Welcome to the CS50 glossary. This is a file which hopes to
clarify any terms which you can’t figure out.

This glossary was maintained by Saurabh Jha (sjha@fas). It is
currently being maintained by Winslow Taub (wbtaub@fas). If you
think anything could be done to improve this glossary, do not
hesitate to email me. This includes requesting for additional
terms, so if anything from lecture or any of the readings is
unclear, drop me a line.

To use this glossary you should be able to just type ’glossary’
at the prompt. This should take you into an editor (by default,
vi) and you can go through this file.

The general format of an entry is as follows

! <term>:

<definition>

If you’re using vi and are in command mode, you can search for a
term by hitting ’/’ and searching for ’! <term>’ with <term> being
whatever term you’re looking for.

****

! address:

n. This is a number which refers to the location of a particular
piece of memory. Each memory address refers to one byte of data
in memory. Addresses are generally given in hexadecimal format.

! addressing mode:

n. In SPIM this is a method of accessing data. Data can either
be in a register or in main memory. If data is in memory, the
address of its memory location must be known in order to access
it.

! algorithm:

n. A procedure to do something. Algorithms must be precise (meaning
that they should not be ambiguous), correct (they should do what they
intend to do) and finite (they must have a definite end).

! ancestor:

n. In a binary tree, any node in the path above a given node.

! argument or ! parameter:

n. An input to a function. If you think of a function as a black
box, the arguments are what you send into the black box in order
to get an answer out. Sending arguments to a function is called
passing the arguments.

! array:

n. A group of linear memory locations referenced by one variable.
An array has an index, which is a number indicating the position

in the array. In the black box analogy, an array is like a group of boxes in a row. The first position in an array always has an index of 0.

! assembly language:

n. A programming language (like C) which is very low-level. Instructions in assembly language are highly processor-specific, and often translate to just one or two actual machine language instructions.

! binary tree:

n. A data structure which consists of elements, each containing two pointers to the same type. These pointers go to the "left" and "right", and form a tree like structure. Binary trees are very efficient when it comes to searching, although it is difficult to insert elements into an existing tree. Elements of a binary tree are called nodes.

! bit:

n. Short for binary digit. It is the smallest unit of representation that is used by a computer. A bit can be either 0 or 1.

! boolean:

adj. A description applied to a variable or expression which indicates that it can take on one of the two values: TRUE or FALSE. In C boolean values are implemented as integers, with 0 indicating FALSE and non-zero indicating TRUE.

! block or ! compound statement:

n. A group of simple statements surrounded by curly braces ({ and }). A block is considered as one statement in C constructs. Variables in C can only be declared at the beginning of a block.

! branching:

v. In SPIM, this is the process of controlling execution. A branch to a label causes the instruction at that label to be executed, instead of just the next line. Branching occurs in two types, conditional and unconditional. Conditional branching refers to branching only in a certain situation (for example, when one register is greater than another). Unconditional branching occurs regardless of the situation.

! breakpoint:

n. A place in a program to halt execution. Breakpoints are usually set in code through the use of a debugger. This enables the programmer to check the contents of variables, step through the code one line at a time, etc.

! byte:

n. A group of eight bits. This is usually the smallest unit of data that people refer to when they talk about memory organization.

! call or ! function call:

n. A statement which executes a function.

```
e.g.:   printf("Hello!\n");
```

v. Execute a function.

! caller:

n. A term referring to the _function_ which calls another function. For example if in the main() function, the statement:

```
printf("Hello!\n");
```

appears, main is said to be the caller of printf. It is important to distinguish the caller from a person who is using the program. The latter is always referred to as the user.

! cascading if:

n. The construct as follows:

```
if (expr1)
        statement1;
else if (expr2)
        statement2;
...
else if (exprn)
        statementn;
else
        statement;
```

where all the expr's are expressions and all the statements are simple or compound statements. If a cascading if is of a certain form, a switch statement is often a better implementation.

! central processing unit or ! CPU:

n. This is the "brain" of a computer; also called a microprocessor. It contains a register file (or register set), a control and a datapath. The datapath, in turn, consists of an ALU (arithmetic logic unit, to do arithmetica and logical operations) and a shifter. Examples of CPUs include the Intel 80x86, Pentium; Motorola 680x0, PowerPC, MIPS R4000, PA-Risc.

! children:

n. In a binary tree these are the nodes immediately below a given node.

! conditional:

n. A statement in C which makes a decision. Conditionals (such as the if..else construct and the switch construct) are usually used to decide what instructions will get executed next (called the control flow.)

! control flow:

n. The flow of execution in a program. Most programs do not always execute each line of code in succession, but rather do different things in different situations. The control flow is overall diagram of how the program will execute given a particular situation.

! debug:

v. The process of finding errors (usually alogrithmic) in some code and fixing them. This process is greatly simplified with the use of a _debugger_ such as gdb.

! declaring a function:

v. To tell the compiler that a certain function exists an can be used. In a declaration, the function name, its return value (if any) and the types of its arguments must be specified.

! defining a function:

v. Actually writing the code for the function (called the body). This is the code that is executed when the function is called.

! descendant:

n. In a binary tree, any node which is below a given node.

! deterministic:

adj. A term which refers to actions which are completely predictable. Deterministic programs produce identical ouputs when given identical inputs.

! dereference:

v. The act of taking a pointer variable, and accessing the contents of the memory location which is in the pointer.

! dump core or ! drop core:

v. What happens when your program does something bad (like segmentation fault). The core is a file which contains the image of what the memory looked like when the error occurred, so the programmer can try and figure out what went wrong (usually with the help of a debugger).

! dynamic memory allocation:

n. A system in which space for variables is allocated when the need arises, in the course of program execution. This is accomplished through the use of the malloc() function (or one of its relatives). This is contrasted with static memory allocation in which all the space is reserved ahead of time. The advantage of doing this dynamically is that the programmer does not have to know in advance how much room will be required to hold the data.

! expression:

n. An expression is the basic building block of the C language. For a precise definition of an expression (which would fill pages) take a look at Kernighan and Ritchie, _The C Programming Language_, 2nd Ed., pp. 200-9. An important point is that every expression in C takes on a value, that is, the compiler interprets every expression by assigning to it a value, which can be used in other computations.

! field:

n. One of the components of a structure.

! file:

n. A set of bytes grouped under a common name on a disk. Files can be of many types, ranging from pure text files (those containing bytes representing ASCII characters) to binary files (those containing raw data in binary format). Almost everything that goes on in UNIX manipulates files and their contents. A file has a name (called its filename) by which you or any program can refer to it. Whatever information is stored on disk in a computer is stored in the form of files (be they text, data or programs.)

! floating point number:

n. A number which has (or could have) digits to the right of the decimal point. To keep track of this information, such a number must be stored in a variable of type float or type double.

! function:

n. A function in C is a subroutine designed to implement a specific algorithm. A function must be declared and defined in order to be used. The function must have a name; it may take arguments (or parameters) and it may have a return value.

! global variable:

n. A variable which is declared outside of any function, and thus which can be accessed by any function in the file, or across a number of files.

! hierarchical decomposition:

n. Also called top-down programming. This is the idea of solving a problem by breaking it down into smaller, more manageable steps. It is the basis for algorithmic thinking. In C, it translates primarily to the idea of writing functions to handle different aspects of the problem.

! home directory (~):

n. This is the directory which is the current directory whenever you log in. Every user has a home directory. A user can refer to his or her home directory as ~, e.g. more ~/.login . A user can refer to the home directory of another user by ~username where username is replaced by the username of the owner of the other home directory. Users are allowed (with some exceptions) only to place files in their own home directories (or subdirectories of their home directories.)

! iteration:

n. Also known as looping. This is the process of having the computer execute the same instructions a number of times (the number may or may not be known in advance). The iteration constructs in C are the for loop, the while loop and the do..while loop.

! leaf:

n. A node in a binary tree which has no children.

! library:

n. This is a file which contains functions that have already been written and are available for programs to use. Libraries generally consist of two parts, a header (.h) file and a precompiled object (.a) file. To use a library function, you must use the #include directive in your program to include the correct header file. In addition you must link into your program the precompiled object file, which is done when you are actually compiling and linking the program. Examples of libraries include the standards: stdio.h, math.h and the ones Roberts defines: simpio.h, genlib.h, etc.

! linked list:

n. This is a data structure which is a collection of structures. Each structure has at least one pointer to its own type. This pointer usually points to the next (or previous) element in the list. In this way the list is "linked" that is, starting at the first element, following each element's pointer will take you through the whole list.

! local variable:

n. A variable which is declared inside a function (or inside a block in a function), and which cannot be accessed outside of the function.

! memory:

n. The storage area for a computer, just like a person's memory. This is the place that variables are stored, as well as where programs are (when they are running). This type of memory is not permanent; it goes away if the machine is turned off or crashes.

! Monte Carlo technique:

n. This is a technique of approximating results by using random samples to estimate the characteristics of a large group (for example, finding the area of a region).

! nondeterministic:

adj. Actions that are not predictable. The same inputs to a nondeterministic program may give different outputs. This is useful when programming games or simulations.

! parent:

n. The node immediately above a given node in a binary tree.

! pass:

v. To send arguments to a function. For example if I say:

```
y = foo(x);
```

then I have _passed_ the variable x into the function foo.

! passing by reference:

v. Passing arguments to a function by passing their memory locations; that is, pointers to them. In this way, the function can actually manipulate (and thus change) the values stored in those locations. This enables the function to change variables which are not local to it.

! passing by value:

v. Passing arguments to a function by passing only the contents of the variables. In this way the called function only gets copies of the argument variables. So the function cannot change the original variables.

! pointer:

n. A variable whose contents are a memory address. Pointers in C are just like any other data type. You can access the contents of the memory location in a pointer by dereferencing it.

! program:

n. An executable file which contains instructions for the computer to carry out. To execute a program in Unix, you just type its filename.

v. The act of creating a program; also the ultimate in sadism. :-)

! pseudo-random number:

n. A seemingly random number generated by a computer or other pseudo-random generator. Since the inner workings of a computer are deterministic, it is not possible to generate truly random numbers. Instead, we generate sequences of numbers that behave like random numbers from a statistical point of view and are sufficiently difficult to predict.

! random number:

n. A number whose value is totally impossible to predict. Any number that can be generated is equally likely to be generated.

! recursion:

n. The idea of a function calling itself in a simpler fashion. A recursive function has two parts: a base case, which is when the function returns, and a recursive case in which the function calls itself.

Also, see recursion. :-)

! register:

n. Part of the "register file" or "register set" on the microprocessor or CPU. Registers are like temporary variables which can be accessed very quickly. On most modern computers, operations are performed primarily on the registers.

! return:

v. The act of a function completing and resuming execution from where it was called. If the function has an output, it is said to _return_ a value.

! return value:

n. The output of a function. More formally, it is the value that the function takes on when viewed as an expression. This value then can be used as any other (in assignment, etc., for example).

! root:

n. In a binary tree, this the node at the top. [In computer science trees grow upside down. :-)]

! seed:

n. A number used to start a random number generator. The same seed will result in the same sequences of numbers from a pseudo random number generator. Thus different seeds should be used in order for a program to run with different random numbers. The way to achieve this is ordinarily to use a seed derived from the current time.

v. The process of passing a seed to a random number generator.

! shell:

n. A program which handles the interaction between the user and the operating system. The shell reads all commands typed in and tries to figure out what they mean (e.g. execute a specific program, set an environment variable, etc.). Different shells have different features, allowing you to control your environment to varying degrees. Some of the shells available on this system include, sh (Bourne shell), bash (GNU Bourne-Again shell), csh (Berkeley C shell), tcsh (enhanced C shell) and ksh (Korn shell). Check out the manpages to find out about each shell. The ones we recommend are tcsh and csh.

Sample Usage:
"She sells C shells by the sea shore." :-)

! short circuit evaluation:

n. This is the method which is used by the C compiler in determining the value of a boolean expression involving logical operators. Once the value of an expression is determined definitively, the compiler stops evaluating other parts. For in example the expression

```
(x < y) || (printf("hello"))
```

does not print out the message hello, if x < y since the logical expression can be evaluated as true before even looking at the second part of the condition.

! sorting:

v. The process of placing an array (or some other data structure) in some kind of order.

! stack:

n. A data structure which can be thought of as a stack of trays in the Union dining hall (supposedly where the idea originated). Items can be "pushed" onto the stack and then "popped" off. Note that things pushed onto a stack are popped of in reverse order. Thus a stack is a LIFO data structure, that is, Last In First Out. The last thing pushed onto a stack is the first thing that can be accessed.

! statement or ! simple statement:

n. In C statements are expressions followed by a semicolon (;). Statements can be grouped together in curly braces ({ and }) to form blocks or compound statements.

! structure:

n. In C a stucture is a collection of data (possibly of different types) that is grouped together and treated as a single object. The internal elements of a structure are called fields.

! symbolic link:

n. This is a type of directory which is actually a link to another region of the disk. cd'ing to a symbolic link takes you into the directory which the link is to.

! user:

n. The person who is using the program.

It is important to design programs that are user-friendly, i.e. they do not do unexpected things on bad user input. While programming always remember the very applicable McGruff catch-phrase, "Users are losers", and plan accordingly. :-)

! variable:

n. A variable is an identifier in a C program which is the name for a "box" in memory which has certain contents. A variable has a type (refering to what kind of data is stored in the variable) as well as a value, which is the actual data that is stored in the box. Every variable used in a C program must have a declaration.

! whitespace:

n. Characters such as tabs, newlines and blanks. In C whitespace is ignored by the compiler (except inside of strings), thus allowing for programs to be formatted according to good visual style.

! word:

n. On most modern computers this is defined as 4 bytes. In general it is the size of the integer data type, or the size of a register on the microprocessor.

****