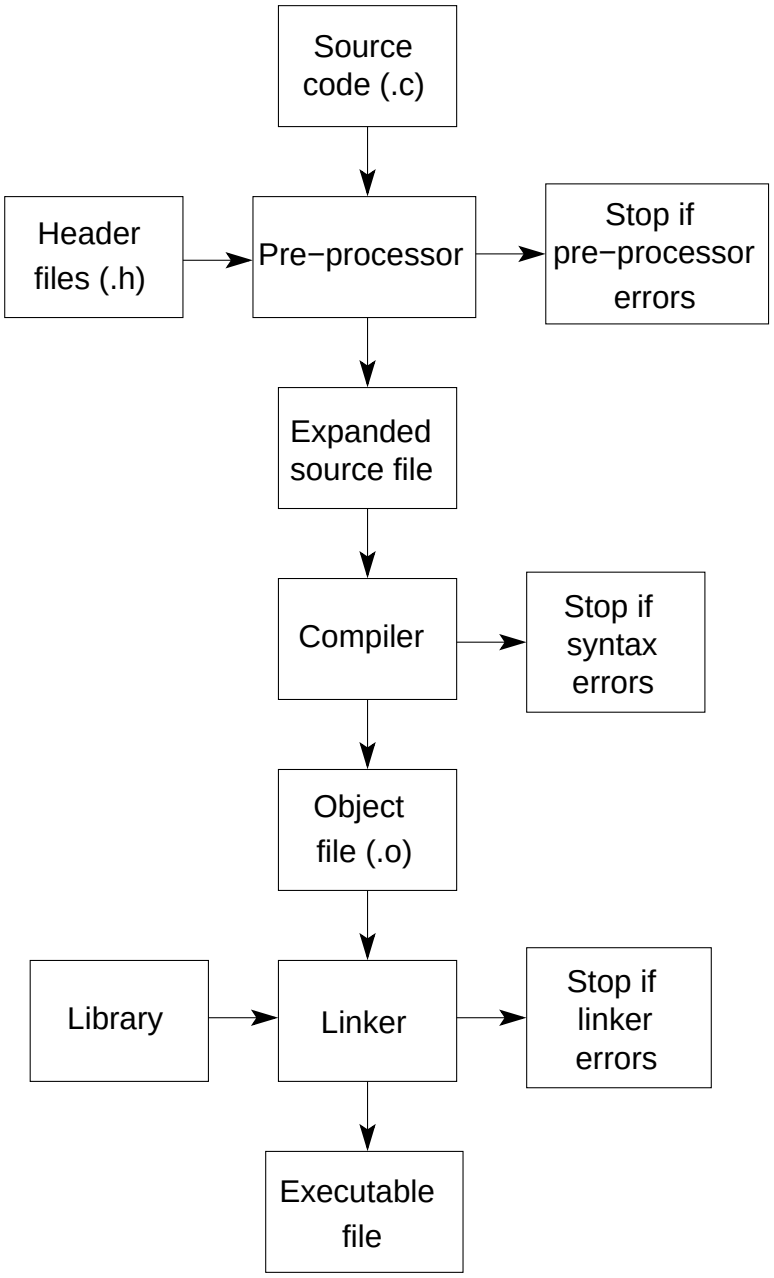
- However, the goal of the compiler is the same: to ensure that the source code is free of syntax errors, and to translate the code into machine language.

- The C compilation process includes running through the C pre-processor, whose job is to check for pre-processor directives (such as `#include` statements), and insert the indicated header files into the source code to create an expanded source file.

- For example, in our first C program the pre-processor would recognize the `#include <stdio.h>` line and insert the `stdio.h` header file into the source file.

- This new, expanded source file, is sent to the compiler.

The C Compilation Process



The C Compiler

- We will be using the GNU C compiler, `gcc`.
- For example, to compile a program in source file `myprog.c` we would have,

```
gcc myprog.c
```

- To specify the name of the executable, use the -o flag,

```
gcc -o myprog myprog.c
```

  The executable will be named `myprog`

- To view the results of the pre-processor we can use the -E flag,

```
gcc -E myprog.c > myprog.s
```

  The expanded source file is `myprog.s`.

- To compile to the object file, use the -c flag (defaults to `myprog.o`).

C: Source code comments

- There are two ways to add comments or *comment out* code in C.

  1. If a comment is a single line, a double slash // can be used. For example,
     ```
     // This is a comment.
     ```
     Everything from the double slash to the end of the line is a comment.
  2. The standard C comment uses /* to begin the comment and */ to end the comment. Thus comments can span multiple lines without adding a special symbol to each line. For example,
     ```
     /* This is a comment */
     /*
     This is a
     comment block.
     */
     ```

Output in C: printf

- Syntax:

  ```
  printf("description",variable-list);
  ```

  where `description` is a formatted character string to be printed to the screen.

- The order of the variables in the `variable-list` must match the order in which they are referred to in the `description`.

- This function is similar to `WRITE(*,*)` in Fortran in that it prints to the screen.

- However, `printf` has formatting built into the string, rather than using a separate format statement.

- The `description` can consist of character strings to be printed as is, *escape sequences* for special formatting, and *conversion codes* which indicate the type and format of variables that are to be printed.

Output in C: printf

- For example, in the case of

```
printf("Hello, world\n");
```

  the `\n` is an escape sequence indicating that a newline should occur after printing Hello, world.

- The backslash of escape sequences indicates that the following character signifies formatting information.

- Conversion codes begin with a % symbol, followed by a series of letters and digits similar to that used in Fortran FORMAT statements.

Output in C: printf

- For example,

```
printf("Today is March %d",day);
```

  The `d` indicates that an integer will be printed in that position. The value that will be printed in place of the `%d` is the value contained in the variable `day`.

- The number and type of conversion codes must match the number and type of variables and/or constants in the `variable-list`.

- The more common conversion codes include:

%d	Converts to integer notation.
%c	Converts to a single character.
%s	Converts to single characters until reaching a `\0`.
%f	Converts to signed real number notation.

- Each conversion code takes an optional number after the % (ex. `%5.2f` or `%3d`).

C: Assignment statement

- An assignment statement in C is similar to that in Fortran.

- In its general form we have,

  ```
  variable=expression;
  ```

  where `expression` is some valid arithmetic or logical expression, or function call.

- Note the semicolon, indicating the end of the line.

The main function

- The main function in C has the following form:

```
int main(int argc, char *argv[])
{
/* body of the main function */
   return 0;
}
```

- `main` has two arguments, `argc` and `argv`.

- The first argument is an integer (`argc`) which contains the number of command line arguments passed to the program.

- The second argument is a pointer to an array of pointers to the individual command line arguments. We will revisit these two arguments later.

- The main function returns an integer, as indicated by the `int` before `main`. Thus, a return value is required. Above, we have returned a value of 0, traditionally used to tell the operating system that the program has completed successfully.

Data types in C

- Like Fortran, C has several intrinsic data types. The basic types include:

`char`	Character and/or integer in $[-128, 127]$.
`int`	Integer with range depending on the architecture, often [-2147483648, 2147483647], ie. 32 bits.
`float`	Single precision real numbers in $[-3.40282 \times 10^{38}, 3.40282 \times 10^{38}]$.
`double`	Double precision real numbers in $[-1.79769 \times 10^{308}, 1.79769 \times 10^{308}]$.

- Qualifiers `short` and `long` can also be applied to integer variables (often 16 bits and 32 bits respectively). For example, `short int i`. Often, the `int` is omitted.

- The `long` qualifier can also be used with `double`.

- Qualifiers `signed` and `unsigned` can also be applied to `chars` or any integer. In the `unsigned` case, only positive values can be stored.

Variable declarations

- Syntax:

  ```
  type identifier-list;
  ```

  where `type` is any valid C data type (int, float, etc.), and `identifier-list` is one or more valid identifiers separated by commas.

- Examples:

  ```
  int num1;
  float xval,yval;
  char grade;
  ```

- It is also possible to initialize variables on the declaration line. For example:

  ```
  int num1=0;
  float xval=1.5,yval;
  char grade='A';
  ```

Identifiers in C

- The following rules must be followed when choosing identifiers:

  1. Less than 32 characters.
  2. No white space.
  3. First character must be a letter or an underscore (_).
  4. Variable names are case sensitive (ie. `sum` is not the same as `Sum`).
  5. C keywords cannot be used as variable names.
  6. Can consist of uppercase and lowercase letters (A-Z, a-z), digits (0-9), and an underscore (_).

- The qualifier `const` can be applied to the declaration of a variable to specify that the value is not to be changed. For example,

```
const double pi = 3.141592654;
```

Logical data in C

- Unlike Fortran, C does not have a logical data type.
- Instead, an integer or a character can be used.
- A variable evaluates to false if its value is zero, and true if its value is nonzero.
- For example, we could have the following:

```
#include <stdio.h>
int main(int argc, char *argv[])
{
   int a,b;
   a=0; b=1;
   if (a) printf("a\n");
   if (b) printf("b\n");
   if (-5) printf("c\n");
   return 0;
}
```

  which will print `b` and `c` on separate lines.

Arithmetic expressions

- The following are legal arithmetic operators in C:

`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`%`	Modulus (remainder)
`=`	Assignment

- Note that arithmetic expressions can also be used within function calls. For example,

```
printf("%d\n",a+b);
```

What about exponentiation?

- There is no exponentiation operator in C.
- There is a function called `pow` that is contained in the math library, (accessible via `math.h`).
- For example, to compute a^b we would have `pow(a,b)`.
- We would also require `#include <math.h>` at the beginning of the source file.
- The math library is separate from the standard C library. Therefore, when compiling code that requires functions from the math library we must explicitly link to that library.
- We add the `-lm` flag to our compile command:

```
gcc filename.c -lm
```

The math library

The following is a partial list of functions contained in the math library:

`sin(x)`	sine of x.
`cos(x)`	cosine of x.
`tan(x)`	tangent of x.
`asin(x)`	inverse sine of x.
`acos(x)`	inverse cosine of x.
`atan(x)`	inverse tangent of x.
`sinh(x)`	hyperbolic sine of x.
`cosh(x)`	hyperbolic cosine of x.
`tanh(x)`	hyperbolic tangent of x.
`exp(x)`	exponential function e^x.
`log(x)`	natural logarithm $\ln(x)$.
`log10(x)`	base 10 logarithm $\log_{10}(x)$.
`pow(x,y)`	x^y.
`sqrt(x)`	$\sqrt{x}$.
`fabs(x)`	absolute value $\|x\|$.

Arithmetic expressions

- In C, it is possible to shorten certain expressions. For example, `i++` is equivalent to saying `i=i+1`.

- The following is a list of common short-cut operators:

Expression	Meaning
`x+=y`	$x = x + y$
`x-=y`	$x = x - y$
`x*=y`	$x = x * y$
`x/=y`	$x = x/y$
`x%=y`	$x = x\%y$
`x++`	$x = x + 1$
`x--`	$x = x - 1$

Increment and decrement operators

- The increment and decrement operators (++ and --) can be added as either prefix operators (ie. before the variable), or postfix operators (ie. after the variable).

- Example:

```
#include <stdio.h>

int main(int argc, char *argv[]){
   int x,n;
   n=5;
   x=n++;
   printf("x=%d, n=%d\n",x,n);
   n=5;
   x=++n;
   printf("x=%d, n=%d\n",x,n);
   return 0;
}
```

- Output:

```
x=5, n=6
x=6, n=6
```

Arithmetic expressions and assignment

- In the case of mixed-mode operations, where we have variables of different types on each side of an operator, the variables are converted to the type with the highest precedence.

- The order of precedence from highest to lowest is: double, float, long, int, short, char.

- For example, `double` *op* `int` produces a `double`, where *op* is an arithmetic operator.

Relational and Logical operators

- The following relational and logical operators can be used in C to build a logical expression:

Operator	Meaning
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
&&	And
||	Or

- Recall that, in C, a value of zero is treated as false, while a nonzero value is treated as true.

- The default value for true is 1.

Logical expressions

- The following are examples of valid logical expressions in C:

```
weight > 90
x>=2 && y<4
initial=='a'
a+b!=0 || b+c!=0
```

- An additional operator that can be used in C is the `?`.

- This operator has the following syntax:

```
logical_exp ? if_true : if_false
```

  where `logical_exp` is some logical expression that evaluates to true or false (non-zero or zero), `if_true` is executed if the `logical_exp` is true, while `if_false` is executed if the `logical_exp` is false.

Operator precedence

- The order of precedence for evaluating arithmetic expressions is as follows:

  1. Parentheses from inside outwards.
  2. Function calls.
  3. Unary operators (ex. ++), evaluated right to left.
  4. Multiplication, division, and modulus, evaluated left to right.
  5. Addition and subtraction, evaluated left to right.
  6. Relational operators <, <=, >, and >=, evaluated left to right.
  7. Relational operators == and !=, evaluated left to right.
  8. Logical operator &&, evaluated left to right.
  9. Logical operator ||, evaluated left to right.
  10. Assignment operators (ex. =, +=, -=), evaluated right to left.