Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Software Systems

Lectures Week 9

Basic Software Engineering Techniques, and Introduction to Systems Programming

(GDB, GIT, Machines)

Prof. Joseph Vybihal
Computer Science
McGill University

Week 9 Lecture 1

GDB and Git

Readings: http://www.tutorialspoint.com/gnu_debugger/ and https://www.learnenough.com/git-tutorial?gclid=Cj0KEQjw8tbHBRC6rLS024qYjtEBEiQA7wIDeXo_owx61WwsbBiFPTFsgvcLriD526qlAmford4B1IgaAk9j8P8HAQ

Bug

A software bug is an error, flaw, mistake, failure, or fault in a computer program that prevents it from working as intended, producing an incorrect result.

- Bugs can exist at different levels
  - Design
  - Source Code
- Bugs have severities
  - Some bugs produce an incorrect answer
  - Some bugs simply crash an application.
  - Some bugs cause loss of data.
  - Some bugs cause loss of money.
- The worst bugs cause loss of life.

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Well known bugs

- Y2K – date overflow
- Ariane 5 Flight 501 – conversion overflow
- MIM-104 Patriot bug – clock drift
- MARS orbiter crash - metric and imperial

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Debugging

Debugging is the act of finding the source of a bug and fixing it.

- The hardest part of debugging is finding the problem.
  - This becomes exponentially difficult when the source is very large.
- Just analyzing the code is not always enough to find bugs.
  - You need to run the application.
- Debuggers are tools that help the debugging process.

Debuggers

- Debuggers allow a programmer to run the application in a different mode.
- When running under this different mode ("debug mode"), many new features are available to the programmer.
  - If an application crashes, the programmer can see what line caused the fatal operation. He can also consult the content of the memory.
  - A programmer can analyse an application, line-by-line. At each line, he can consult the content of the memory.
- Most programming languages have a debugger.

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

To Printf or Not to Printf

- Printf is useful when debugging
  - How can we use it? Suggestions?
- A symbolic debugger can do things printf() can't.
  - halt the program temporarily,
  - list source code,
  - print the datatype of a variable
  - jump to an arbitrary line of code
- You can use a symbolic debugger ON a process that has already crashed and died.

Introduction to GDB

GDB is a debugger which is part of the Free Software Foundation's GNU operating system.

GDB can be used to debug:

- C, C++, Objective-C, Fortran, Java and Assembly programs.

Unix
Bash
C
GNU
Systems

Using GDB

First:

- gcc -g -o HelloWorld HelloWorld.c
  - The program is then compiled with additional information such as the source code and symbol table.

This additional information is needed by the debugger.

COMP 206 – Joseph Vybihal
Software Systems

Starting GDB

Second:

```
gdb HelloWorld
```

The executable

```
GNU gdb Red Hat Linux (5.3post-0.20021129.18rh)
   Copyright 2003 Free Software Foundation, Inc.
   GDB is free software, covered by the GNU General
   Public License, and you are welcome to change it
   and/or distribute copies of it under certain
   conditions. Type "show copying" to see the
   conditions. There is absolutely no warranty for
   GDB. Type "show warranty" for details. This GDB
   was configured as "i386-redhat-linux-gnu"...

(gdb)
```

The command line

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Getting Help

Type Help:

```
(gdb) help
```

You can also get help on a specific command by typing “help” and the name of that command:

```
(gdb) help breakpoints
 Making program stop at certain points.
```

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Looking at the Source code

You can use the *list* command:

- List
- List filename
- List linenumber
- List function

```
(gdb) list main.c:10
5
6       int main (int argc, char **argv) {
7
8               library* mylibrary = createLibrary(20);
9
10              loadLibrary("lib.txt", mylibrary);
11
12              addBookToLibrary(mylibrary, createBook("Lotr", "Tolkien", 300));
13              addBookToLibrary(mylibrary, createBook("Harry_Potter", "Rowing",
    50));
14              addBookToLibrary(mylibrary, createBook("C_Prog", "Kerning", 100));
```

You can change the size of a list using the *set listsize*

GNU Debugger with Core Dump

```
$ gcc -g file.c
$ gdb a.out core-dump
  GDB is a free software and you are welcome
   to distribute copies of it under centerain conditions;
   GDB 4.15.2-96q3; Copywrite 2000 Free Software Foundation,
   Inc.

   Program terminated with signal 7, Emulation trap.
   #0 0x2734 in swap (l=93643, u=93864, strat=1) at file.c:110
   110    x=y;

   (gdb) run
```

The run-time error message (crash)

Some Terminology

```
Bash-prompt $ gdb a.out
(gdb) break 4
(gdb) run                    ← run until end, crash, or breakpoint

int main() {
  int age;

  printf("Enter age: ");
  scanf("%d", &age);                         Breakpoint   Stop & prompt
                                                            Step & inspect
  if (age > 17)                                             Step & inspect
          printf("Welcome\n");                              Continue
  else
          printf("Try again when you are older\n");

  return 0;
}
```

How to use GDB

- Run you program to see error in Bash
- Use printf to solve problem – not good?
- Take notice to where the problem happened
- Bash-Prompt $ gdb a.out
- (gdb) break before-problem-location
- (gdb) step and print through code to see
- (gdb) change variables, continue or run fn

Step-by-Step

STEP & NEXT

- STEP
  - Step into
  - Go to next line of program. If the next line is a function call then enter into the function.
- NEXT
  - Skip next
  - Go to next line of program. If the next line is a function call then do not go into the function just execute the function in its entirety. Go to the next line after the function call.

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

GDB Command-line Commands

•Quit	end gdb
•List	show 10 lines
•List n,m	show lines n to m
•List function	show all of a function by name
•Run	run your program
•Run (later ctrl-c)	run, then interrupt program
•Run –b < invals > outvals	redirect input and output to program
•Backtrace	see the run-time stack (call stack)
•Whatis x	show x’s declaration
•Print x	show value stored at x
•Print fn(y)	execution fn with y as parameter
•Print a @ length	show “length” elements of array a

Unix
Bash
C
GNU
Systems

GDB Command-line Commands

•break LINE_NO	interrupt program at line number
•break FUNC_NAME	interrupt program at function call
•break LINE_NO if EXPR	interrupt at line number if expr true
•break FUNC_NAME if EXPR	interrupt at fn call if expr true
•break FILE_NAME:LINE_NO	interrupt at line number is source-file file
•continue	continue program execution after break
•watch EXPR	stop program as soon as expr is true
•set variable NAME = VALUE	change contents of a variable
•ptype NAME	pretty print of structure n
•call fn(y)	execute fn with parameter y

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

GDB Printing Data and History

•(gdb) whatis p
type = int *

•(gdb) print p
$1 = (int *) 0xf8000000

•(gdb) print *p
$2 = Cannot access memory at address 0xf8000000

•(gdb) print $1-1
$3 = (int *) oxf7fffffc

•(gdb) print *$3
$4 = 0

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

GDB Breakpoint Management

•(gdb) break 17
Breakpoint 1 at 0x2929: file.c, line 17.

•(gdb) break 30 if x == 100
Breakpoint 2 at 0x3550: file.c, line 30.

•(gdb) info breakpoints

Num	Type	Disp	Enabled	Address	What
1	breakpoint	Keep	Y	0x2929	in calc at file.c: 17
2	breakpoint	Keep	Y	0x3550	in sum at file.c: 30

•(gdb) delete 1
•(gdb) delete ← everything!
•(gdb) clear 17 ← any break or watch on line 17
•(gdb) disable n ← do not delete but turn off
•(gdb) enable n
•(gdb) enable once n ← turn on for one time

Where am I?

At any moment, even after a crash, you can use the *where* command to produce a backtrace. (stacktrace)

```
(gdb) where

#0  createBook (title=0x804a218 "Lotr", author=0x804a110
   "Tolkien",
    pages=300) at book.c:8
#1  0x080487fe in loadLibrary (filename=0x8048aa8 "lib.txt",
    myLibrary=0x804a008) at file.c:20
#2  0x08048567 in main () at main.c:10

(gdb)
```

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

GIT

Backup

A folder that contains a copy of your file.

Often the copy is older than the original file because as you develop you modify the original file.

```
Bash-prompt $ vi file.c
Bash-prompt $ cp file.c backup
Bash-prompt $ vi file.c
```

If something bad happens to your original file you can always revert back to your backup copy.

Repository

A database containing all the versions of your file (version control system)

This is an improvement over a backup because a repository contains the entire history of all your backup files, also:

- It can be shared
- You can assign permissions
- You can enforce development and deployment rules

Repository History

File1.c version 1 ← Known as the **root** version

↓

File1.c version 2

↓

File1.c version 3

↓

File1.c version 4

↓

File1.c version 5 ← Known as the **tip** version

The files in your directory as
known as the **current** version

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Git

A popular version control system

Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Its current maintainer since 2005 is Junio Hamano.

How to use basic Git

First – create the folder for your project

Bash-prompt $ mkdir project

Second – put/create the files for the project

Bash-prompt $ cd project

Bash-prompt $ cp ../afile

Bash-prompt $ vi anotherfile

Third – create the Git repository

Bash-prompt $ git init

How to use basic Git

Fourth – Select the files to put in the repo

Bash-prompt $ git add file1 file2

Fifth – Now put the selected files into the repo

Bash-prompt $ git commit –m “message”

- It is very important to add a message describing the changes you made since the last commit. In the future if you will want to undo what you did these messages will help you figure out how far back you may want to go.

Sixth –

- This is the basic loop.
- You can now continue working on your files until the next commit, where you will repeat steps 4 and 5.

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

How to get something back

There are two cases where you may want to undo your work:

- You did an commit but you want to undo it
  - Bash-prompt $ git checkout -f
- You made an error in your original file that you cannot correct, so you want to go back to a previous version
  - Bash-prompt $ git log ← to see commit history, find <SHA>
  - Bash-prompt $ git checkout <SHA>

Which files to commit

There are two cases:

- You were not paying attention to the files you changed and now you would like to commit
  - Bash-prompt $ git diff ← all tip files compared with current
- Your team member worked on the file at the same time as you!!
  - Using Bash:
    - Bash-prompt $ diff yourfile friendfile
    - Bash-prompt $ vi files and copy past, then use git add/git commit
  - Using Git:
    - Bash-prompt $ git diff filename ← compares tip file with current file

Important

Git tries its best to merge different versions of a file automatically by itself.

It will resort to showing you the diff of two or more files when it is having trouble merging.

Unix
Bash
C
GNU
Systems

How to make & use a shared Git

Step 1 – create project folder & cd into it

Step 2:

- Start your own shared Git
  - Bash-prompt $ git init –bare ← makes it more compatible
  - Share your URL with friends
- Join an existing shared Git
  - Bash-prompt $ git clone <url>

The <url> is:

- ssh://user@server/path/folder
- http://user@server/path/folder

Often you will need to ask the system administrator for the URL info/permission.

COMP 206 – Joseph Vybihal
Software Systems

How to make & use a shared Git

Step 3 – Before you work on any file, make sure that no one else has remotely changed the file

Bash-prompt $ git pull <url>

Step 4 – Now vi your files

Step 5 – Commit to the shared repo

Bash-prompt $ git add <filename>

Bash-prompt $ git commit –m “message”

Bash-prompt $ git push <url>

Unix
Bash
C
GNU
Systems

How to work in a team

Option 1:

- Assign a source file to a developer.
- No other developer is permitted to edit that file.
- Option 1 can be carried out with the git commands we have already seen.

Option 2:

- Anyone can edit any file.
- Divide the work into branches.
- Option 2 requires us to understand branching.

Repository History

Mary fixing function 1

File1.c version 1 ← Known as the **root** version

File1.c version 2

File1.c version 3

File1.c version 4

File1.c version 5 ← Known as the **tip** version

At same time Bob fixing function 2

Branch 1

File1.c vr 3.1.1

File1.c vr 3.1.2

File1.c vr 3.1.3

At same time Beth fixing function 3

Branch 2

File1.c vr 3.2.1

File1.c vr 3.2.2

File1.c vr 3.2.3

File1.c vr 3.2.4

Merge point

File1.c version 6

Uses:
- Auto merging, and
- Diff

To sort things out.

How to use branching

Step 1 – See the current branches

Bash-prompt $ git branch

Step 2:

- Join a branch, or
  - Bash-prompt $ git checkout <branch_name>
- Create your own branch
  - Bash-prompt $ git branch <new_branch_name>
  - Bash-prompt $ git checkout <new_branch_name>
  - Or do it in one shot:
    - Bash-prompt $ git checkout –b <new_branch_name>

Step 3 – now whatever you do it will effect only the branch

How to use branching

Step 5 – To exit a branch

Bash-prompt $ git checkout master

Step 6 – To merge a branch with the master

Bash-prompt $ git checkout master

Bash-prompt $ git merge <branch_name>

Step 7 – Manager branches

- Delete merged branch
  - Bash-prompt $ git branch -d <branch_name>
- Delete a branch that has not been merged
  - Bash-prompt $ git branch -D <branch_name>

Week 9 Lecture 2

Class Test

Week 9 Lecture 3

Introduction to Systems Programming

Unix
Bash
C
GNU
Systems

About Systems Programming

Software that interacts with the computer and not only with human users

There are three basic ways to interact with your system:

- The Shell
- The OS through libraries
- Directly with the devices connected to your machine

COMP 206 – Joseph Vybihal
Software Systems

The Shell

There are three basic ways to interact with the shell:

- The command-line arguments (this lecture)
- Executing shell commands (later)
- The shell memory (later)

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

C Shell Arguments

Bash-prompt $ ./a.out 15 Bob 4.2 X

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

argc ← counts the number of arguments
argv ← an array of strings, each cell is one argument

In this example:

argc = 4
argv = {“a.out”, “15”, “Bob”, “4.2”, “X”}

C Shell Arguments

Bash-prompt $ ./a.out 15 Bob 4.2 X

```
int main(int argc, char *argv[]) {
    int a; char name[30]; float b; char c;

    // Assume I am expecting 4 arguments
    if (argc != 4) exit(1);

    a = atoi(argv[1]);
    strcpy(name, argv[2]);
    b = atof(argv[3]);
    c = *argv[4];
}
```

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Example

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
  char c; FILE *p, *q;

  if (argc != 2) exit(1);

  p = fopen(argv[1],”rt”);
  q = fopen(argv[2],”wt”);
  if (p==NULL || q==NULL) exit(2);

  c = fgetc(p);
  while(!feof(p)) {
          fputc(c,q);
          c = fgetc(p);
  }

  fclose(p);
  fclose(q);
  return 0;
}
```

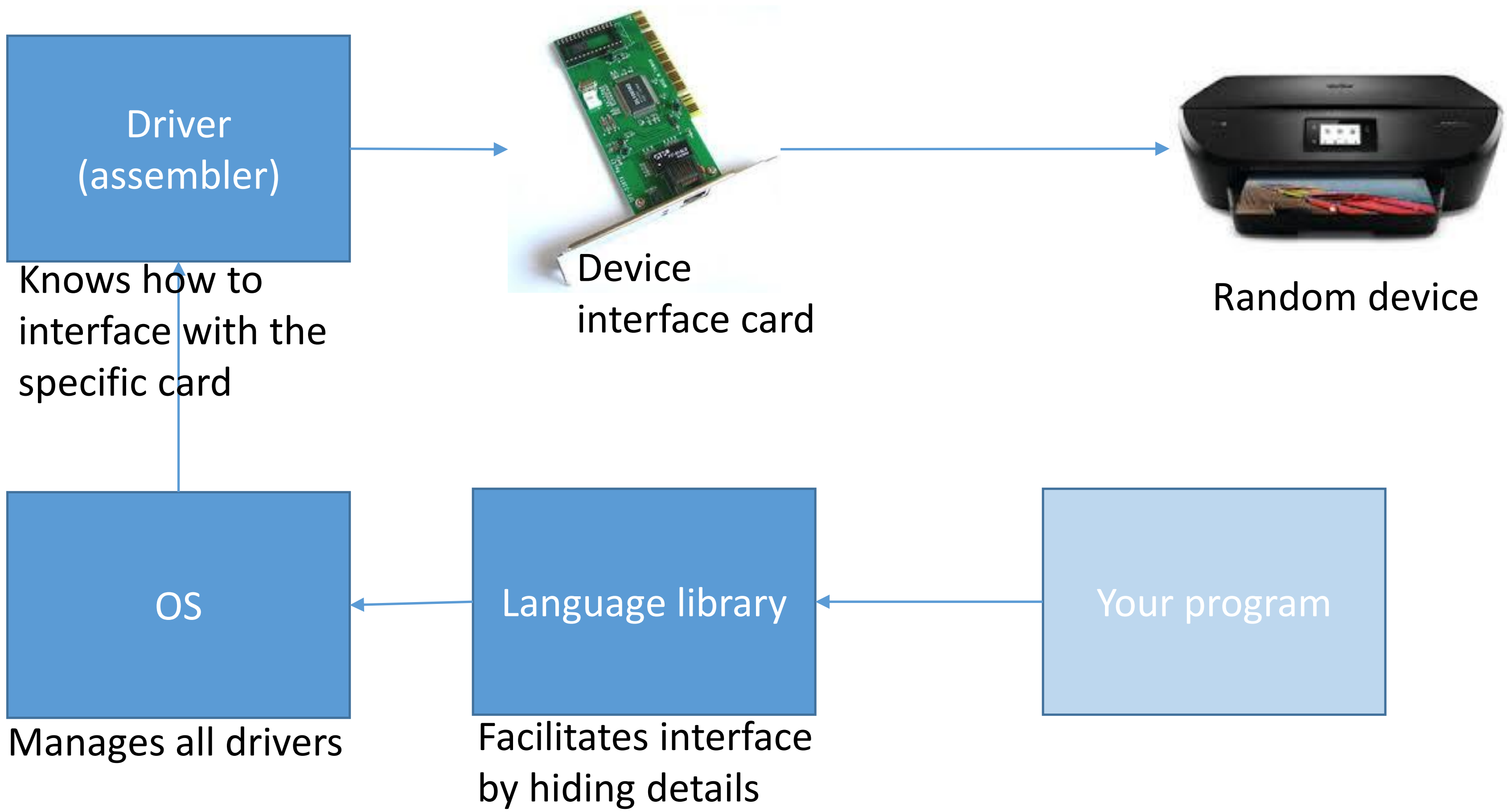
```
Bash-prompt $ vi copy.c
Bash-prompt $ gcc –o copy copy.c
Bash-prompt $ ./copy file1.txt file1.bak
```

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Libraries as system interfaces

Every machine connected to your computer passes through the same pipeline:



Example

#include<time.h>

- Connects to the system clock
  - time_t seconds; // struct stores time since Jan 1 1970
  - struct tm // parsed date & time structure
    - tm_year, tm_ mon, tm_ mday, tm_ hour, tm_ min, tm_ sec, tm_ isdst
  - seconds = time(NULL);// local time since Jan 1 1970
    - printf("Hours since 1970: %d", seconds/3600);
  - float x = difftime(secondsX, secondsY);
  - struct tm *t = localtime(&seconds);
  - char *p = asctime(t);
  - seconds = mktime(t);

The time.h library

struct tm

int	tm_sec	seconds [0,59]
int	tm_min	minutes [0,59]
int	tm_hour	hour [0,23]
int	tm_mday	day of month [1,31]
int	tm_mon	month of year [0,11]
int	tm_year	years since 1900
int	tm_wday	day of week [0,6] (Sunday = 0)
int	tm_yday	day of year [0,365]
int	tm_isdst	daylight savings flag

time_t

An unsigned long integer or unsigned long double number (depends on implementation) that measures the number of milliseconds from a fixed point in time to the present as an offset (or distance measurement).

Jan 1 1970 is the earliest date/time that can be represented. Jan 1 1970 = 0.

The time.h library

```
char *        asctime(const struct tm *);
char *        asctime_r(const struct tm *, char *);
clock_t       clock(void);
int           clock_getres(clockid_t, struct timespec *);
int           clock_gettime(clockid_t, struct timespec *);
int           clock_settime(clockid_t, const struct timespec *);
char *        ctime(const time_t *);
char *        ctime_r(const time_t *, char *);
double        difftime(time_t, time_t);
struct tm *getdate(const char *);
struct tm *gmtime(const time_t *);
struct tm *gmtime_r(const time_t *, struct tm *);
struct tm *localtime(const time_t *);
struct tm *localtime_r(const time_t *, struct tm *);
time_t        mktime(struct tm *);
int           nanosleep(const struct timespec *, struct timespec *);
size_t        strftime(char *, size_t, const char *, const struct tm *);
char *        strptime(const char *, const char *, struct tm *);
time_t        time(time_t *);
int           timer_create(clockid_t, struct sigevent *, timer_t *);
int           timer_delete(timer_t);
int           timer_gettime(timer_t, struct itimerspec *);
int           timer_getoverrun(timer_t);
int           timer_settime(timer_t, int, const struct itimerspec *, struct itimerspec *);
void          tzset(void);
```

Unix
Bash
C
GNU
Systems

Example

Using time.h as a profiler.

```
#include <stdio.h>
#include <time.h>
 int main()
 {
    time_t begin,end;
    long i;

    begin= time(NULL);
    for(i = 0; i < 150000000; i++);
    end = time(NULL);

    printf("for loop used %f seconds to complete the execution\n", difftime(end, begin));

    return 0;
}
```

Unix
Bash
C
GNU
Systems

Directly with devices connected to the computer

As a system's language, C permits direct access to devices.

- A device card has an address
  - void *p = 145; // assume we know the card's address is 145
- We can communicate with the device
  - *p = 'A'; // if p points to printer then printer prints 'A'
  - int x = *p; // if p points to printer status then x = status
- Devices speak in binary…

Binary

Counting in decimal and binary:

0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101

In Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
In Binary : 0, 1

In decimal: 9 + 1 = 10 (we reuse digits, = 10)

In binary: 1 + 1 = 10 (we reuse digits, = 2)

Binary is used in many ways:

- The char is ASCII which is coded binary
- Numbers are stored as binary
- Binary 00101 can be thought of as three false values and two true values

Manipulating binary in C

Operators:

&	Binary and
|	Binary or
~	Binary complement
>>	Binary shift right
<<	Binary shift left

Manipulating binary in C

Meaning:

&	1011 & 0110 → 0010
|	1011 | 0110 → 1111
~	~1011 → 0100
>>	1011>>3 → 0001
<<	1011<<2 → 1100

AND:

A	B	AND	R
1	1		1
1	0		0
0	1		0
0	0		0

OR:

A	B	OR	R
1	1		1
1	0		1
0	1		1
0	0		0

Complement:
Means oposite

Manipulating binary in C

Usage:

```
int a = 11; // Binary 1011
int b = 6;  // Binary 0110
int r;
r = a & b;  //    1011 & 0110 → 0010
r = a | b;  //    1011 |  0110 → 1111
r = ~a;     //    ~1011 → 0100
r = a>>3;   //    1011>>3 → 0001
r = a<<2;   //    1011<<2 → 1100
```

Example

Masking:

- When we want to change a single bit, or
- When we want to find out about a bit.

```
int b = 6;   // Binary 0110

b = b | 0001;      // "set" the last bit to 1
b = b &1110;       // "set" the last bit to 0

if (b & 0001)      // =0 if last bit was 0, else =1
```