module1

gar

Outline

Introduction to C Programming

Extras – Introduction to Unix-like OSes

Useful commands

- `man`: This is a command to know about other commands. E.g. `man ls` gives the manual of the `ls` command.
- `ls` : List contents in current directory (folder)
- `cal`: Display the calendar
- `rm`: Remove (delete) any file
- `mv`: Move a file from one location to another, also useful for renaming files
- and many more – explore the directories `/bin` and `/usr/bin`

Extras – Getting started with C

- C is a general-purpose programming language
- Used mainly for implementing Operating Systems, and application softwares for computers/embedded systems
- Developed by Dennis Ritchie and used to re-implement the Unix OS

Basic Structure of a C Program

```
Preprocessor Directives

Global Declarations

main()
{
  Local Declarations

  Statements
}

User defined functions
```

Basic Structure of a C Program

- Print the words "Hello, world!"

```c
#include <stdio.h> /* include information about the
                      standard input/output library */
int main() /* define function named main
              and returns an integer */
{
    printf("Hello, world!\n"); /* prints the words,
                                  \n is the newline character */
    return 0;                 /* The value 0 is returned
                                 to the OS on completion    */
}
```

Extras – Getting started with C

Code compilation:

- Compilation is a process of converting the source code to machine code
  - i.e. converting from a human readable code to a code which machine understands
  - The output will be a binary file (0's and 1's)
  - They encode instructions regarding the action to be performed by the CPU (e.g. copy from one location to another, multiply two numbers etc.)

Compiling and executing in linux:

- `gcc hello.c`
- `./a.out`
- Program consists of *functions* and *variables*
  - functions contain *statements* that specify operations to be done
  - variables store the values used during the operation
- The main function is the beginning of the execution
- That may call other functions

Getting started

- `"Hello, world!\n"`

  is called a string constant
- `printf("Hello, world!\n");`

  would cause an error

Extras – Comments and Programming Style

- Comments are statements which describe the function of the code
  - A single line comment (C++ style, not allowed in ANSI C standard) – `//`
  - A multiline comment – `/* ... */`
  - `/*` and `*/` are called the comment delimenters
- All the characters within the comments are ignored during compilation
- Use an editor which supports
  - syntax highlighting
  - automatic indentation
  - autocompletion
- All these features will minimize the chances of errors and bugs
- e.g. emacs, geany, vi etc.

Variables and arithmetic expressions

- Declaration statement declares a variable to be used in the program
  - E.g. `int num;`
- Assignment statement
  - E.g. `int num = 4;`
  - Value of 4 is assigned to a variable called `num`
- Increment the value of `num` by 2
  - `num = num + 2;`
  - Don't think of it as a mathematical equation
  - It means whatever value `num` contains, 2 will be added to it, and then stored back in `num`
- To compute square of a number
  - `sqnum = num * num;`

Variable Names

- Consists of letters (underscores allowed) and digits, must begin with a letter
- Usually lower case letters are used for variable names and all upper case for symbolic constants
- Keywords like `if`, `else`, `int`, `char` etc. can't be used as variable names
- Use meaningful names to indicate the purpose of the variable

Data Types and Sizes

`char`	single byte capable of holding one character
`int`	an integer
`float`	single-precision floating point
`double`	double-precision floating point

Qualifiers:

- E.g.

```
short int a;
long int c;
unsigned int d;
```

- `short` will modify the size taken by an `int`. Instead of 32 bits, the integer will now be represented using 16 bits

Constants

1234	`int`
1234566789L	`long`
1234566789ul	`unsigned long`
1.1 or 11e-1	`double`
0x3f	`hexadecimal`
037	`octal`
‘a’	character constant, ASCII value 97

- ASCII – American Standard Code for Information Interchange

Constants

- Constant expression:

```
#define LEN 100
char line[LEN+1];
```

- String constant or string literal

```
"this is a string"
```

- can be concatenated at compile time:

```
"this is" "a string"
```

- enumeration constant:

```
enum boolean {NO, YES};
```

- \ followed by another character is called an escape sequence, which are translated to another character when used in a string literal

Declarations

- Declaration specifies a type, and a list of one or more variables of that type:

```
int high, mid, low;
char a,c;
```

- It can also be split into separate lines
- const qualifier specifies that its value will not be changed:

```
const double e = 2.71828182845905;
const char st[] = "Test String";
```

Arithmetic Operators

- `x % y` gives remainder when x is divided by y
- % can't be applied to float or double
- + and - have same precedence, but lower than * / and %

Relational and Logical operators

- `>, >=, <, <=` have the same precedence
- Outcome is true or false, indicated by digits 1 or 0, respectively
- `a < b+1` means `a < (b+1)`
- `&&` (logical AND) and `||` (logical OR) operations are evaluated left to right
- E.g. for `int a = 1, b = 2;` – outputs for different cases are shown:

`a > b`	0 (false)
`a < b`	1 (true)
`a+1 >= b`	1

- Any non-zero (positive or negative) value is considered true
- For `a=0, b=10`

`a && b`	0
`a || b`	1

Type conversions

- When an expression has operands of different types, they are converted to a common type
- Automatic conversions convert a narrower data type to a wider one or vice versa
  - E.g. f = f + i;
  - An implementation of `atoi` to convert a character string of digits to its numeric equivalent
- Type conversions can also be forced with a unary operator called a *cast* E.g. to convert 'i' from an integer to a double, we may use `(double)i`

Type conversions

E.g.

- Implicit type conversion

```
int i = 3, j;
float f = 4.0;
f = i+f;
```

  - On the RHS, `i` is converted to float first, then addition is performed, and finally assigned to `f`
  - If `i` was on the LHS instead of `f`, all the above steps occur, but during assignment the result is converted to an integer

- Explicit type conversion (casting)
  - This is useful when dealing with fractions having integer data type

```
int i=11, j=12;
float f=(float)i/j;
```

  - If we had left out '(float)', the result would have been 0

Extras – Floating Point Representation

The `float` and `double` data types are represented using IEEE 754 floating point format

- `float` takes 32 bits of memory
- Its format is given by

S	E (8)	Mantissa (23)

- Value is : $(-1)^S \times 2^{E-127} \times$ Mantissa
- `double` takes 64 bit of memory and the format is given below

S	E (11)	Mantissa (52)

- Value is given by: $(-1)^S \times 2^{E-1023} \times$ Mantissa
- Search around for examples

Increment and Decrement operators

- `++` adds 1 to operand
- `--` subtracts 1 from the operand
- Can be used as postfix or prefix
- `x++;` and `++x;` is same as `x = x+1;`
- The following table shows the difference when using the increment operator as a postfix and a prefix to the variable `x`

Using increment operator	Equivalent statements

`int x = 5;`
`int a = x++;`
	`int x = 5;`
	`int a = x;`
	`x = x+1;`

`int x = 5;`
`int a = ++x;`
	`int x = 5;`
	`x = x+1;`
	`int a = x;`

- The same applies to the decrement operator too

Bitwise Operators

- Has six operators for bit manipulation

&	bitwise AND
|	bitwise inclusive OR
^	bitwise exclusive OR
<<	left shift
>>	right shift
~	one's complement (unary)

- & masks of some bits
  - `n = n & 0177;`
  - last 7 bits retain the previous values, all higher bits set to 0

Extras – Experiment with debugger (gdb)

- gdb is a standard debugger available in GNU/Linux systems
- A debugger can be used to pause a running program and check the state of program (values in variables, trace of functions called etc)
- Along with being a debugger, it can be used as a programmer's calculator
- Run gdb without arguments
- Set a variable in gdb and try all the operators
  - `(gdb) set $a = 10`
  - `(gdb) p/t $a`
  - `(gdb) p/t ~$a`
  - `(gdb) p/t $a&10`
- `/t` is a switch to the print command which tells the debugger to display the variable in binary
- Other switches: `/x`, `/o`, `/d`

Assignment operators and expressions

- Expressions where a variable on LHS is repeated immediately on the RHS can be written in a compact form
  - `i = i+2;` $\Longrightarrow$ `i += 2;`
  - `+=` is called an assignment operator
- Thus

  expr_1 op= expr_2 is equivalent to
  $\text{expr}_1 = (\text{expr}_1)$ op (expr_2)
  - `x *= y+1;` means `x = x * (y+1);`

Operator Precedence

Operators	Associativity
`() [] -> .`	left to right
`! ~ ++ -- + - * & (type) sizeof`	right to left
`* / %`	left to right
`+ -`	left to right
`<< >>`	left to right
`< <= >= >`	left to right
`== !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`?:`	right to left
`= += -=` etc.	right to left
`,`	left to right

Operator Precedence – Examples

- From the table, find the output of each, when `a = 2`, `b = 3`, `c = 4`
  - `a + b<<3 + c`
  - `a ^ b & 5 + c * 3`
  - `(a ^ b) & (5 + c) * 3`