Block 01: Thursday of first week

You need the textbook. If you don't have it, you're already behind,

so order it next-day delivery.

Download and install CodeBlocks MinGW for Windows:

NOTE THE "MINGW"! (CodeBlocks by itself doesn't include a C++ compiler,

you need both.)

http://sourceforge.net/projects/codeblocks/files/Binaries/13.12/Windows/codeblocks-13.12-setup.exe/download

Don't try to use a Macintosh for this class. No Code Warrior, no Xcode,

none of that is any good. Don't waste time. If you use a Mac as a

terminal to log into Buffy, the Linux mainframe, that's fine. If you have

problems getting your Macintosh to work, that is no surprise to anybody.

Macs are not worth the time and aggravation. If you are enthusiastic

over the platform, the logical move is for you to go to work for Apple.

NO student who has insisted on using their Mac has ever finished the

course. They wasted time trying to get it to work until they'd gotten

too far behind, then dropped or just froze up and got an "F". I do

talk about the Mac, because the expectations for modern GUIs stem from

that Super Bowl ad in 1984: Graphic User Interfaces are a fact of life.

But they're an accomodation for people too lazy to learn to type, to

use commands, to understand the nature of computing. If you start your

design with the assumptions of laziness and stupidity, you're going to

end up with something as spavined as Microsoft Windows. If you demand

competence from users, you'll maintain a standard as high as that of Unix.

Apple makes a lot of money from people's ignorance, but the productivity

of personal computer designs wasn't improved one whit. I say, good for

Apple, BUT: programmers cannot afford to be ignorant.

"If you make something idiot proof, they'll build a better idiot."

<OPINION>

If you must deploy GUI software, Visual Basic if the most painless

way.

If you need performance (you're in the game industry) then you WILL use

Visual C++ CLR/CLI and the DIRECTX libraries. No option. You'll also

use OPEN GL, since DIRECTX is as unrealiable in consistency as all

Mifrosoft, and Open GL is open source. This is why if your World

of Warcraft game won't play with your graphics card, you can enable

the Open GL side of it and keep playing...a little slower. So you'll

be learning those libraries.

BEST CHOICE: Java Swing. Unlike every dialect of C, Java has fixed

variable sizes, and a standard GUI model. Also, it is possible to

migrate between platforms.

</OPINION>

USING BUFFY, THE LINUX HOST:

DEMO PROGRAMS:

To run demo programs you must log into Buffy.

READ THE system\_access FILE AT:

http://209.129.16.61/~hhaller/data/cisc192/modules/system\_access

Be careful following the directions. They work, the file has no errors,

but commands must be correct. If you can't log in, slow down and keep

trying until you can.

REMEMBER: PASSWORDS ARE NOT ECHOED. THAT MEANS: YOU CANNOT SEE YOURSELF

TYPE YOUR PASSWORD.

The most important DEMO for this block is waterpump.cpp. It demonstrates

character-based "filter" i/o, and is a skeleton we will use over and

over again, so it merits memorization.

There are compiled executables:

waterpump.cpp compiles to waterpump

Run programs by typing their names, then when the screen is waiting,

type in lines of text and see what the system does. To halt input,

press ^D. (CTRL-D). (CTRL-Z on Windows.)

Don't press ^Z on a Unix box. It will put the current program

into the Twilight Zone. You can background it with bg, or

bring it back to the foreground with fg, but in the Twilight

Zone, it is not executing.

====================LECTURE NOTES==============================

This class practices Writing Programs.

Programs are lists of Instructions that are executed by Computers.

Computers are unliving things. We must be very specific in how we

instruct them. They cannot handle shades of meaning.

SOME OF THEM CAN CHANGE THEIR PROGRAMS AS THEY RUN! We've been building

them for at lest 30 years, and we don't yet know now to program them.

That is called "Self-Modifying Code" or "Von Neumann Code". If you want

to try it, in August download Core War and study it. Buy Excedrin.

http://en.wikipedia.org/wiki/Core\_War

BUT FOR NOW: we'll only try to write "static" program code: it

loads, it runs without changing itself.

So what is a "command"? An "instruction"?

They're numbers. The computer CPU is designed so that commands are numbers

contained in bytes.

POINT TO REMEMBER: When we're working inside the machine, or with raw

data from memory, the familar Base-10 system is cumbersome. The Base-2

system the computer uses isn't much better, it puts one to sleep with

long strings of 1's and 0's. If we need to deal with addresses and memory

contennts, we use Base 16, Hexidecimal. You should already know it from

CISC 181.

You can do arithmetic with any number system. There is no STRUCTURAL

difference between PROGRAMS and DATA.

All data is numeric. A lower case 'a' is really a 97. A space ' ' is a 32.

If you want to look at data, use my hexdump utility. The Unix od program

isn't as flexible. When I wrote hexdump in 1987, I was still programming

IBM mainframes. It's a recreation of the IBM utility.

Here's a dump of the above paragraph using my program:

HEXDUMP Originally Written in Turbo C by TEK, 1987.

00000000: 49 66 20 79 6F 75 20 77 - 61 6E 74 20 74 6F 20 6C - If.you.want.to.l

00000010: 6F 6F 6B 20 61 74 20 64 - 61 74 61 2C 20 75 73 65 - ook.at.data,.use

00000020: 20 6D 79 20 68 65 78 64 - 75 6D 70 20 75 74 69 6C - .my.hexdump.util

00000030: 69 74 79 2E 20 20 54 68 - 65 20 55 6E 69 78 20 6F - ity...The.Unix.o

00000040: 64 20 70 72 6F 67 72 61 - 6D 0A 69 73 6E 27 74 20 - d.program.isn't.

00000050: 61 73 20 66 6C 65 78 69 - 62 6C 65 2E 20 20 57 68 - as.flexible...Wh

00000060: 65 6E 20 49 20 77 72 6F - 74 65 20 68 65 78 64 75 - en.I.wrote.hexdu

00000070: 6D 70 20 69 6E 20 31 39 - 38 37 2C 20 49 20 77 61 - mp.in.1987,.I.wa

00000080: 73 20 73 74 69 6C 6C 20 - 70 72 6F 67 72 61 6D 6D - s.still.programm

00000090: 69 6E 67 0A 49 42 4D 20 - 6D 61 69 6E 66 72 61 6D - ing.IBM.mainfram

000000a0: 65 73 2E 20 49 74 27 73 - 20 61 20 72 65 63 72 65 - es..It's.a.recre

000000b0: 61 74 69 6F 6E 20 6F 66 - 20 74 68 65 20 49 42 4D - ation.of.the.IBM

000000c0: 20 75 74 69 6C 69 74 79 - 2E 0A - .utility..

See? It's a little foreign. When you need to know it, it'll take you

about an afternoon to get comfortable with it. You just count 0-9 and

A-F, then 10. Simple. Don't worry about the specifics, just

remember the concept. I'll be referring to specific characters

almost immediately.

--------------

To build a program:

Get an Idea

Express it as a series of Steps

Simplify the Steps as much as possible

Convert the steps from English to C++

Compile (translate) the C++ "Source Code" into computer "Object Code".

(fix Syntax Errors and repeat as needed)

Run the program

(check for Logic Errors: rethink, recode, repeat until results are right.)

Consider how to improve the program.

Now: read the first chapter of the Text on how to play with program code in CodeBlocks.

"Building and Executing your First Application"

Please do it now. I'll wait.

(Q.Q)

l | l

. \_ .

Now that you're back, let's get hands-on and play with some code.

Let's do this as we read, please. It'll make things flow later.

Bring up CodeBlocks.

(START->ALL PROGRAMS->CodeBlocks)

Click Create a New Project

Click Console Application

Click Go

Click Next

Click C++

Click Next

Name it guessnum

Decide where to put it. Click the square with the elipsis [...] and navigate to your USB drive. When the

file dialog comes up click on "Computer", then the E: or F: or whatever the volume name of your thumb drive is,

you can click "make new directory" and call it guessnum, that keeps things orderly.

Click Next

Click Finish

If it helps, paste this into your Diary. ("Keep a Diary!") (If you hear loud mooing, you misspelled Diary.)

At the top menu bar, click View -> Focus Management Panel -> click the '+' next to Source, then DOUBLEclick main.cpp.

Now you see the EDITOR pane. This is were we Create.

You'll notice that CodeBlocks has already written a program for you. Let's play with it.

Select Build on the top menu bar,

Click Build in the drop-down menu.

Look at the bottom (long) pane, below the Editor. This is where the Message appear. If you make an error, the

compiler will print errors here. Reading errors is not as hard as reading heiroglyphs. In time you'll make sense

of all that, but for now you should see: "status 0". 0 is Good.

Look at the toolbar below the Menu Bar. You will find a green "play" icon. Click it once. The program will

run in a pop-up window. wonderful. After you're done looking at it, hit ENTER, it will vanish.

Now, go back into the Editor, mouse to the top, left-click-drag to the bottom of the Editor, and press

the DEL key. The program code should vanish.

(Now would be a good time to put the coffee on, or pour a cup.)

We will Enter, Compile, and Run the program below.

OK, Here's the program source code, fire up CodeBlocks, create a project,

enter this and get it running:

PLEASE take the time to type this in, don't cut and paste it. There are

no shortcuts. I realize that's heresy, but I don't know any working

programmers who took the shortcuts. Some muscles need to get worked.

As you enter the code, note the syntax-colors in the Editor. Sweet.

/\*-------------------------------------------------------------------

\* guessnum.cpp - guess the number

\* (Your Name) (date)

-------------------------------------------------------------------\*/

#include <cstdio>

#include <cstdlib>

#include <iostream>

#include <ctime>

using namespace std ;

int main ()

{

srand(time(NULL)) ; // "seed" the randomizer

int limit = 1001 ; // set a range limit

int secret = rand() % limit ; // % is the "remainder" operator

int max\_tries = 10 ; // the challenge

int guess ;

int trial = 0 ; // count the guesses here

LOOPSTART:

trial = trial + 1 ;

if (trial > max\_tries )

{

cout << "Too many guesses: you lose.\n" ;

return -1 ;

}

cout << "Guess: " ;

cin >> guess ;

// be super nice to the user, give hints:

if (guess < secret)

cout << "Low\n" ;

else

if (guess > secret)

cout << "High\n" ;

else

cout << "RIGHT!!\n" ;

if (guess != secret)

goto LOOPSTART ; // I know, goto is a dirty word, don't nag.

cout << "Number of tries: " << trial << endl ;

cin.get() ;

return 0 ; // (0 is good)

} // main

========================================end of listing=========================

Click Build -> Build

If you don't get a clean compile, (status 0), look at the FIRST ERROR MSG,

note the line number, look at that line in the Editor, compare it (or the

line above it) with the original listing.

(Most errors are caused by the line above the one that confuses the compiler.)

(Computers are stupid.)

ALWAYS FIX THE FIRST ERROR. (See line above.)

Once it's up and running, start the program.

Play the game.

Think about it.

Optional:

Continue to play the game until you can win it every time you play.

Be prepared to explain your strategy for always winning in 10 tries or less.

If your fellow students want to argue, can you come up with a

justification for never needing 11 tries?

Hint: if you can couch it in mathematical terms it always shuts 'em up.

(Fans of Car Talk may have an advantage with this puzzle.)

Here's a sample run:

BUFFY:hhaller:~/data/cisc192/block00$ ./a.out

Guess: 250

High

Guess: 125

High

Guess: 75

Low

Guess: 90

Low

Guess: 115

High

Guess: 100

High

Guess: 95

High

Guess: 93

Low

Guess: 94

RIGHT!!

Number of tries: 9

=========================================================================

1.1 RECAP

Programs are instructions the computer follows.

Computers cannot understand human language. Not even the human languages

C or C++. We write in the human language C++, then the computer runs a

Compiler to translate our Source Code into Object Code so it can run.

Here is how:

Design the logic.

Write the source code.

Compile the source code.

Run the object code.

Example:

1. Use an editor, create and save the program hello.cpp

#include <cstdio>

int main()

{

printf("Hello, world\n") ;

getchar() ;

}

2. Compile it with: g++ hello.cpp or Build->Build (command line vs Code::Blocks)

3. Run the object file with: a.out or RunIcon (command line vs Code::Blocks)

4. See the result. Pres ENTER to end run.

That is how you program. Different compilers and systems have simpler

or more complicated procedures, but the principles don't change.

We will take the next few weeks and learn the basics. After that,

we'll learn the new features that made C into C++. We'll only touch the

surface of the "Object Oriented Programming" side of C++, it's too huge.

HARD FACT:

Young programmers are not hired to create new software unless they work

in game design. In general, they start as maintenance programmers,

maintaining and enhancing old programs. This means that knowing C is

what will get you your first job. That's good, since you won't understand

the reason for Object Oriented Programming features until you are working

on large, complex programs. Otherwise, they're frightfully bureaucratic

and inefficient. GOOD PROGRAMMING IS LEAN, SPARE, AND FAST.

C++ is a Stealth Bomber. C is a Bowie Knife.

We will concentrate on the C core of C++ to start. The class notes will

contain what you need to concentrate on.

You'll want to know the command line, Code::Blocks, Microsoft Visual

Studio, and probably several others, so you should play with them all,

and get to know them enough to jump right into them on the job. For now,

learn one well. The second one should be the Unix command line, since

Unix machines are where all the real databases and peocesses live.

Everybody knows Windows. If you know Unix, you're more promotable.

================EXAMPLE PROGRAMS FROM THE TEXT=====================

1.2 VARIABLES AND ARITHMETIC EXPRESSIONS //STORE //CALCULATE

Consider this program:

1 #include <stdio.h>

2 /\* print Fahrenheit-Centigrade table

3 for fahr = 0, 20 ..., 300 \*/

4

5 int main()

6 {

7 int fahr, centigrade ;

8 int lower, upper, step ;

9

10 lower = 0 ;

11 upper = 300 ;

12 step = 20 ;

13

14 fahr = lower ;

15 while ( fahr <= upper)

16 {

17 centigrade = 5 \* (fahr - 32) / 9 ;

18 printf("%d\t%d\n", fahr, centigrade) ;

19 fahr = fahr + step ;

20 }

21 }

Here's what it prints:

0 -17

20 -6

40 4

60 15

80 26

100 37

120 48

140 60

160 71

180 82

200 93

220 104

240 115

260 126

280 137

300 148

Variables are names for addresses in computer storage.

We store values there so we can use them in programs.

They have different types:

char character - a single byte

int integer - a whole number

double double-precision floating point number

different sizes:

short

long

other characteristics:

short

long

long long

signed (signed int)

unsigned (unsigned char)

static (static long int )

automatic

extern

register (register int loopctr )

C programmers have more control than, say, BASIC programmers about the

size and other characteristics of variables. It's important to know all

that, because if you don't use them correctly, you'll get wrong answers.

(Examine the program above. Try to figure out what it

does as you type it into your editor.

Learn to identify "reserved words" in a programming language:

include

int

main

while

Be sensitive to punctuation and syntax:

If you see a hash: '#', the next word is a command for the Preprocessor,

which reads in files and edits your source before the compiler gets it.

<> The brokets (broken brackets?) hold the name of a file to include.

= The value on the right is assigned to the variable on the left.

() The contents of the parens are resolved, then used as a single value.

; A statement ends here. (Many statements may be on a single line, but it's ugly.)

<= The value on the right of the <= is being compared with the value

on the left. If the left is less than or equal to the right, the

expression's value is one(true), otherwise, it's zero(not true).

x \* y the expression value is the product of x times y

x / y the expression value is the quotient of x divided by y

x - y the expression value is the difference between x and y

You get the idea.

Here's an older way to print to the screen than "cout << ":

printf()

It's so versatile the Java community demanded it be added to Java 5.1.

(Seriously, they got mean about it.) It's really all you need to know

to print anything. It's very powerful, and worth learning. It's how

to get printing to the screen, any way you like, with only one tool.

printf("Good morning, %s\nYour coffee is coming.\n", user\_name) ;

Good morning, Bob

Your coffee is coming.

printf("%7.2f", 802.701 );

802.70

The "%7.2f" is the FORMATTING.

%c single char

%d int

%u unsigned int

%f double

%lf long double

%s string of characters

%x hexadecimal number

We control the printing with "escape characters".

\n New line (carriage return)

\t tab

\b backspace

\a bell

1.3 The for statement.

#include <stdio.h>

/\* print Fahrenheit-Centigrade table

for fach = 0, 20 ..., 300 \*/

int main()

{

int fahr ;

for (fahr = 0 ; fahr <= 300 ; fahr += 20)

{

printf("%3d %6.1f\n", fahr, (5.0/9.0 \* (fahr-21))) ; // line 13

}

}

This prints:

0 -11.7

20 -0.6

40 10.6

60 21.7

80 32.8

100 43.9

120 55.0

140 66.1

160 77.2

180 88.3

200 99.4

220 110.6

240 121.7

260 132.8

280 143.9

300 155.0

Note: Line 13 shows a printf() with multiple formatted terms.

Note: line 13 shows how terms and mathematical operations can be

combined, and their results printed as single values.

(You notice that if you give a program the wrong logic, it prints bad answers, right?)

Can you correct the code above? (Anybody take General Science?)

SYMBOLIC CONSTANTS

This is another use for the Preprocessor:

#include <stdio.h>

#define LOWER 0

#define UPPER 300

#define STEP 20

int main()

{

int fahr ;

for (fahr = LOWER ; fahr <= UPPER ; fahr += STEP)

{

printf("%3d %6.1f\n", fahr, (5.0/9.0 \* (fahr-21))) ;

}

}

It's not encouraged any more, because it has interesting errors

it's prone to. Instead, we're told to use "const" VARIABLES

instead of preprocessor MACROS.

const int LOWER = 0 ;

const int UPPER = 300 ;

//s.4.c

#include <stdio.h>

#define LOWER 0

#define UPPER 300

#define STEP 20

int main()

{

int fahr ;

for (fahr = LOWER ; fahr <= UPPER ; fahr += STEP)

{

printf("%3d %6.1f\n", fahr, (5.0/9.0 \* (fahr-21))) ;

}

}

//1.5.c

#include <stdio.h>

main()

{

int c ;

double nc ;

c = getchar() ;

for (nc = 0 ; getchar() != EOF ; ++nc)

;

printf("%.0f\n", nc) ; /\* num chars read \*/

}

//1.5.3.c

/\*count lines in input \*/

#include <stdio.h>

main()

{

int c, nl ;

nl = 0 ;

while (( c = getchar()) != EOF)

{

if (c == '\n')

{

++nl ;

}

}

printf("lines: %d\n", nl) ;

}

//1.5.4.c

/\* count words \*/

#include <stdio.h>

#define IN 1 /\* inside a word \*/

#define OUT 0 /\* outside a word \*/

int main()

{

int c, nw, state ;

state = OUT ; /\* we aren't in a word yet \*/

nw = 0 ;

while ( (c = getchar() ) != EOF)

{

if (c == '\n' || c == '\t' || c == ' ' )

{

state = OUT ;

}

else if (state == OUT)

{

state = IN ;

++nw ;

}

}

printf("words: %d\n", nw) ;

}

/\* 1.7.c - test power function \*/

#include <stdio.h>

int power (int m, int n) ;

int main()

{

int i ;

for (i = 0 ; i < 10 ; ++i)

{

printf("%d %d %d\n", i, power(2,i), power(-3,i)) ;

}

return 0 ;

}

/\* power: raise base to n-th poer; n >= 0 \*/

int power(int base, int n)

{

int i, p ;

p = 1 ;

for (i = 1 ; i <= n ; ++i)

p = p \* base ;

return p ;

}

//1.9.c

/\*

char arrays

\*/

#include <stdio.h>

int main()

{

char buffer[81] , i ;

printf("Type your first, middle, last names: ") ;

fgets(buffer, sizeof(buffer)-1 , stdin) ;

puts("You typed: ") ;

puts(buffer) ;

}

For the nerves:

Either version of Sanislaw Lem's Solaris is artistic and mind altering.

The Russian Tartkovskij's version is great, but Mosfilm didn't make

it clear how much of a love story this was. I became a Cluny fan from

this movie. And the music.... Let this lie on your head a bit....

https://www.youtube.com/watch?v=bLV6JrygUb0

/\*-----------------------------------------------------

\* multiply.cpp - a first example of programming

\* T.E. Harrisburg, Army Security Agency

Sat May 31 21:28:12 PDT 2014

-----------------------------------------------------\*/

#include <iostream>

using namespace std;

int main()

{

int a, m, p ;

cout << "Enter a whole number: " ;

cin >> a ;

m = 0 ;

looptop:

p = a \* m ;

cout << a << " \* " << m << " = " << p << "\n" ;

m = m + 1 ;

if (m < 13)

goto looptop ;

}

/\*-----------------------------------------------------------------

\* vartest.cpp - how is data stored in different types?

\* th Wed Aug 21 11:34:41 PDT 2013

\* revision 0

-----------------------------------------------------------------\*/

#include <cstdio>

#include <cctype>

#include <cstdlib>

#include <iostream>

#include <iomanip>

using namespace std ;

int main ()

{

// show the results of math with typed constants:

cout << "33 / 2 = " << 33 / 2 << "\n" ; // ints

cout << "33.0 / 2.0 = " << 33.0 / 2.0 << "\n" ; // doubles

// show the results of math with typed variables:

int i ;

i = 33.0 / 2.0 ;

cout << "i = " << i << "\n" ;

double d ;

d = 33.0 / 2.0 ;

cout << "d = " << d << "\n" ;

return EXIT\_SUCCESS ;

}

/\*-----------------------------------------------------------------

\* waterpump.cpp - pump data through

\* you'll use this a LOT, so run it and type

\* into it. Really, it's time well spent.

NOTICE: we use c = cin.get() instead of cin >> c, because

we want "raw" input, we want to read and store ALL chars,

including end of line ('\n') and space in general.

cin >> uses whitespace as DELIMITERS or word boundaries,

so they don't get stored and printed out.

This illustrates the "high level language" issues, and

why we prefer to work down low in C country. Real programmers

don't want the tools to do their thinkng for them.

-----------------------------------------------------------------\*/

#include <cstdio>

#include <iostream>

using namespace std;

int main ()

{

char c , total = 0 ;

c = cin.get() ;

do

{

cout << c ;

c = cin.get() ;

} while (! cin.eof()) ;

return 0 ;

}

Char Dec Oct Hex

. 0 000 0x000

. 1 001 0x001

. 2 002 0x002

. 3 003 0x003

. 4 004 0x004

. 5 005 0x005

. 6 006 0x006

. 7 007 0x007

. 8 010 0x008

. 9 011 0x009

. 10 012 0x00a

. 11 013 0x00b

. 12 014 0x00c

. 13 015 0x00d

. 14 016 0x00e

. 15 017 0x00f

. 16 020 0x010

. 17 021 0x011

. 18 022 0x012

. 19 023 0x013

. 20 024 0x014

. 21 025 0x015

. 22 026 0x016

. 23 027 0x017

. 24 030 0x018

. 25 031 0x019

. 26 032 0x01a

. 27 033 0x01b

. 28 034 0x01c

. 29 035 0x01d

. 30 036 0x01e

. 31 037 0x01f

32 040 0x020

! 33 041 0x021

" 34 042 0x022

# 35 043 0x023

$ 36 044 0x024

% 37 045 0x025

& 38 046 0x026

' 39 047 0x027

( 40 050 0x028

) 41 051 0x029

\* 42 052 0x02a

+ 43 053 0x02b

, 44 054 0x02c

- 45 055 0x02d

. 46 056 0x02e

/ 47 057 0x02f

0 48 060 0x030

1 49 061 0x031

2 50 062 0x032

3 51 063 0x033

4 52 064 0x034

5 53 065 0x035

6 54 066 0x036

7 55 067 0x037

8 56 070 0x038

9 57 071 0x039

: 58 072 0x03a

; 59 073 0x03b

< 60 074 0x03c

= 61 075 0x03d

> 62 076 0x03e

? 63 077 0x03f

@ 64 100 0x040

A 65 101 0x041

B 66 102 0x042

C 67 103 0x043

D 68 104 0x044

E 69 105 0x045

F 70 106 0x046

G 71 107 0x047

H 72 110 0x048

I 73 111 0x049

J 74 112 0x04a

K 75 113 0x04b

L 76 114 0x04c

M 77 115 0x04d

N 78 116 0x04e

O 79 117 0x04f

P 80 120 0x050

Q 81 121 0x051

R 82 122 0x052

S 83 123 0x053

T 84 124 0x054

U 85 125 0x055

V 86 126 0x056

W 87 127 0x057

X 88 130 0x058

Y 89 131 0x059

Z 90 132 0x05a

[ 91 133 0x05b

\ 92 134 0x05c

] 93 135 0x05d

^ 94 136 0x05e

\_ 95 137 0x05f

` 96 140 0x060

a 97 141 0x061

b 98 142 0x062

c 99 143 0x063

d 100 144 0x064

e 101 145 0x065

f 102 146 0x066

g 103 147 0x067

h 104 150 0x068

i 105 151 0x069

j 106 152 0x06a

k 107 153 0x06b

l 108 154 0x06c

m 109 155 0x06d

n 110 156 0x06e

o 111 157 0x06f

p 112 160 0x070

q 113 161 0x071

r 114 162 0x072

s 115 163 0x073

t 116 164 0x074

u 117 165 0x075

v 118 166 0x076

w 119 167 0x077

x 120 170 0x078

y 121 171 0x079

z 122 172 0x07a

{ 123 173 0x07b

| 124 174 0x07c

} 125 175 0x07d

~ 126 176 0x07e

. 127 177 0x07f

. 128 200 0x080

. 129 201 0x081

. 130 202 0x082

. 131 203 0x083

. 132 204 0x084

. 133 205 0x085

. 134 206 0x086

. 135 207 0x087

. 136 210 0x088

. 137 211 0x089

. 138 212 0x08a

. 139 213 0x08b

. 140 214 0x08c

. 141 215 0x08d

. 142 216 0x08e

. 143 217 0x08f

. 144 220 0x090

. 145 221 0x091

. 146 222 0x092

. 147 223 0x093

. 148 224 0x094

. 149 225 0x095

. 150 226 0x096

. 151 227 0x097

. 152 230 0x098

. 153 231 0x099

. 154 232 0x09a

. 155 233 0x09b

. 156 234 0x09c

. 157 235 0x09d

. 158 236 0x09e

. 159 237 0x09f

. 160 240 0x0a0

. 161 241 0x0a1

. 162 242 0x0a2

. 163 243 0x0a3

. 164 244 0x0a4

. 165 245 0x0a5

. 166 246 0x0a6

. 167 247 0x0a7

. 168 250 0x0a8

. 169 251 0x0a9

. 170 252 0x0aa

. 171 253 0x0ab

. 172 254 0x0ac

. 173 255 0x0ad

. 174 256 0x0ae

. 175 257 0x0af

. 176 260 0x0b0

. 177 261 0x0b1

. 178 262 0x0b2

. 179 263 0x0b3

. 180 264 0x0b4

. 181 265 0x0b5

. 182 266 0x0b6

. 183 267 0x0b7

. 184 270 0x0b8

. 185 271 0x0b9

. 186 272 0x0ba

. 187 273 0x0bb

. 188 274 0x0bc

. 189 275 0x0bd

. 190 276 0x0be

. 191 277 0x0bf

. 192 300 0x0c0

. 193 301 0x0c1

. 194 302 0x0c2

. 195 303 0x0c3

. 196 304 0x0c4

. 197 305 0x0c5

. 198 306 0x0c6

. 199 307 0x0c7

. 200 310 0x0c8

. 201 311 0x0c9

. 202 312 0x0ca

. 203 313 0x0cb

. 204 314 0x0cc

. 205 315 0x0cd

. 206 316 0x0ce

. 207 317 0x0cf

. 208 320 0x0d0

. 209 321 0x0d1

. 210 322 0x0d2

. 211 323 0x0d3

. 212 324 0x0d4

. 213 325 0x0d5

. 214 326 0x0d6

. 215 327 0x0d7

. 216 330 0x0d8

. 217 331 0x0d9

. 218 332 0x0da

. 219 333 0x0db

. 220 334 0x0dc

. 221 335 0x0dd

. 222 336 0x0de

. 223 337 0x0df

. 224 340 0x0e0

. 225 341 0x0e1

. 226 342 0x0e2

. 227 343 0x0e3

. 228 344 0x0e4

. 229 345 0x0e5

. 230 346 0x0e6

. 231 347 0x0e7

. 232 350 0x0e8

. 233 351 0x0e9

. 234 352 0x0ea

. 235 353 0x0eb

. 236 354 0x0ec

. 237 355 0x0ed

. 238 356 0x0ee

. 239 357 0x0ef

. 240 360 0x0f0

. 241 361 0x0f1

. 242 362 0x0f2

. 243 363 0x0f3

. 244 364 0x0f4

. 245 365 0x0f5

. 246 366 0x0f6

. 247 367 0x0f7

. 248 370 0x0f8

. 249 371 0x0f9

. 250 372 0x0fa

. 251 373 0x0fb

. 252 374 0x0fc

. 253 375 0x0fd

. 254 376 0x0fe