

# CS100 Introduction to Programming

Recitation 1

<TA NAME>

# NO PLAGIARISM!!!

- The most likely cause for failing this course.
- You WILL be caught!
- We WILL punish!
- They WILL know!
  - Parents
  - University
  - School
  - Fellows

# Administrative affairs recap

- 2 Quizzes
- 8 Homework
- Recitation attendance
- Mid term
- NO final
- No mandatory text book

# Outline

- NO Review
- Variables
- Functions
- Basic IO
- Control structures
- Code quality?
- Setting up environment

Review

# NO REVIEW

- Concepts will be taught by lectures
  - We will not repeat what professors have made clear
- Recitation will focus on
  - Examples
  - Specific problems
    - e.g. `a+++++a`?
  - Details not covered in lectures
    - e.g. How to use `printf`?
  - Answer your ~~weird~~ interesting questions

This will be valid for all recitation sessions.  
If you were distracted during lecture,  
there is no way to make up the loss.

Variables

# Declaration

```
#include <stdio.h>
```

```
int foo = 1;
```

```
int main() {  
    int foo = 3, bar = 0;  
    static persist = 9;  
    printf("Foo is: %d\n", foo);  
    printf("Bar is: %d\n", bar);  
    return 0;  
}
```

`static`: Persistence?

`int`: type, e.g. integer, boolean

`foo`: name, reference this variable by this name

`= 1` : initializer, give it a default value, can be done later

**Uninitialized variable should not be used!**



# Scoping

```
#include <stdio.h>
int foo = 1;

int test(int foo) {
    return -foo;
}

int main() {
    int foo = 3, bar = 2;
    bar = test(bar);
    printf("Foo is: %d\n", foo);
    printf("Bar is: %d\n", bar);
    return 0;
}
```

What is the output? Why?

# Operators

# Precedence

- Ask your questions

\*Some are not C but C++.

2	a++ a-- type() type{} a() a[] . ->	Suffix/postfix increment and decrement Functional cast Function call Subscript Member access	
3	++a --a +a -a ! ~ ( type) *a &a	Prefix increment and decrement Unary plus and minus Logical NOT and bitwise NOT C-style cast Indirection (dereference) Address-of	Right-to-left
4	.* ->*	Pointer-to-member	Left-to-right
5	a*b a/b a%b	Multiplication, division, and remainder	
6	a+b a-b	Addition and subtraction	
7	<< >>	Bitwise left shift and right shift	
8	<=>	Three-way comparison operator (since C++20)	
9	< <= > >=	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
10	== !=	For relational operators = and ≠ respectively	
11	&	Bitwise AND	
12	^	Bitwise XOR (exclusive or)	
13		Bitwise OR (inclusive or)	
14	&&	Logical AND	
15		Logical OR	
16	a?b:c throw co_yield = += -= *= /= %= <<= >>= &= ^=  =	Ternary conditional <sup>[note 2]</sup> throw operator yield-expression (C++20) Direct assignment (provided by default for C++ classes) Compound assignment by sum and difference Compound assignment by product, quotient, and remainder Compound assignment by bitwise left shift and right shift Compound assignment by bitwise AND, XOR, and OR	Right-to-left
17	,	Comma	Left-to-right

# Trivia: Assignment operators

- Assignment operator, along compound assignment operators, is an expression, not statement, and can be evaluated to a value.

```
int first, next;  
first = next = 1;
```

- Strongly discouraged
  - Prone to typos.
  - Reduce code readability

# Trivia: Undefined behaviors (i.e. UB)

- C standard is not exhaustive on how certain actions should behave.
- The behavior of such action will vary greatly depending on OS, compiler and hardware.
- Compilers are free to blow you up if you step on the land mine called UB.
- e.g.
  - Attempt to self increment/decrement twice or more between two sequence points: `a = (a++) + (a++); printf("%d %d %d", a++, a++, a++);`
  - .....

# Functions

# Declaration & Implementation

- File: math.h

```
int square(int base, int exponent);
```

```
int abs(int n);
```

```
int neg(int n);
```

- File math.c

```
#include "math.h"
```

```
int square(int base) {  
    return base * base;  
}
```

```
int abs(int n) {  
    if (n >= 0) {  
        return n;  
    } else {  
        return neg(n);  
    }  
}  
  
// .....
```

# Why?

- To use function declared later
- Prevent duplicate implementation
  - main.c include utils.c and math.c, utils.c include math.c.
  - Preprocessors include math.c twice! BOOM!
- Declare an interface for others to use
  - So when others are linking against your library, they don't need to compile your library, while knowing what you provided.



Basic I/O

# Something worth noting

- Some concepts are not introduced yet
  - They are still listed for the sake of completeness
- We will only show you that some concepts exists
  - We won't walk you through step by step, as we have very little time.
- Man page is more exhaustive
  - Don't worry if you missed some part of scanf/printf

# Input

- `getc` and its friends
  - Less common
- `scanf()`
  - `scanf("Some not captured, some are %s", string_buffer)`
  - Whitespace in formats stands for one or more whitespace.
  - Cannot directly handle non deterministic number of captures (i.e. array)
  - `%%`
  - Complete syntax (some parts will not be cover here)

`%[*][' ][m][length][mod]type`  
`%                  3      1     u`  
This captures an unsigned long at most 3 characters wide.

# scanf: Type specifier

- Common ones:
- %d Signed decimal integer
- %i Signed integer in hexadecimal, octal or decimal
- %u Unsigned integer
- %f Floating number
- %X Unsigned hexadecimal
- %s String
- %c A single char

# scanf: Modifiers (All optional)

- Length modifier: Maximum characters to read
  - e.g. %7s reads a 7 character wide string
  - e.g. for input 1234abc, scanf(“%3i%3i%5s”, &i1, &i2, s) will set i1 to 123, i2 to 4 and s to abc
- Type modifier: One or two “l” or “L” between format and % mark
  - Typically double the size of base type.
  - e.g. %lu reads an unsigned long instead of unsigned
  - e.g. %lf reads an double instead of float
- Discard: an asterisk (\*) immediately after % mark
  - This placeholder will match but won’t capture.
  - e.g. for input 1234abc, scanf(“%\*3i%s”, s) will set s to 4abc

# scanf: Examples

- 123 abc 998 => int a, c; char \*b;. What is the format string?
  - scanf(“%d %s %d”, &a, b, &c);
- 12345ab123. Format string is “%4lu%s23”. What will happen?
  - First will become 1234, second will become 5ab123
- 12345ab1 23. Format string is “%4lu%s23”. What will happen?
  - First will become 1234, second will become 5ab1

# Output

- `putc()` and its friends
  - Less common
- `printf()`
  - `printf("%s is %d%% not hard.", "C/C++", 100);`
  - Cousin of `scanf`
  - We do not present everything, see man page for an exhaustive list
  - Complete format syntax:  
    `%[flags][width][.precision][length]type`  
    %    +        4        .        3            1        u  
    This prints an unsigned long to at least 3 digit long and left-padded with whitespace to least 4 character.

# printf: Precision

- Use with floating point numbers
  - Digits after the decimal point
  - e.g. `printf("%.3f", 100.0f);` yields `100.000`
- Use with integer
  - At least that many digits written. Left pad with 0.
  - e.g. `printf("%.3d %.3d", 10, 1000);` yields `010 1000`
- Use with string
  - At most that many character
  - e.g. `printf("%.3s", "ABCDEFGH");` yields `abc`



# printf: Width

- At least that many characters are written
- Insufficient -> left pad with whitespace
  - e.g. `printf("%4d", 100);` yields " 100"
  - e.g. `printf("%2d", 100);` yields "100"

# Printf: flags

- -: left align
  - right align is default
- +: always print the sign
  - Note: +/- is not functionally opposite.
- " ": Prepend a space if positive, otherwise the same as "+"
  - e.g. 10 becomes " 10", -10 becomes "-10"
  - Ignored if + in place.

# Control Structures

# if-else: Conditional execution

- Recall your summer homework
- It don't have to be a comparison

```
#include <stdio.h>

int main() {
    int i = 0;
    if (i) {
        printf("executed");
    }
    return 0;
}
```

# switch: Not too many else if

- Too many else if can be ugly and inefficient
- Limitations:
  - Only switch on integers, no strings
- Remember to break!
- Remember to add default!
- Some other languages have much more powerful version of switch.  
See [Pattern Matching](#)

# switch: Not too many else if (cont.)

```
#include <stdio.h>

int main() {
    int i = 0;
    scanf("%d", &i);
    switch (i) {
        case 0:
            printf("Hi binary 0!");
        case 1:
            printf("Hi binary 1!");
    }
    printf("Glad you only know 1 and 0!");
    return 0;
}
```

- What's wrong?

# Switch or if-else?

- Rule of thumb:
  - more than 3 case => switch.
  - Otherwise => if-else
- Who is faster?
  - Smaller switch will be transformed into if-else by compiler.
    - e.g. a single-case-switch is very likely to be turned into an if.
  - *Has Qing Dynasty ceased to exist?*
    - 2019 CPUs can execute  $\sim 10^9$  instructions per second
    - The difference between two is mostly less than 100 cycles.
    - *Ma Yun picking up a sesame.*

# While: loop over a condition

- While some expression is true, do something
- Typical usage: “main loop”

```
#include <stdio.h>

int main() {
    int i;
    while (true) {
        scanf("%s", &i);
        switch (i) {
            case 0:
                return 0;
            case 1:
                printf("Hi 1!");
                break;
            default:
                printf("Did you say %d?", i)
        }
    }
}
```



# For: enhanced loops

```
for (int i = 1; i < LENGTH ; ++i) {  
    go(i);  
}
```

- Evaluation sequence?
- Don't off by 1!
- ++i or i++?
- C89?

# Anti pattern: assignment in condition

- Be cautious when using assignment in if/while/for conditions

```
while (x = read_an_int()) {  
    foo(x);  
}
```

- What if you accidentally typed this instead:

```
while (x == read_an_int()) {  
    foo(x);  
}
```

# Nesting

- All control structures can be nested in any other structure indefinitely many times.
- Too many levels signal bad design.
  - Use functional decomposition
  - Extract portion into functions
  - Otherwise
    - Hard to read, it could go outside of the right bound of the screen.
    - Hard to maintain, as it obscures the control flow.

# Nesting: HAZARD

- What is this monstrosity doing?

```
for (int i = 0; i < n; i++) {  
    foo(i);  
    if (...) {  
        for (int j = i; j < n; j++) {  
            if (test(j))  
                for (int k = j; k > i ; k++) {  
                    bar(k);  
                }  
            }  
        }  
        break;  
    }  
}
```

Code quality?

# Coding style: Why important?

- You want codes like this:

```
1  #include <stdio.h>
2  int main(){
3      printf("Hello, world!\n");
4      return 0;
5  }
```

Not this:

```
1  #include <stdio.h>
2  int
3  main(){printf(
4      "Hello, world!\n");return 0;}
```

- And of course not this:

```
1  #include <stdio.h>
2  int main(){int a=0,aa=1,A,AA;for(AA=0;AA<10;++AA,A=a,a=aa,aa=aa+A,printf("%d\n",a));}
```

# Coding style: Why important?

- Makes code easier to read for others.
- Easier to read for yourself!  
Otherwise, you will soon lose track if you start on a project that you haven't worked on for a while.
- Makes code consistent when co-operating with other people.
- (Also makes code aesthetically pleasing)

# Coding style: Why important?

- Rule at thumb:
- Pretend somebody else is reading your code.
  - This guy will be yourself three month later.



# WARNING

Some code fragments we are about to show  
is from what students have submitted.

No one will know who wrote these, though.

# Naming variables

```
int main()
{
    int a, b;
    float A, c, B, d;
    A = a % 12;
    d = A * 30;
    c = 1.0 * b / 2;
    A = d + c;
    B = b * 6;
```

Do not use void main()!  
Use int main() instead

```
void main()
{
    int a, b;
    float A, B, C;
    char c;
    scanf("%d%c%d", &a, &c, &b);
}
```

# Naming variables

- Good variables names:
  - Reflects its value
  - Eliminates ambiguity
- Examples:
  - Good names:  
totalOfAllSelectedItem, totalOfAllUnselectedItem, a\_very\_long\_variable\_name, ...
  - Bad names:  
a, b, c, d... aa, aaaa, A, NAME, O0oOllL1l, ...

# Naming variables

- Don't worry if it gets too long!
- Solutions:
  1. add underscores(\_) or hyphens(-).
    - `score_CS100_Shanghaitech_students`
  2. camelCase (小驼峰命名法)  
Capitalize the first letter of each word except the first one
    - `variableNamingUsingCamelCase`
- TIP: We do not recommend any variable name which begins with a UPPERCASE LETTER!

# Naming schemes

- Snake case
  - `this_is_a_variable`
  - `it_has_no_upper_case_letter`
- Little camel case
  - `g_globalVariable`
  - `thisIsAnIntegerVariable`
  - Some prefer to prepend several letters to indicate the variable type
- Big camel case
  - `ThisIsAFunction`
  - Do not use this for variables. Only functions or classes (C++).

# Commenting

- Meaningful comments
  - Key implications, side effects
  - Known issues
  - Complicate calculus/control flow/binary magic/magic number
  - Regular expressions
- Not transliteral of your code
  - No: `"z=x*y; // This line multiples x by y and store it in z"`
  - Yes: `"z=x*y; // This calculate the total and store it in z"`
  - Better: `"total = quantity * price;"`
    - Good naming remove the need of commenting!
    - Most inexperienced developers don't know how to do so though.....

# Whitespaces

```
if(op=='+')c=a+b;
if(op=='-')c=a-b;
if(op=='*')c=a*b;
if(op=='/')c=a/b;
printf("%f",c);
```

```
float a,b,c,d,e,f,g;
char c;
printf("What time is it?\n");
scanf("%f%c%f",&a,&c,&b);
if(a>12&&a<=24)
{
    a=a-12;
    g=(a/12)*360;
    d=(b/60)*360;
    e=(b/60)*30;
    f=g+e-d;
    f=fabs(f);
    if(f>180)
```

# Whitespaces

- Before & after some operators like +, -, ==, >, and =.
- After any comma(,).

```
int a = 3, b = 4, c = 5;
```

```
float a,b,c,d,e,f,g;
char c;
printf("What time is it?\n");
scanf("%f%c%f",&a,&c,&b);
if(a>12&&a<=24)
{
    a=a-12;
    g=(a/12)*360;
    d=(b/60)*360;
    e=(b/60)*30;
    f=g+e-d;
    f=fabs(f);
    if(f>180)
```

```
float a, b, c, d, e, f, g;
char c;
printf("What time is it?\n");
scanf("%f%c%f", &a, &c, &b);
if(a > 12 && a <= 24)
{
    a = a - 12;
    g = (a/12)*360;
    d = (b/60)*360;
    e = (b/60)*30;
    f = g + e - d;
    f = fabs(f);
    if(f > 180)
```



# Whitespaces

- Not convinced? Look at this:
  - Which of the residual calculation expressions look cleaner?

```
for(int i=0;i<15;++i){  
    double t_i=(8.0-i-1.0)/2.0;  
    residual=x1*exp(-x2*(t_i-x3)*(t_i-x3)/2.0)-y;  
}
```

```
for (int i = 0; i < 15; ++i) {  
    double t_i = (8.0 - i - 1.0) / 2.0;  
    residual = x1 * exp(-x2 * (t_i - x3) * (t_i - x3) / 2.0) - y;  
}
```

# Indenting

```
int main()
{while(ju=='y')

    {
        printf("Enter the arithmetic expression\n");
        scanf("%f%c%f",&a, &op, &b);

        /*
         *      SOME
         *      CODES
         *      HERE
        */

        printf("\nDid you want to enter another expression (y/n)");
    }

    return 0;
}
```

```
printf("The next number is 1\n");
for(int i=0;i<88;++i){

    last=a+b;
    printf("The next number is %d\n",last);
    a=b;
    b=last;

}
return 0;
}
```

# Indenting

- Organize your code blocks with indents.
- A general rule: an open bracket( { ) → increase an indent  
a closed bracket( } ) → decrease an indent
- An indent can be a tab, two, four or eight spaces.

```
int main()
{while(ju=='y')

    {
        printf("Enter the arithmetic expression\n");
        scanf("%f%c%f",&a, &op, &b);
        /*
         *      SOME
         *      CODES
         *      HERE
        */
        printf("\nDid you want to enter another expression (y/n)");
    }

    return 0;
}
```

```
int main()
{
    while(ju == 'y')
    {
        printf("Enter the arithmetic expression\n");
        scanf("%f%c%f",&a, &op, &b);
        /*
         *      SOME
         *      CODES
         *      HERE
        */
        printf("\nDid you want to enter another expression (y/n)");
    }

    return 0;
}
```

# Own line for curly braces?

- There are two styles when you open a curly brace:
- In a new line:
- In the previous line:

```
int main()
{
    if(1)
    {
        for(;;)
        {
            while(true)
            {
                doSomething();
            }
        }
    }
}
```

```
int main(){
    if(1){
        for(;;){
            while(true){
                doSomething();
            }
        }
    }
}
```

# Own line for curly braces?

- There is NO right or wrong for this question!
- It's totally personal preference!
- ◆ Once again, although there are a lot of debates, this is a TRIVIAL problem!
- ◆ However, it is strongly recommended and often mandatory to follow the style guide of your project.
- ◆ Stay consistence with yourself and your surroundings.

We don't allocate a line for braces because there  
is insufficient line space on the slides

# Some misuse

- Can be confusing!

```
for(i = 1;i; ++i)
```

- Comma?

```
b = 10;  
for(a = 1; b < 500, b = b+c)  
{  
|
```

# “for loops”

- `for(i = 0; i < 100; ++i)`  
    initialization      condition      increment
- For loop is best used when you have a certain start, end, and step.
- Otherwise, try using while loops instead.
- Also, it's best to initialize to 0 and use < in condition.  
    (We count from 0, not 1.)

# “goto” is evil!

- Goto statement is considered unsafe, as it might mess up the structure of your code.
- When you are about to write a “goto”, write a while loop instead!

```
if (E == 'y')
{
    goto select1;
}
```

Unless absolutely necessary, don't use it.

```
if (op == 'v'){
    goto 18;
}
if (op == 'n'){
    return 0;
}
else{
    if ( op == '-'){
        printf("%f\n",i
        printf("Did you
        scanf("%c",&op)
        getchar();
        if (op == 'v'){
            goto 18;
        }
    }
}
```



# Exercise (NOT A QUIZ)

- Calculate the sum of the first 30 non-prime Fibonacci number.
  - Add 30 number together
  - It doesn't have to be super efficient. Just correctness is enough.
- Answer: 432949606
- Code smell:
  - Your code has compile warnings.
  - Your code does not have a variable named "sum".
  - Your code does not have a dedicated `is_prime` function.
  - Your code has no comment whatsoever.
  - Your code has no comment in `is_prime` function.

# Reference implementation

- Header: calc.h

```
int is_prime(int n);
```

```
int calculate(int n);
```

- Main should never be included in the header.
- You can merge `calculate` into `main`.
  - This will make it harder for code reuse though.

# Reference implementation

```
1. int is_prime(int n) {  
2.     int divisor;  
3.     /* A brute force search through all numbers  
4.     smaller than n to find a divisor */  
5.     for (divisor = 2; divisor < n; divisor++) {  
6.         /* test if a divisor */  
7.         if (n % divisor == 0) {  
8.             return 0;  
9.         }  
10.    }  
11.    return 1;  
12. }
```

# Reference implementation

```
13. int calculate(int n) {
14.     int sum = 0, first = 1, last = 1, counted = 0;
15.     while (counted < n) {
16.         /* calculate the next element in F sequence */
17.         int next = first + last;
18.         if (!is_prime(next)) {
19.             counted++;
20.             sum += next;
21.         }
22.         /* Advance our position... */
23.         first = last;
24.         last = next;
25.     }
26.     return sum;
27. }
```

# Some coding styles

- Google: <http://google.github.io/styleguide/>
  - Python: <https://www.python.org/dev/peps/pep-0008/>
  - Airbnb: <https://github.com/airbnb/javascript/blob/master/README.md>
- 
- Most modern languages define their own style.
  - Popular languages got more than one.

# Last: Do not copy others' codes!

- Even open-source codes are forbidden in this course.
- Please read rules of academic conduct carefully!

```
#include <stdio.h>int main(){    const int ARRSIZE=5050, DISPCNT=5000; //定义数组大小, 显示位数    //c
```

-----

版权声明：本文为CSDN博主「chisir2000」的原创文章，遵循CC 4.0 by-sa版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/chisir2000/article/details/79781281>

Setting up environment

# A word of advice

- No IDE is better in all regards.
- No IDE suits all needs.
- Don't start yet another editor war.
- If you prefer another editor, feel free to use it.
  - VSCode is taken only because all TAs have experience installing/using it.
- No IDE isn't death.
- Notepad is death.



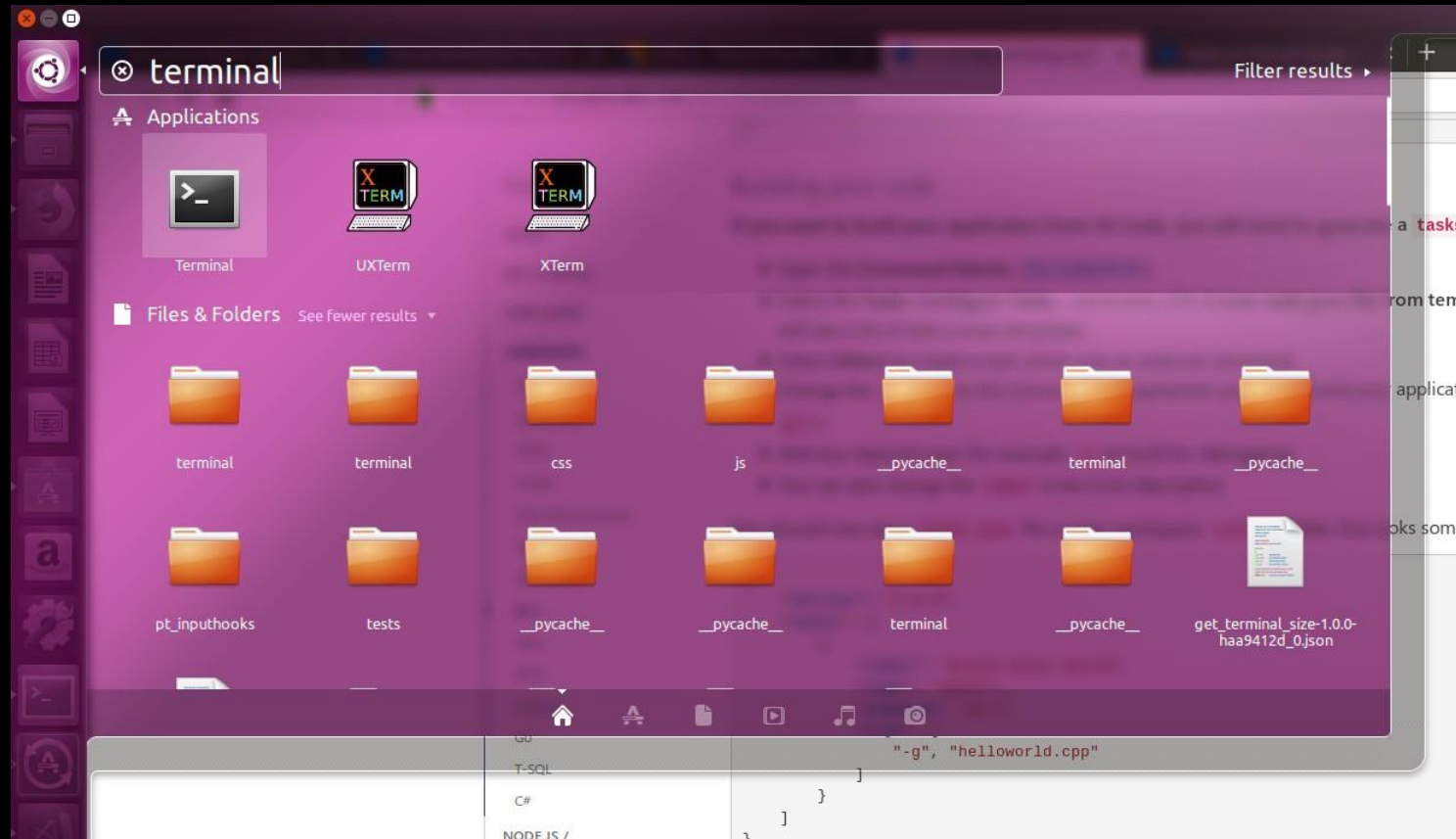
# Operating System?

- For this course, all OSes are the same.
- Sooner or later, a SIST student will need a machine with Linux.
- So, what to do with your Windows PC?
  - Wipe it clean and install Linux?
  - Dual boot?
  - Virtual machine?
  - WSL?
  - A new computer dedicated to Ubuntu?

**Installation of compiler...**

# ... under Ubuntu

- Open a terminal



## ... under Ubuntu

- Install build-essentials using apt-get

```
sudo apt-get update && sudo apt-get install build-essentials -y
```

# ... under Ubuntu

- Too slow? Use a mirror!

<http://mirrors.geekpie.club/ubuntu/>

```
sed -i 's/[^/]+\u005Cu/mirrors.geekpie.club\u005Cu/' /etc/apt/sources.list
```

... under Ubuntu

- Verify that the compiler is working by entering

*gcc -v*

# ... under OSX

- Install Xcode

## Step #1: Install Xcode on a Apple Mac OS X

First, make sure Xcode is installed. If it is not installed on OS X, visit [app store](#) and install Xcode.

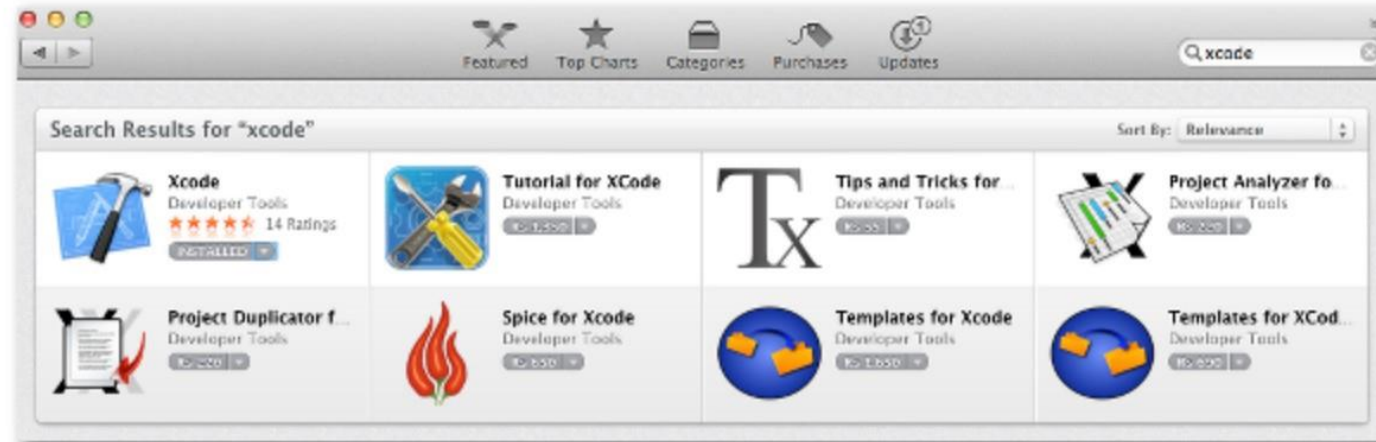


Fig.01: Make sure Xcode developer tools are install OS X

# ... under OSX

- Install  
command--line  
tools

## Step #2: Install gcc/LLVM compiler on OS X

Once installed, open Xcode and visit:

Xcode menu > Preferences > Downloads > choose "Command line tools" > Click "Install" button:

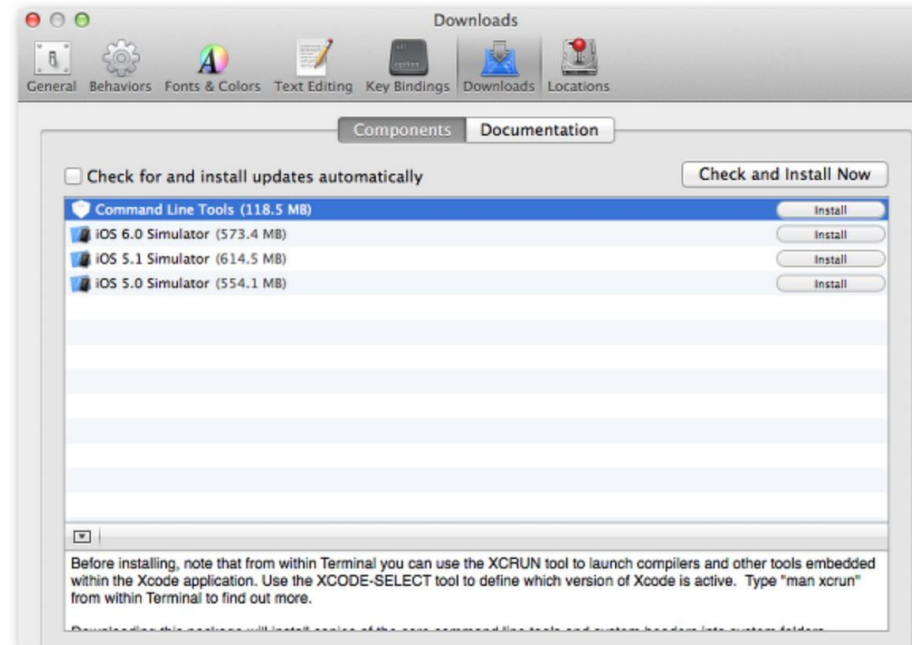
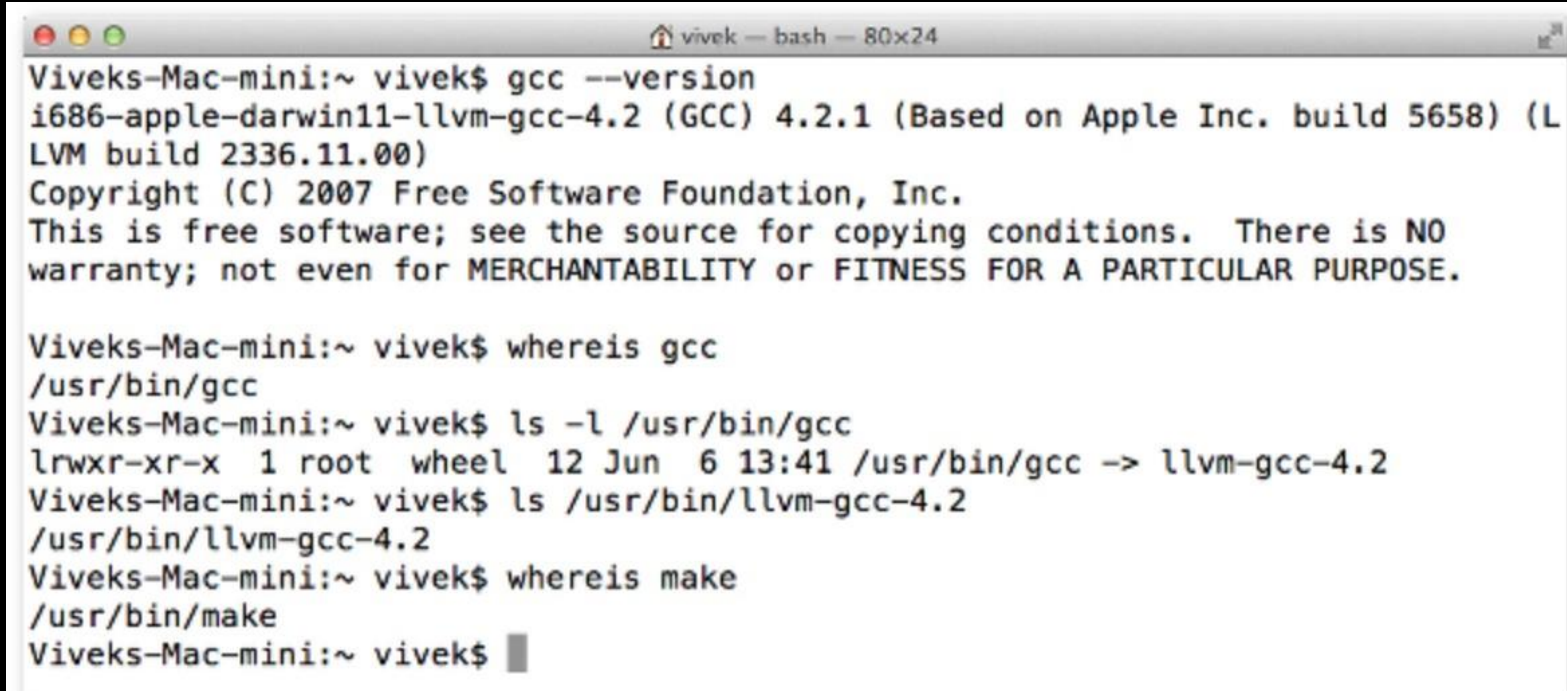


Fig.02: Installing gcc compiler on Mac OS X



# ... under OSX

- Verify that it is working

A screenshot of a macOS terminal window titled 'vivek — bash — 80x24'. The window shows the output of several commands used to verify the installation of the gcc compiler. The commands and their outputs are as follows:

```
Viveks-Mac-mini:~ vivek$ gcc --version
i686-apple-darwin11-llvm-gcc-4.2 (GCC) 4.2.1 (Based on Apple Inc. build 5658) (L
LVM build 2336.11.00)
Copyright (C) 2007 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Viveks-Mac-mini:~ vivek$ whereis gcc
/usr/bin/gcc

Viveks-Mac-mini:~ vivek$ ls -l /usr/bin/gcc
lrwxr-xr-x  1 root  wheel  12 Jun  6 13:41 /usr/bin/gcc -> llvm-gcc-4.2

Viveks-Mac-mini:~ vivek$ ls /usr/bin/llvm-gcc-4.2
/usr/bin/llvm-gcc-4.2

Viveks-Mac-mini:~ vivek$ whereis make
/usr/bin/make

Viveks-Mac-mini:~ vivek$
```

Fig.03: Verify gcc compiler installation on Mountain Lion OS X

## ... under Windows

- We recommend MinGW (Minimalist GNU for Windows)
- Download link:

<https://sourceforge.net/projects/mingw/files/latest/download>

- Detailed installation instructions:

[http://www.mingw.org/wiki/Getting\\_Started](http://www.mingw.org/wiki/Getting_Started)

# ... under Windows 10

- Set PATH variable
  - When you install command line tools, such as [MinGW](#), or GnuWin32 tools, you have to tell the command line interpreter where to find them. This is usually accomplished by adding the appropriate directory names to the *PATH* variable in your user environment. The installers will *not* do this for you.
  - **NOTE** If you choose to alter your PATH variable, you must ensure you alter your *user* PATH variable, *not* your system PATH variable –there are two of them!
  - Click search bar. Enter “Advanced System Setting” even if you use Chinese locale. Choose the one looks correct. It should appear before all three words are typed in. In opened dialog box, click “Environment variable” button.
  - You should be presented with a dialog box with two text boxes. The **top** box shows your *user* settings. The PATH entry in this box is the one you want to modify. Note that the bottom text box allows you to change the *system* PATH variable. You **should not** alter the system path variable in any manner, or you will cause all sorts of problems for you and your computer!
  - Click on the PATH entry in the TOP box, then click on the "Edit" button
  - Click new. Enter this or use browse button to select the exact folder  
  
<installation--directory>\bin
  - press OK →OK →OK and you are done.

# ... under Windows (not Win 10)

- Set PATH variable
  - The same considerations as Win 10
  - Right-click on your "My Computer" icon and select "Properties".
  - Click on the "Advanced" tab, then on the "Environment Variables" button.
  - You should be presented with a dialog box with two text boxes. The **top** box shows your **user** settings. The PATH entry in this box is the one you want to modify. Note that the bottom text box allows you to change the **system** PATH variable. You **should not** alter the system path variable in any manner, or you will cause all sorts of problems for you and your computer!
  - Click on the PATH entry in the TOP box, then click on the "Edit" button
  - Scroll to the end of the string and **at the end add**  
  
;<installation--directory>\bin
  - press OK →OK →OK and you are done.

# ... under Windows

- Other choices?
  - Cygwin
    - Bulky
    - More functions
  - MSYS2
    - Less bulky
    - Fewer functions, e.g. no mmap.
  - Both of above have on campus mirror.

<https://mirrors.geekpie.club/>

**Installation of git ...**

## ... under Ubuntu

- *sudo apt-get install git*

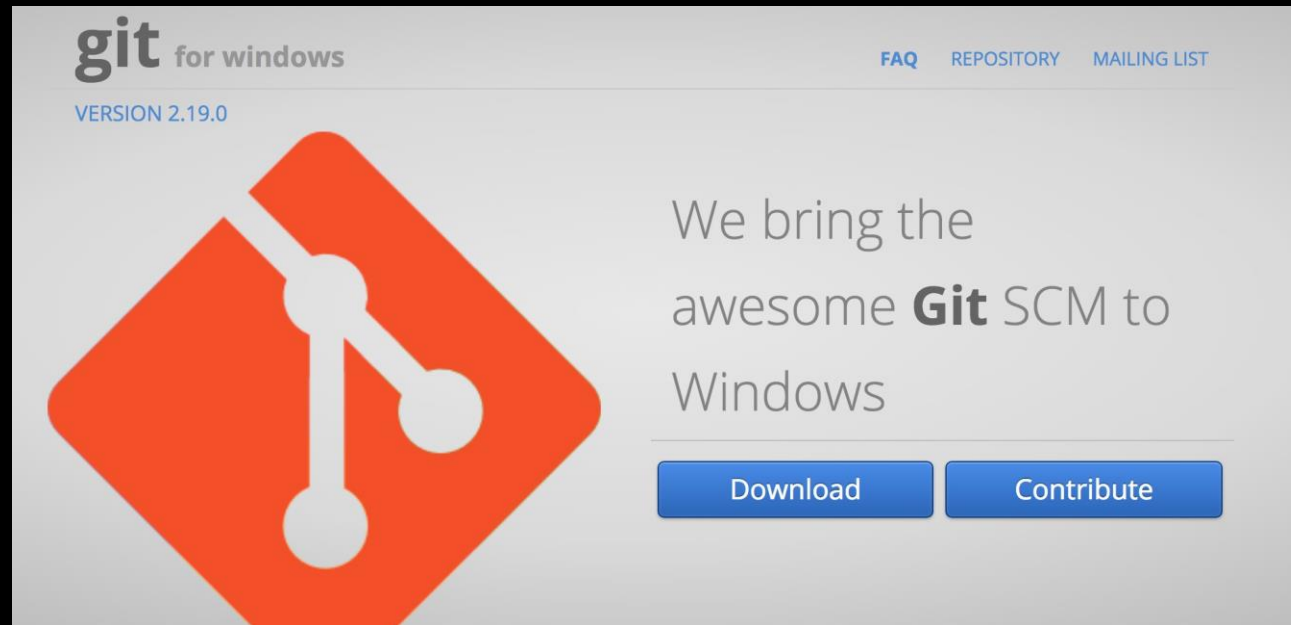
... under OSX

- Install brew (package manager)
  - *ruby -e"\$(curl -fsSL <https://raw.githubusercontent.com/Homebrew/install/master/install>)" brew doctor*
- Install git
  - *brew install git*



# ... under Windows

- Download and install “git for windows”:
  - <https://gitforwindows.org/>
  - <https://mirrors.geekpie.club/git-for-windows/>



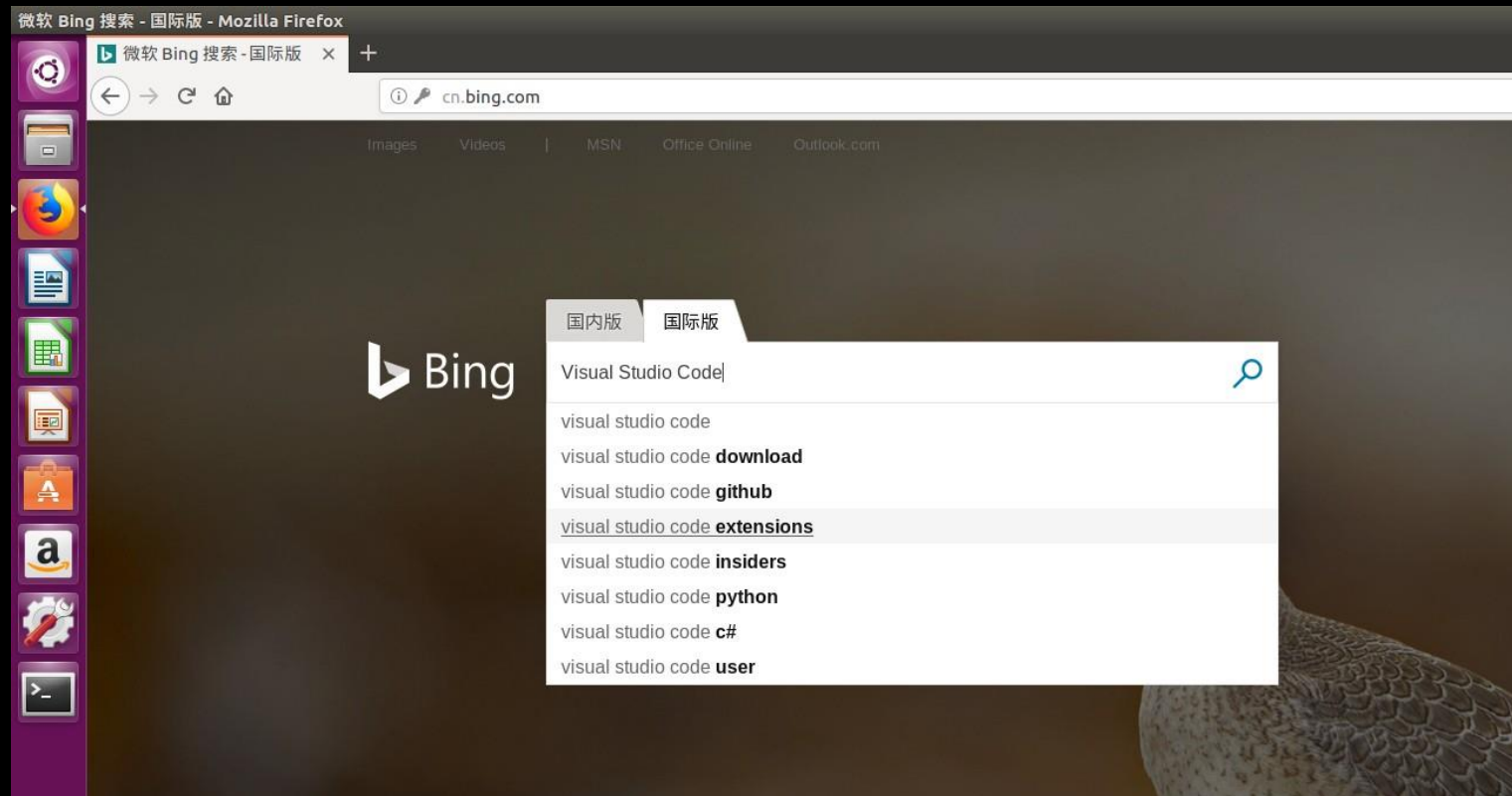
# Installation of Visual Studio Code

# Visual Studio Code

- Code UI
  - Light weight
  - Cross platform
  - Multiple language support
  - Cross platform
- The present is for installation under Ubuntu, the procedure is similar under other OS!

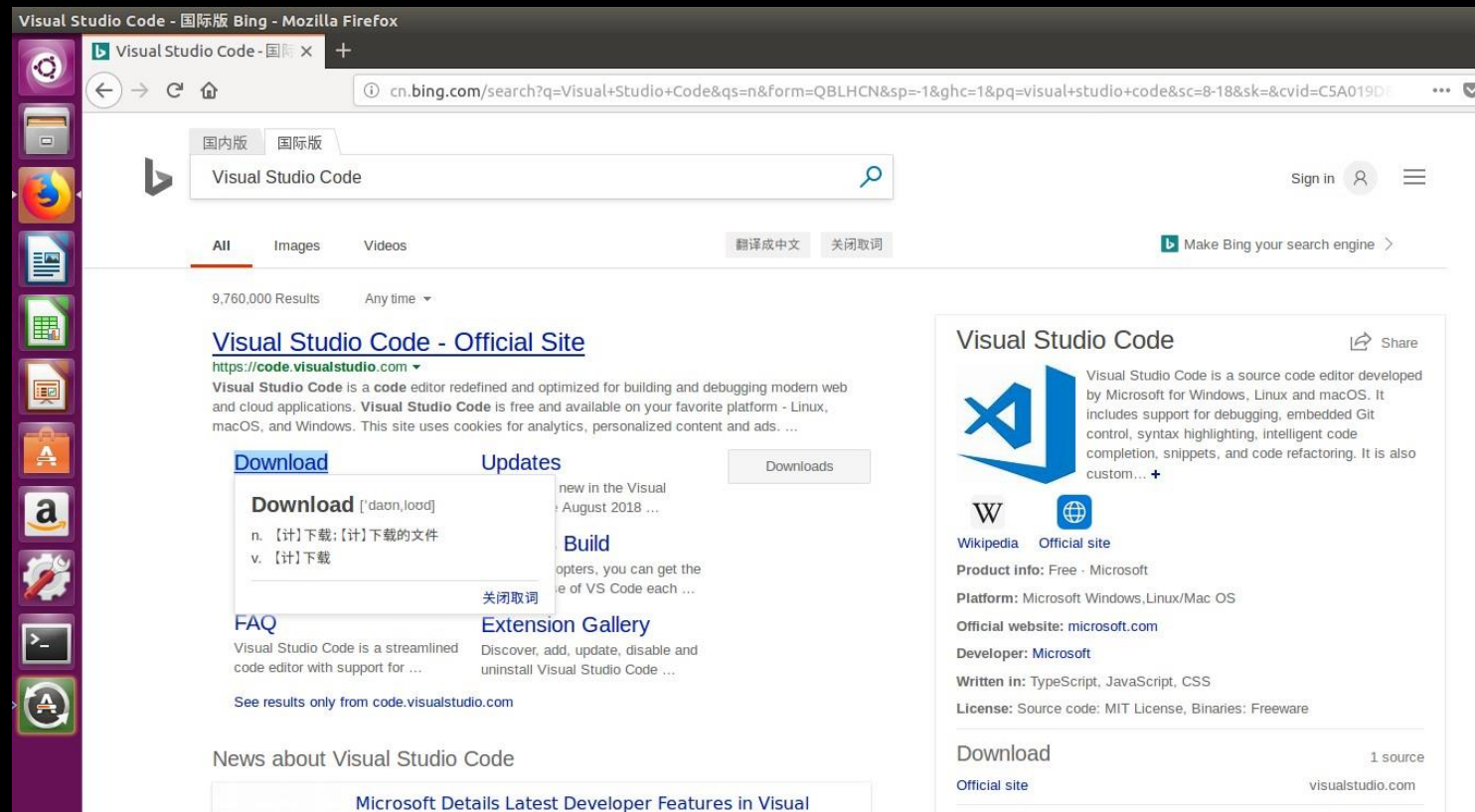
# Installation of Visual Studio Code

- Find online



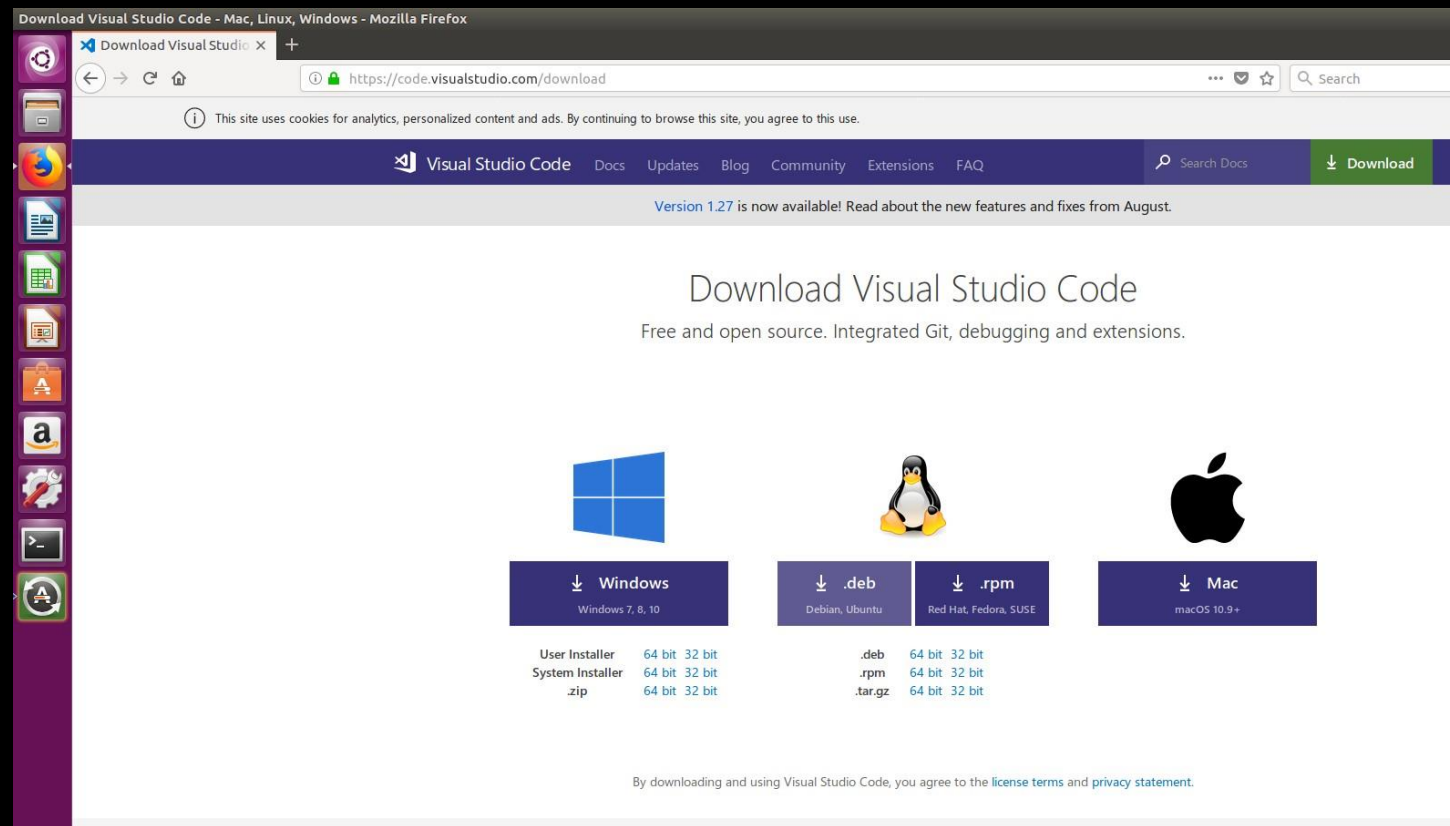
# Installation of Visual Studio Code

- Find online



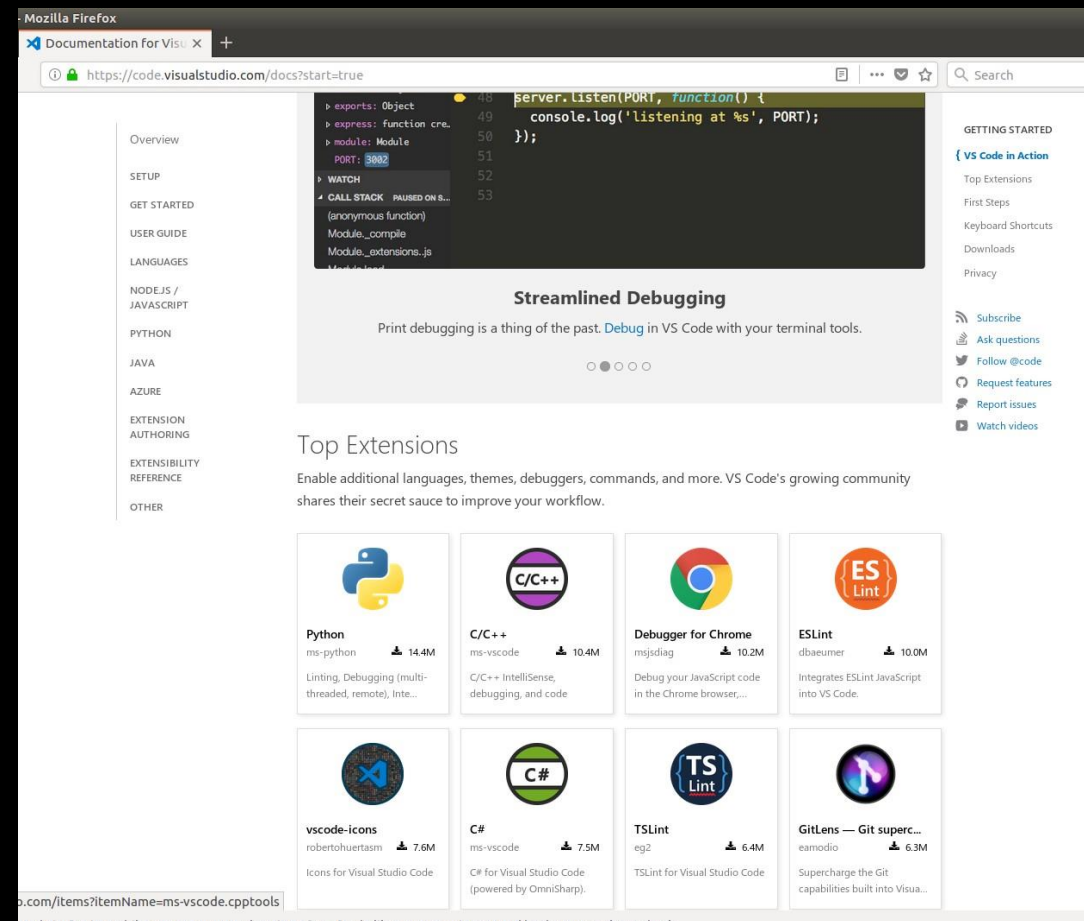
# Installation of Visual Studio Code

- Download for correct operating system!




# Installation of Visual Studio Code

- Add C/C++ extensions



# Installation of Visual Studio Code

- Add C/C++ extensions



**C/C++** Preview

**Microsoft** | 📦 12,133,665 installs | ★★★★★ (225)

C/C++ IntelliSense, debugging, and code browsing.

**Installation**

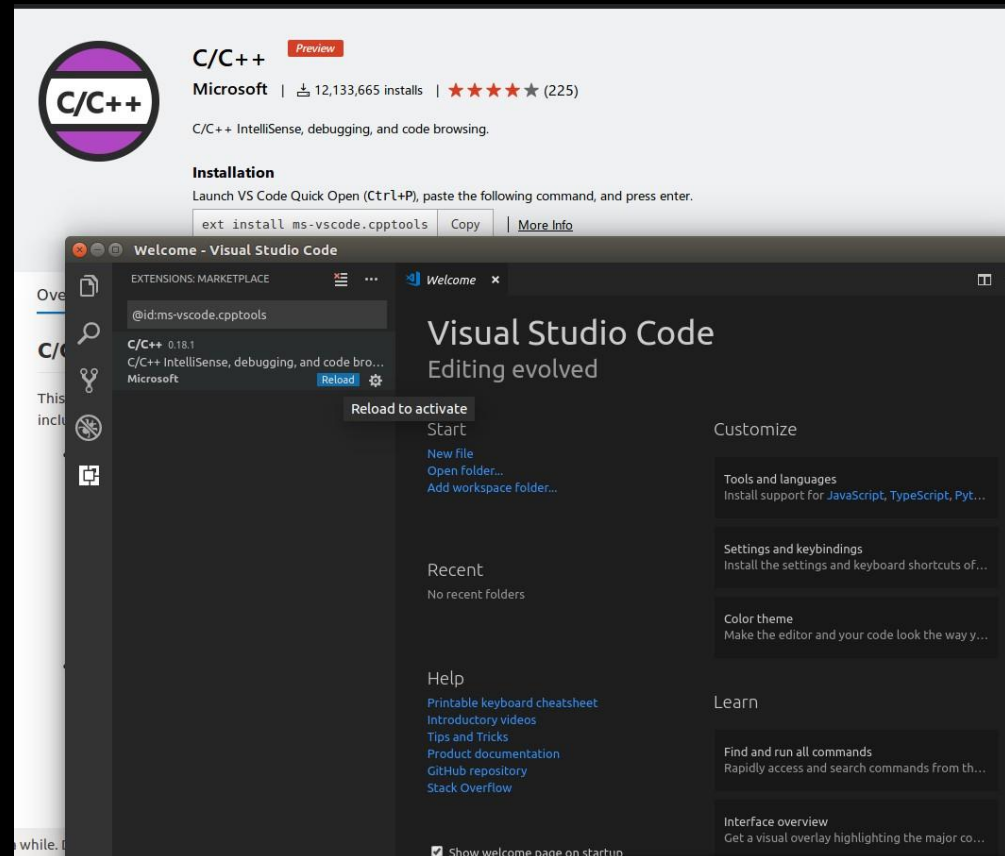
Launch VS Code Quick Open (Ctrl+P), paste the following command, and press enter.

 | [More Info](#)



# Installation of Visual Studio Code

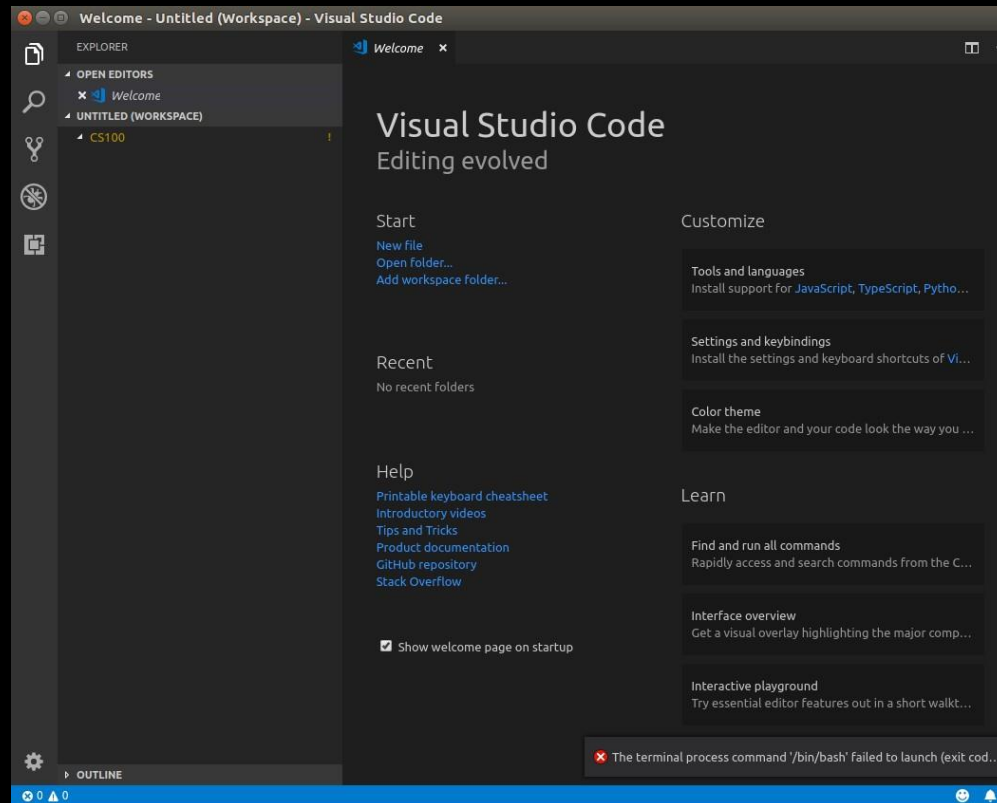
- Install/Reload to activate



**Hello World example**

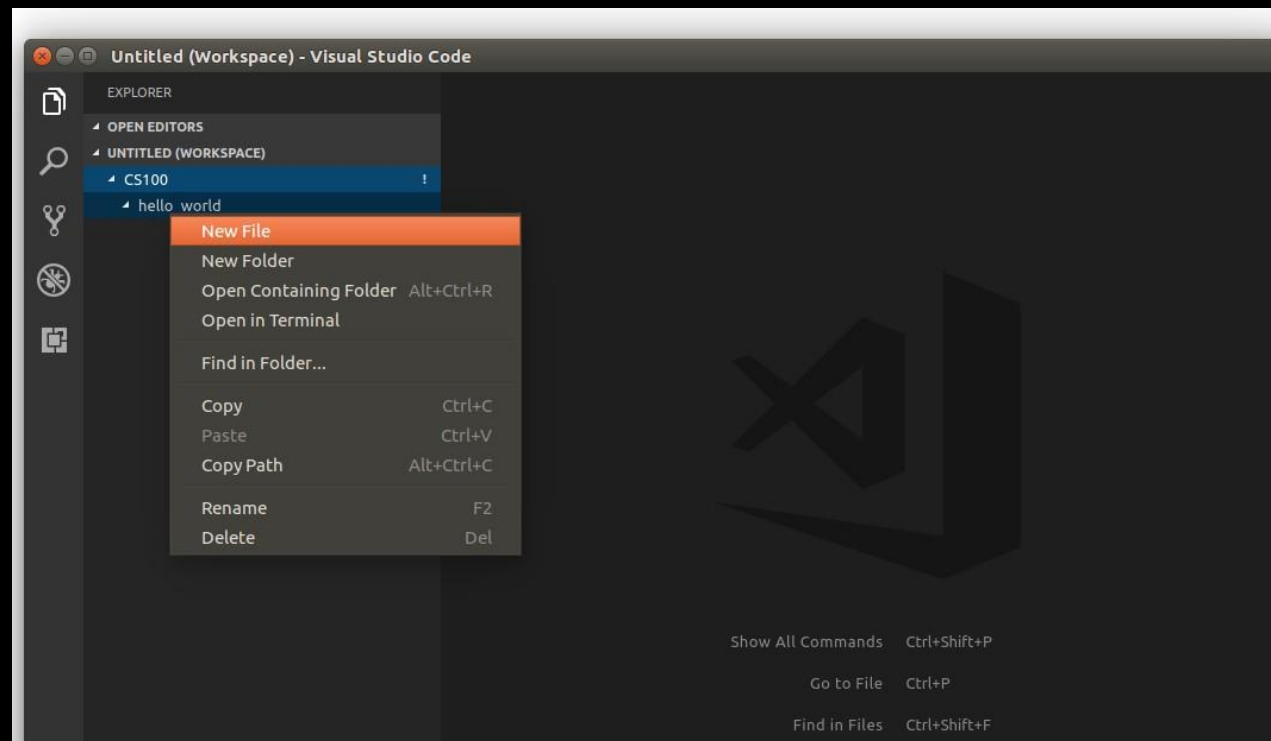
# Hello World example

- Add workspace folder  
(create/select  
folder "CS100")



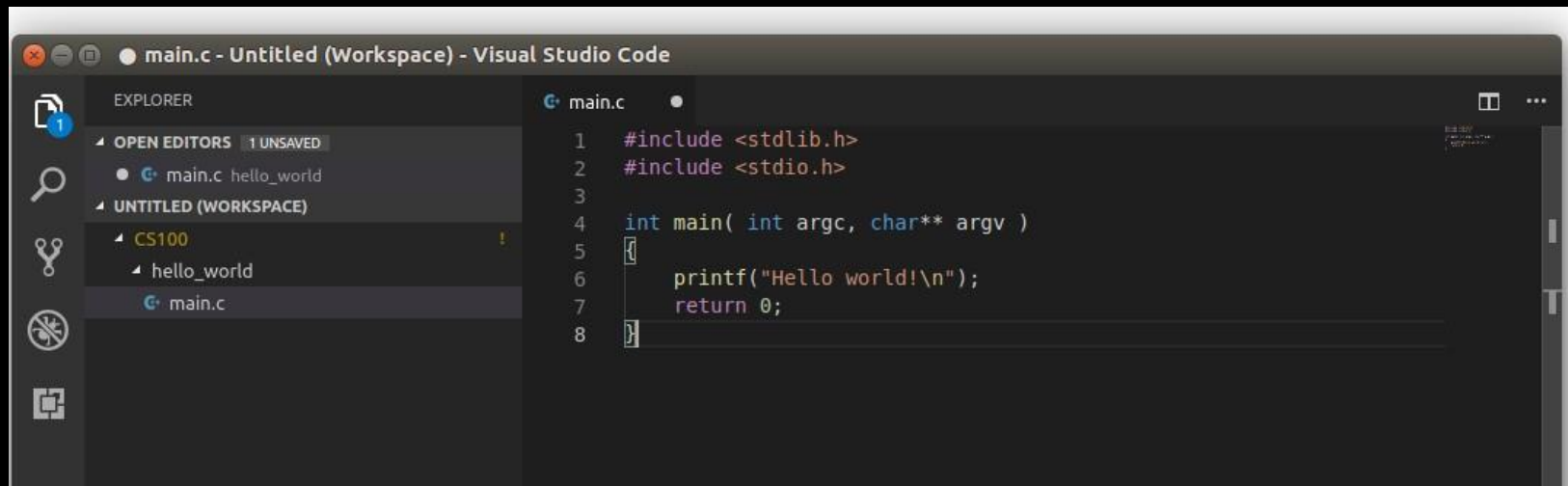
# Hello World example

- Add new folder “hello world”
- Add new file main.c



# Hello World example

- Implement hello world code



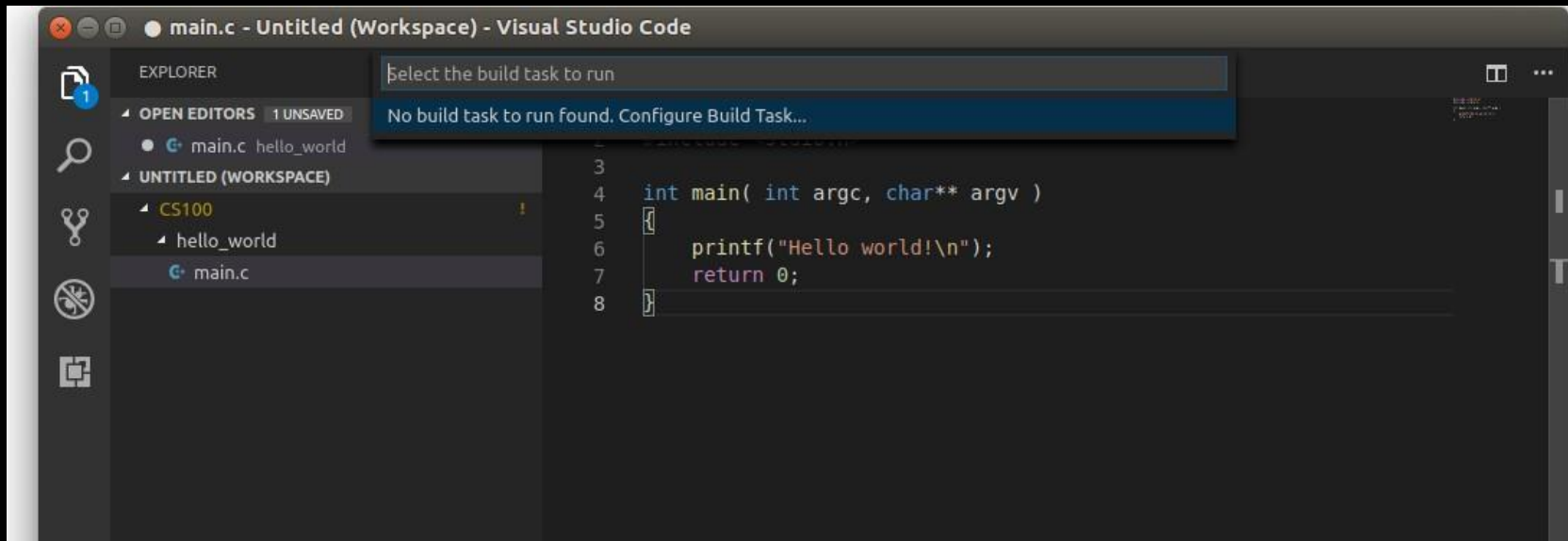
# Hello World example

- Open terminal
- Go to directory CS\_100/hello\_world
  - *cd <root\_directory>/CS\_100/hello\_world*
- Execute compilation
  - *gcc main.c -o main*
- Run program
  - *./main*

```
Last login: Wed Sep 19 14:45:20 on ttys000
Laurents-MacBook-Pro:~ laurent$ cd Documents/CS_100/hello_world/
Laurents-MacBook-Pro:hello_world laurent$ gcc main.c -o main
Laurents-MacBook-Pro:hello_world laurent$ ./main
Hello world!
Laurents-MacBook-Pro:hello_world laurent$
```

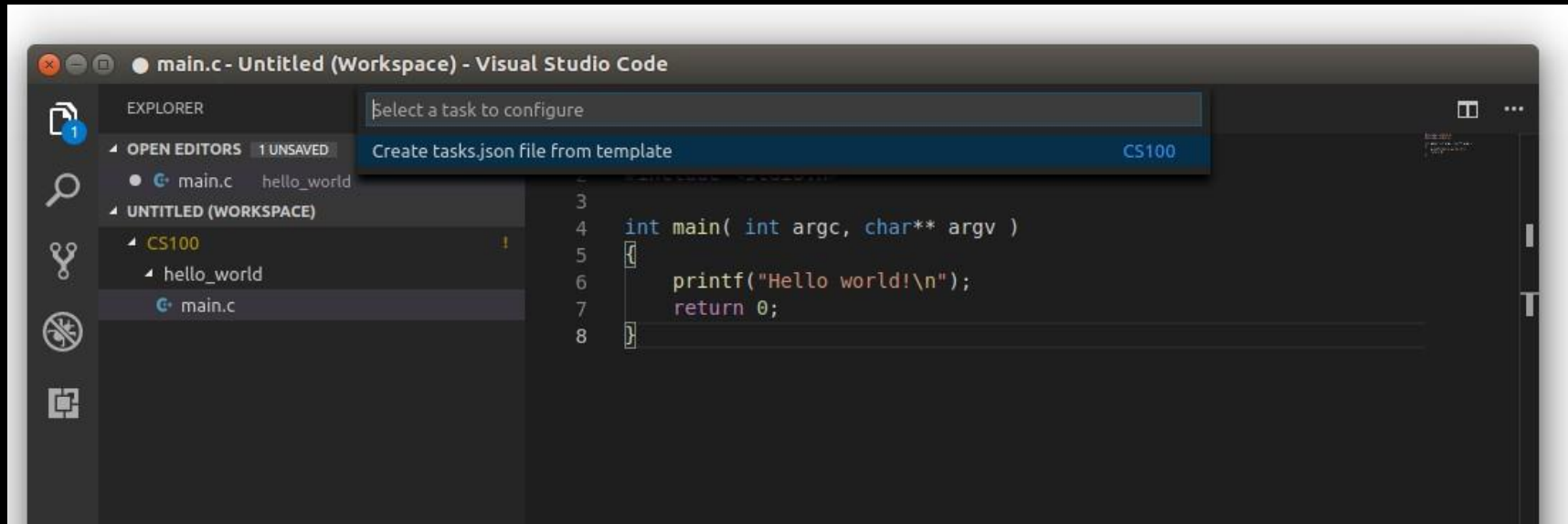
# Hello World example

- Try to trigger build from within VS Code
- Try to build using command CTRL-SHIFT-B
  - It does not know how, and asks whether you want to configure build task → click on it



# Hello World example

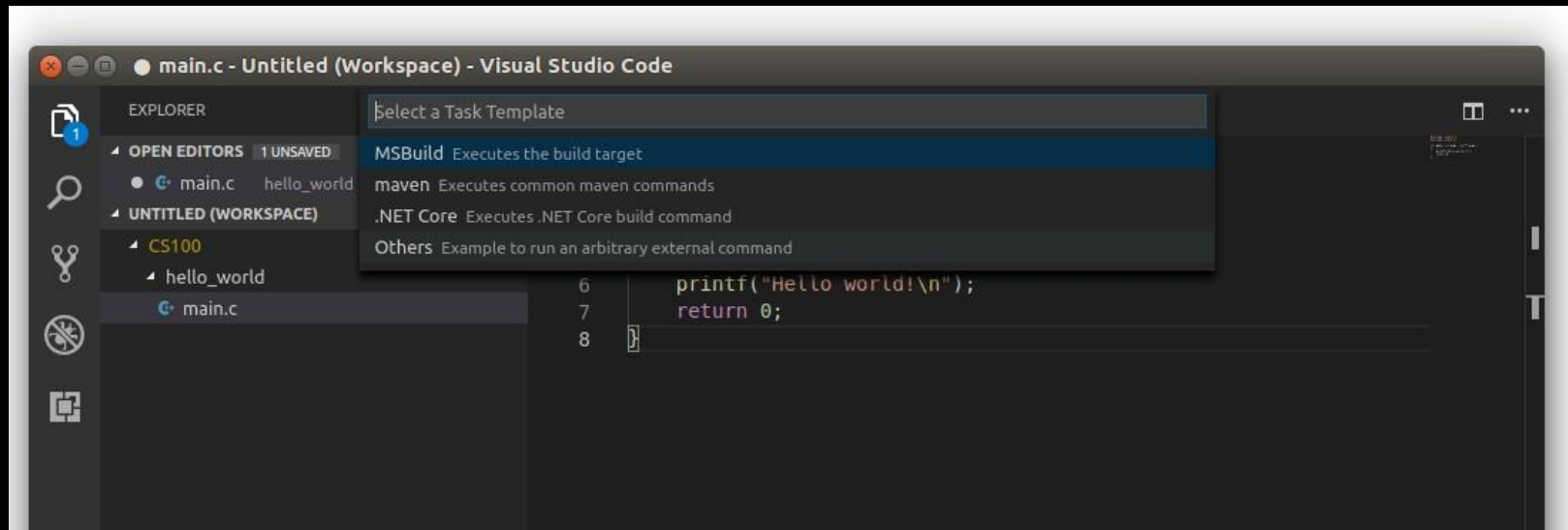
- It does not have a task file, but suggests to create one → click on it!





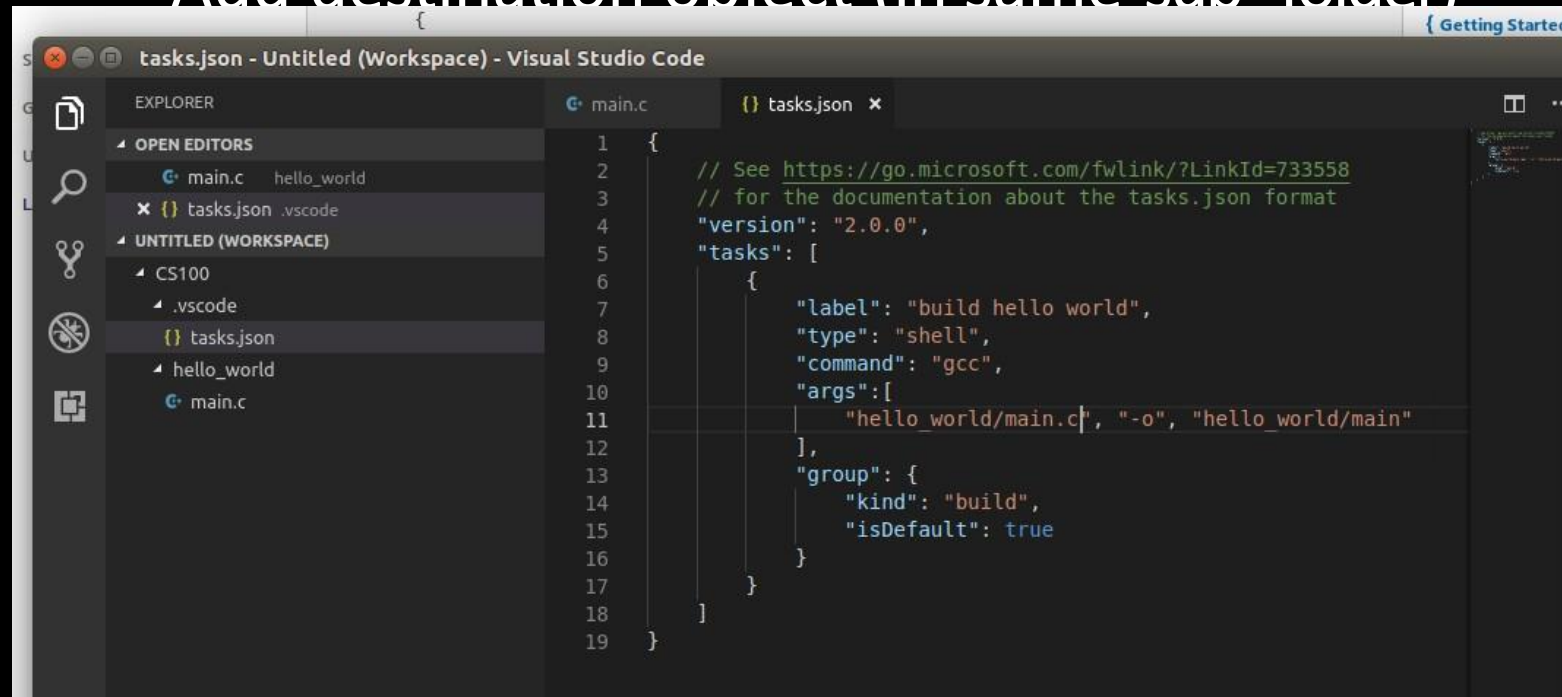
# Hello World example

- Choose “Others” as template



# Hello World example

- Edit tasks.json to add build command
  - Add full path to file
  - Add destination object (in same sub--folder)

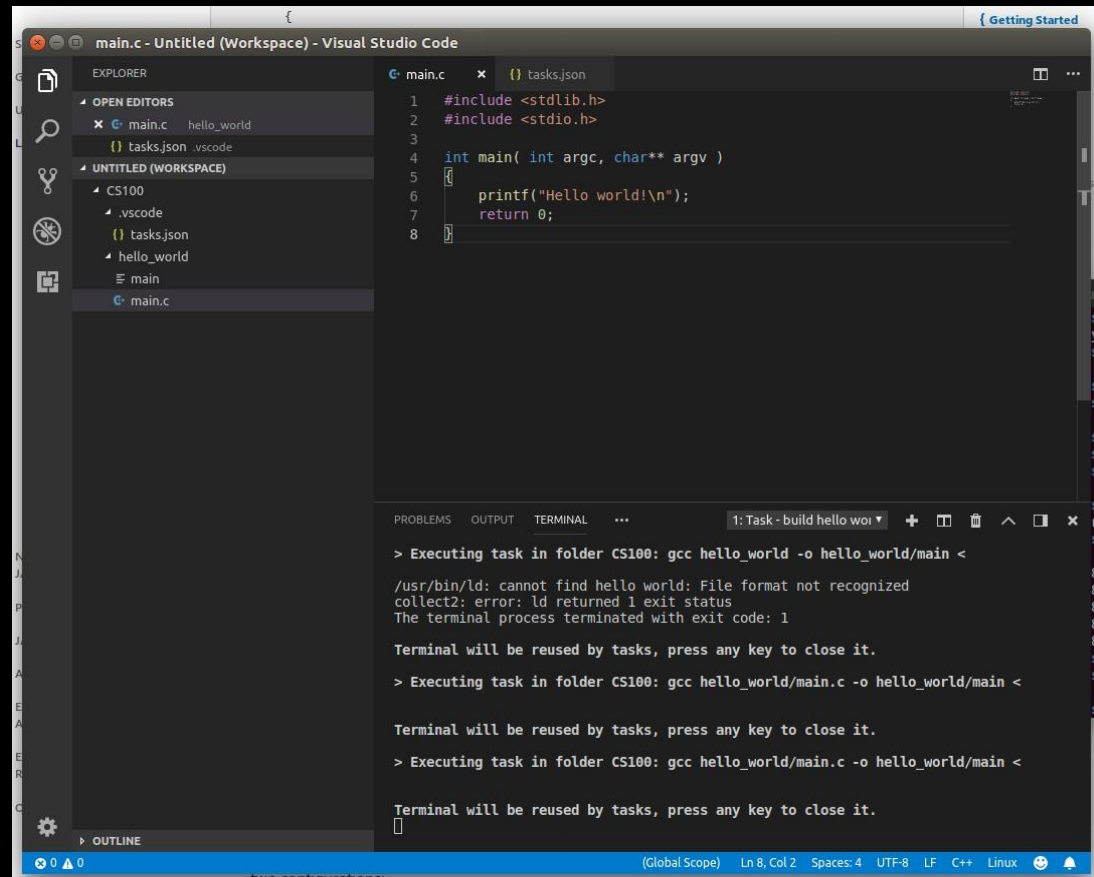


The screenshot shows the Visual Studio Code interface with the 'tasks.json' file open in the editor. The Explorer sidebar on the left shows the project structure: 'main.c' and 'hello\_world' are in the 'OPEN EDITORS' section, and 'tasks.json' is in the 'UNTITLED (WORKSPACE)' section. The main editor area displays the following JSON code:

```
1 {  
2   // See https://go.microsoft.com/fwlink/?LinkId=733558  
3   // for the documentation about the tasks.json format  
4   "version": "2.0.0",  
5   "tasks": [  
6     {  
7       "label": "build hello world",  
8       "type": "shell",  
9       "command": "gcc",  
10      "args": [  
11        "hello_world/main.c", "-o", "hello_world/main"  
12      ],  
13      "group": {  
14        "kind": "build",  
15        "isDefault": true  
16      }  
17    }  
18  ]  
19 }
```

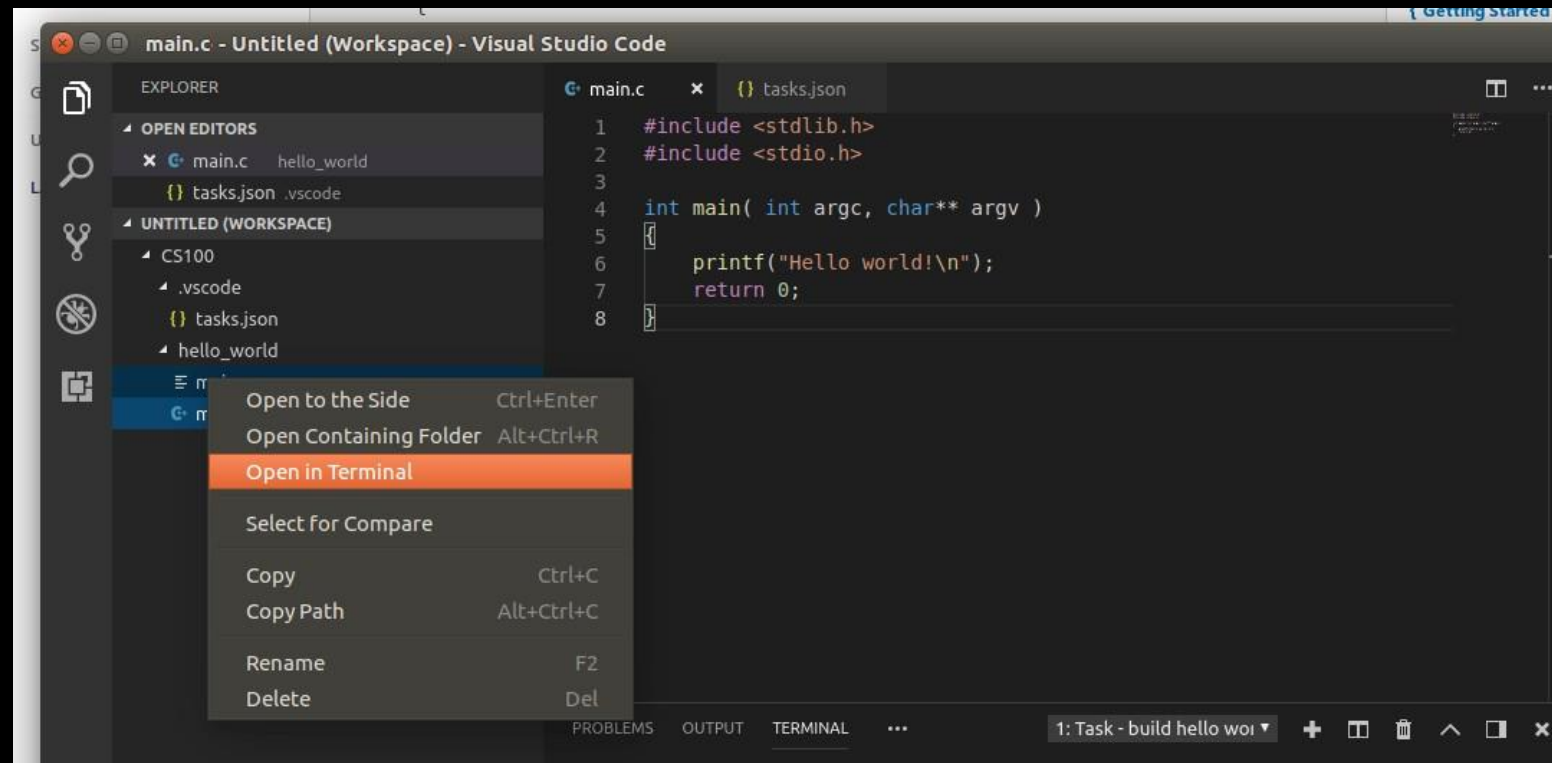
# Hello World example

- Press CTRL-SHIFT-B to run build process
- VS Code may have to be restarted for changes to be effective!



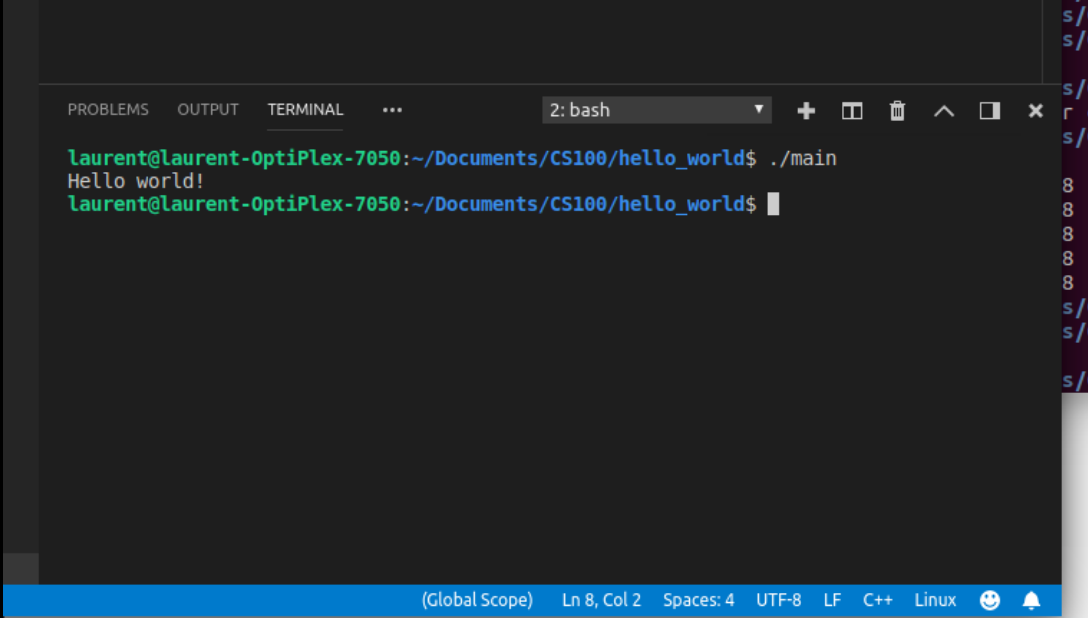
# Hello World example

- Select binary main and choose “open in Terminal” from drop-down menu



# Hello World example

- Run the example  
*./main*
- In windows:  
*main*



The screenshot shows a terminal window with a dark background. The title bar at the top indicates the terminal is running '2: bash'. The terminal content shows the user 'laurent' at machine 'laurent-OptiPlex-7050' in the directory '~/Documents/CS100/hello\_world'. The user enters the command './main', and the program outputs 'Hello world!'. The prompt returns to the user. The bottom status bar of the terminal shows '(Global Scope) Ln 8, Col 2 Spaces: 4 UTF-8 LF C++ Linux' along with some icons.

```
laurent@laurent-OptiPlex-7050:~/Documents/CS100/hello_world$ ./main
Hello world!
laurent@laurent-OptiPlex-7050:~/Documents/CS100/hello_world$
```

**What are you waiting for?  
Try it now!**

# NO PLAGIARISM!!!

- The most likely cause for failing this course.
- You WILL be caught!
- We WILL punish!
- They WILL know!
  - Parents
  - University
  - School
  - Fellows

# Further readings

A bite into the programmer career


You will have the slides, google these buzzwords after class

You may have no idea of something in the next slides is now



# Other popular C/C++ IDEs/text editors

Resource  
Hungry

- 
- Visual Studio (Community ver.)
    - Powerful but very, very bulky
    - Mainly MS technology stack
  - CLion
    - Free for students
    - Requires CMake knowledge
    - JetBrains plugin ecosystem
  - Eclipse CDT
    - Open source
    - Eclipse plugin ecosystem
  - Code::Blocks
    - Drop in replacement of Dev C++
  - Sublime Text
    - Optional pay
    - Lightweight
  - Notepad++
    - Freeware (GNU)
    - Windows only
    - Drop in replacement of Notepad
  - Vim/Emacs
    - Blazing fast, runs fine on headless
    - Terminal based, with visual modes
    - More than enough plugins
    - Steep/awkward learning curve
    - Decades long holy war included

Resource  
Friendly

# Notorious IDE/Editor functions

- Good text editor
  - Search & Replace
  - Syntax highlight
  - Code formatter
  - Block select/edit
  - Different line separator?
- Git/SVN integration
  - Basic functions in GUI
  - Merge assistance
- Code completion
- Build tool integration
  - Compile, run, test in one button
- Navigation assistance
  - Go to definition
  - Find usage
- Code inspection
  - Style problems
  - Potential bugs
  - Compile errors
- Language/framework support
- GUI Debugger
- Integrated terminal
- Huge plugin library