16.070 Introduction to Computers and Programming

February 7 Recitation 1 Spring 2002

Topics:

Logistics

- Sign-up
- Schedule
- 16.070 Website
- Problem Sets (see "Workstations" for more!)
- Use of PCs

Workstations

- Problem Set Turn In
- Computer Locations
- Logging In
- Compiling

C Programming

- Compiler Mechanics
- Basic Program
- Simple Input/Output

Programming Guide

- Readability
- Commenting Code
- Variable Names
- Keywords in C
- Notes on C

Logistics

Sign-up

At lab sessions, you are required to register your presence with the TA present. Please don't leave the sessions without being signed in!

Schedule

Lectures: MWF at 2PM in 3-270

Recitations: Classroom sessions on Thursdays at 2PM. Classroom recitations will be held in 2-190, and will cover important course information, additional material that might not be covered in class, and reinforcement/elaboration of material that was covered in class.

Lab sessions: Mondays at 3PM and 4PM, and Tuesdays at 9AM, 12PM, 1PM, 2PM, 3PM and 4PM. By Friday at 2PM you will have indicated which of the Lab session slots clash with your curriculum. All lab sessions will be similar during any one week, i.e. attend only your assigned one. Only official changes to assignments are accepted, to be approved by Dr. Fesq, Kay Sullivan, or Louis Breger.

Lab sessions will be held in 33-202, a Win2k-Cluster. These sessions will comprise of example programming problems that will augment your programming skills and also guide you towards solving the course's compulsory problem sets. Each tutorial session will be concluded with a 10 minute discussion of the solutions to the problems. Lab problem solutions will be handed out at the end of lab and posted on the course website by 6PM on Tuesdays.

16.070 Website

The 16.070 website is at http://web.mit.edu/16.070/www/. Course information will be posted on this website, including announcements, handouts, Handy Board information and links to C programming sites. Be sure to check the site often. Problem sets, Lectures, Lab problems, Recitations, etc. may be found under the Handouts tab. Please spend a few minutes of your time becoming familiar with the page. Since this is after all a course about software and computers, we will be relying heavily on this page.

Always keep a look-out for announcements under Announcements (link from bottom center of main page)! Essential information such as problem set revisions, emergency help, surveys, etc. will all be posted here.

In addition there is a link to a "Muddy Points" (MP) section, under the Student Resources tab. MP is our way of receiving feedback regarding your understanding of the lectures and recitations. Please complete this section after each lecture whenever you would like to provide feedback. All comments are anonymous and we take your inputs very seriously. This helps us to gauge our performance and your understanding, thereby maximizing your benefit from this course during each lecture. Responses to MP inputs will be posted on the web.

Problem Sets (see "Workstations" for more!)

Problem sets will be posted to the web on Wednesdays. Your solutions to the problem sets' questions need to be returned at the start of class, on the due date indicated on each problem set. Problem sets are normally returned on Wednesdays, usually a week after they were handed out.

Each problem includes a Turn In requirement, listed at the end of the problem. Please follow the instructions carefully, since your grade will depend upon it.

Each separately numbered problem needs to be started on a separate page (regardless of the size of the answer), since each problem within the Problem set will be returned separately. There will be a return box for each problem, in class, on the problem set due date. Place each problem in its assigned box when returning your problem set. If your solution to a problem covers multiple pages, then staple together all pages belonging to a specific problem, and remember to write your name on each page. Do not staple all problems together.

Early Turn In on the due date of the problem set will be possible in the graduate TA office 17-010. If the office is not open, slide your work under the door.

Late Turn In will **ONLY** be accepted if you approach Louis Breger or Kay Sullivan before the problem set is due and the arrangement is accepted. All problem sets returned late without a prior arrangement will be marked down 1/3 of the total original mark for every window of 24 hours that it is late.

Solutions to problem sets will be posted before midnight of the first Monday after turn in.

Some problem sets require you to turn in C code in electronic format. We require you to use a naming convention when turning in files. Your Athena username makes up the first part of all file names that you hand in. This is then followed by "_PS*n*" with *n* indicating the problem set number and then "_P*x*" where *x* indicates the problem number. So let's say your Athena email address is mabaker@mit.edu, and you are turning in a C file for problem 2 of problem set 4, then the filename would be ***MABAKER_PS4_P2.c***. Any questions? If a different convention is required, it will be explicitly stated in the problem.

Indicate how long you spent working on each problem of the problem set at the top of the first page of each problem. Please remember to include this information, since it is used in your interest.

Use of PCs

You are allowed to and encouraged to work on the programming sections of problem sets using your own PC. Laptops are ideal and can even be brought to the Lab sessions. If a PC is not available to you, the computer lab in 33-202 should be considered as your first option. Other areas with computers you can use are the Aero-Astro 1st floor lounge, the Aero-Astro library, and the mezzanine of the hangar area in building 33. Don't postpone your programming tasks until the last minute (esp. at the end of a term), since there are many more students than available PCs in the department...

To ensure that your data is readily available and the network does not get bogged down, save all information on your designated server directory space only (as opposed to the desktop). Your designated server space will be on the Aero-Astro server (18.34.0.143).

For Lab sessions you might want to bring along your Laptop if you have one. You will need to register for DHCP at http://web.mit.edu/is/help/dhcp/ and bring along an Ethernet cable to connect to the network in 33-202. There are a sufficient number of power outlets and network drops in the room.

The TAs will support MS-Visual C as the compiler of choice, with MS-Windows Win2k as the operating system of choice. Experienced programmers may however decide to use a different operating system and compiler, without official support. This may work, but will require significantly more effort in the second half of the course when Windows specific libraries are provided.

For the first problem set, there may not be enough computers available for all the people in the class, so people with Athena skills and programming experience are encouraged to do their homework on Athena. Also **for the first problem set only**, if you are working on Athena or are unable to access the homework turn-in folder on Aero-Astro, you can email your homework answers to lbreger@mit.edu as attachments. Email homework turn-in will still be due at 2PM on Wednesday, January 13th unless prior arrangements have been made.

MS-Visual Studio (C) is available for installation to all 16.070 students from the server \\Aero-Astro (IP = 18.34.0.243). The Student Resources tab of the web page has detailed Visual C install instructions for a variety of Windows operating systems.

For new users logging into machines in bldg 33:
Username: Same as Athena
Password: to be given in recitation or contact Gerry <nayden@mit.edu>

Workstations

Problem Set Turn In

Some problem sets will require you to hand in program code (see Problem Set section above). Electronic turn in is accomplished by copying the .c code file (see naming convention above) into the directory "16.070HW" of the server "Aero-Astro" (IP = 18.34.0.243). Problem set turn in requirements will indicate if you need to copy a file to a specific sub-directory under **\\Aero-Astro\16.070HW**.
You can see all files on the 16.070HW directory, but can only access the contents of files that you saved in the directory and cannot delete or rename any files. You cannot execute any file within the directory either.

Be sure to save only your final copy in this directory, since you cannot over-write even your own file and we want to stick to the naming convention discussed above. If for some reason you do make a mistake, save your updated file to the directory with "_1" or "_2" appended to the end of the file name to indicate a newer revision. Only your highest numbered revision will be graded.

In addition, **you cannot save directly to this directory from within Visual C**. You need to copy and paste your file's to this directory.

How to access \\Aero-Astro\16.070HW\:
From the Aero&Astro Lab. 33-202:
Click *Network Icon*, click *Entire Contents*, click *Microsoft Windows Network*, click *Aa-design*, click Aero-Astro, select desired directory. By the time you get to the design lab you will probably see an icon on the desktop, which links directly to the 16.070HW directory.

From a dorm room (using Windows98 or WindowsME):
Click *Start* button, click *Run*, type \\18.34.0.243 to get to the Aero-Astro server. Be sure that you are logged in locally (at your PC) with the same login name as you have on the server for this to work. Also, your local workgroup should NOT be "AA-Design" (a Windows peculiarity). You cannot remotely log into a PC in building 33 until you have changed your default password.

From a dorm room (using WindowsNT4 or Windows2000):
Same as for Windows 98, but you don't need to be logged using the same username as you have on the server. A peculiarity here is that "Network Neighborhood" or "My Network Places" should be enabled to find the server. You cannot remotely log into a PC in building 33 until you have changed your default password.

From off-campus:
Same as from dorm rooms.

Editing
When writing programs you must use a simple text editor and not a word processing program such as Framemaker or Word. The MS-Visual C text editor is the simplest option, since it colorizes functions and keywords after compilation, thereby adding to the readability of your code. Save your code file with a '*.c*' extension. This tells MS-Visual Studio that you wish to make use of the C compiler when compiling.

Printing
Room 33-202 and most Athena clusters have printers where you can generate hardcopies for problem set turn ins. Whenever you are stuck in a room without a printer, you might think about printing to a postscript file, which can be sent to any Athena printer.

The text contents of console windows, which will typically provide your program outputs, should be captured by pressing the printscreen button to make a screen capture and then pasted into a word document or paint program.

Compiling
Visual Studio is available for installation from \\Aero-Astro\VisualStudio (IP = 18.34.0.243). Install the C compiler (Visual C) from here by choosing the following components during installation: Visual C++, Visual C++ Developer Studio, VC++ RunTime Libraries, VC++ Build Tools, Data Access and Tools. You can reach Aero-Astro by typing *\\Aero-Astro* (or \\18.34.0.243) in the "Run" box accessible from any windows "Start" menu. See the course web page if you require more information regarding installation.

C Programming

Compiler Mechanics

The text file that you write in C is known as *source code*. The function of the compiler is to turn your *source code* into an executable program. A compiler is made up of two parts: the compiler and the linker. The compiler converts your *source code* into machine language code and puts the results into an *object code file*. The *object file* is just a translation of your *source code* into machine language code, it is still missing some of the parts necessary to make it complete. S*tart-up code*, which acts as an interface between your program and the operating system, is then linked to the *object file* by the linker. The *object file* also contains instructions for the compiler to call certain functions from the library routines such as, **printf(), scanf()** or **sqrt()**. The *library routine code* for these functions is stored in C libraries and is linked to the *object file* by the linker. The compiler then uses the *object file, start-up code,* and *library routine code* to produce an executable program.

Basic C Program

```
/**********
* BTK     *
* 16.070  *
* Example *
***********/

#include <stdio.h>

int main(void)
{
  int num = 1;
  printf("We have only %d Earth.\n", num);
  printf("So we should keep it clean.");

  return 0;
} /* end main */
```

#include <stdio.h> includes the contents of the file stdio.h in your code when it is compiled. Stdio.h is a header file that contains the definitions for standard input/ouput function.

int main(void) is the main function. Every C program must have a main function. This is where the program begins its execution when it is run. The **int** means that the function **main** returns a value of integer type. The operating system expects your program to return an int, so always use "int main". The **void** means that main does not take any input. If information was being passed to the function, it would be inside those parentheses.

{…….} these braces enclose all of the expressions that are contained in the main functions. Notice how all of these expressions are indented from the other lines in the program. The braces mark the beginning and end of different blocks of code.

int num is a variable declaration. It says that the variable **num** will be of integer type and secures the required amount of computer memory for that variable.

num = 1 assigns the value of 1 into the computer memory that was allocated for the **num** variable.

printf() is a standard i/o function contained in the stdio.h file. The characters that are enclosed in parentheses are displayed on the screen. The **% d** is a place holder that tells the function to print an integer value its location. The **num** after the quotations determines what integer will be printed in that place. The **\n** causes a new line character to be placed at the end of that line of text.

return 0 is required in the main function because it must return an integer value.

Simple Input/Output

Printf and *scanf* are the basic output and input functions that print and read data from the standard input-output device. They are each defined in the header file *stdio.h*.

Printf

printf writes to the standard output device, usually the screen. The format is as follows:

```
printf(control_string, expressions_list …);
```

Example:

```
printf("Hello World, this is the year %d.\n",2001);
```

Generating the output string:

```
Hello World, this is the year 2001.
```

The control_string, which is enclosed in double quotes " ", is the text that is output to the screen. Variables can be incorporated into *printf* statements by using the % character followed by the format specifier for that variable. The expressions_list is a comma separated list of the variables that correspond to the format specifiers contained in the control_string. Special characters called escape sequences can be incorporated in order to print special characters such as tab, newline, etc…

Escape Sequence	Name	Action
\a	alert	sounds a beep
\b	backspace	backs up one character
\f	formfeed	starts a new screen or page
\n	newline	moves to beginning of next line
\r	carriage return	moves to beginning of current line
\t	horizontal tab	moves to next tab position
\v	vertical tab	moves down a fixed amount
\\	back slash	prints a back slash
\'	single quote	prints a single quotation mark
\"	double quote	prints a double quotation mark
\?	question mark	prints a question mark

Scanf

scanf reads in data from the standard input device, usually the keyboard. It is similar to the *printf* but not the same. The format is as follows:

```
scanf(control_string, pointer_list …)
```

Example:

```
scanf("%d", &IntegerInput);
```

Stores a user input (when followed by a carriage return) in the integer variable *IntegerInput*.

The control string, which is again enclosed in double quotation marks, specifies the formats of the data to be read using the format specifiers. If there is more than one variable to be read the format specifiers should be separated by a space inside the control_string. The pointer_list is a comma separated list of pointers to the variables that are being read in. The pointer format is *&variable_name*.

Format Specifier	Variable Representation
%d or %i	integer, actual width
%*w*d or %*w*i	integer, at least *w* digits wide
%c	single character
%s	string, actual width
%f	floating point, actual width (6 decimal digits by default)
%*w.d*f	floating point, at least *w* characters wide, and *d* decimal digits
%lf	double floating point (*scanf* only!!!)
%e or %E	floating point data in scientific notation
%*w.d*e or %*w.d*E	fl point, scient notation, at least *w* char wide, and *d* decimal digits
%%	prints the % symbol

- The format conversion specifiers must agree in number and type with the pointer variables.
- The field width (*w* or *w.d*) need not be, and shouldn't be, specified for *scanf*
- Blank spaces, tabs, and newline characters n the *input stream* are usually ignored, unless characters are being read.
- Blank spaces and tabs in the *control_string* are usually ignored, except as follows:
    - A leading blank space in front of a %c specification indicates that all blank space in the *input string* should be skipped until the next non-blank character is found and read as a character. If there is no space in front of the %c the next character will be read, even if it is a blank.
    - Any character in the control string other than a blank or a format specifier must match the next non-blank character in the input stream.
- *scanf* returns an integer of the number of fields that were successfully read. This can be useful in error checking.

Programming Guide

Proper style is important when writing programs. It allows others to decipher your code more easily and also makes it easier for you to make changes at a later date. Follow the style guide that is provided on the course website. You will lose marks for poor programming style. People who already have a programming style should learn the style elaborated on the course website and use it at least for duration of 16.070. Here are some important notes on style:

Readability

When you are programming you should make sure that your code is readable. It should be easy for you to go back and read and also easy for others to read. One important step to making code readable is **indenting**. Your program should be indented so that different blocks begin in different columns. Your programs should only contain one expression per line. Lastly, you should choose variable names with some meaning, this makes your program much easier to follow.

Compare:

```
if (a==5) {
b=1;
}
else if (a==6) {
if (a==3) {
b=2;
}
else b=3;
}
```

With:

```
if (a==5) {
  b=1;
}
else if (a==6) {
  if (a==3) {
    b=2;
  }
  else b=3;
}
```

Commenting Your Code

Comments that span multiple lines may be incorporated into your code by enclosing them in `/*` and `*/` comment delimiters. A line comment can be started with the `//` characters in C++, but should not be used in 16.070 in order to maintain compatibility with C. Comments should be used liberally when programming. There will be places that comments are required, but a good rule of thumb is not to comment any unnecessary code.

Variable Names

Variable names should be meaningful and descriptive, in order to make your programs more readable. They can consist of capital letters and lowercase letters, digits from 0 to 9 and the underscore character _. The first character must be a letter or an underscore. There is virtually no length limitation, it is only limited by the compiler which typically only recognize the first 32 characters as significant. There can be no whitespaces, and keywords cannot be used.

Keywords in C

C has several keywords that you may not use as variables names, function names or definitions. They are contained in the following table:

asm	auto	break	case
char	const	continue	default
do	double	else	enum
extern	float	for	goto
if	int	long	register
return	short	signed	sizeof
static	struct	switch	test
typedef	union	unsigned	void
volatile	while		

Notes on C

- *Structured programming* (readable, indented and well commented) is required for this class. Points will be deducted for obscure code.
- KISS! Keep It Straightforward and Simple, is a good acronym to remember.
- C is *case sensitive*
- All statements are terminated with a *semi-colon*
- Limit our use of *Global Variables*, they make code difficult to understand and even risky...
- *Comments* are treated like whitespace (they can appear almost anywhere)
- Assignment is done with "=" and equality is tested with "==".
- **Zero** is the natural starting point in C
    - Counts start with zero
    - Zero means false / not-zero means true
    - Array subscripts start at zero
    - Pointers use zero to designate a null value
    - External and static variables are initialized to zero by default
- **Parenthesis** (), **brackets** [], and **braces** {} always go in pairs, except in strings
- Most operations in C, including assignments, produce (or return) a value