INTRODUCTION TO C

CPSC 441 TUTORIAL – JANUARY 16, 2012
TA: RUITING ZHOU

- TA: Ruiting Zhou
- Email: rzho@ucalgary.ca
- CT hour: Wednesday 3:00pm-4:00pm
  Friday: 11:00am-12:00am
- Math Science Building
- 1st floor, Computer science Lab

SIMPLE C EXAMPLE

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

int main(int argc, char
*argv[])) {
  printf("Hello world!\n");
  return 0;
}
```

COMPILING C

- gcc invokes C compiler
- gcc translates C program into executable for some target
- default file name a.out

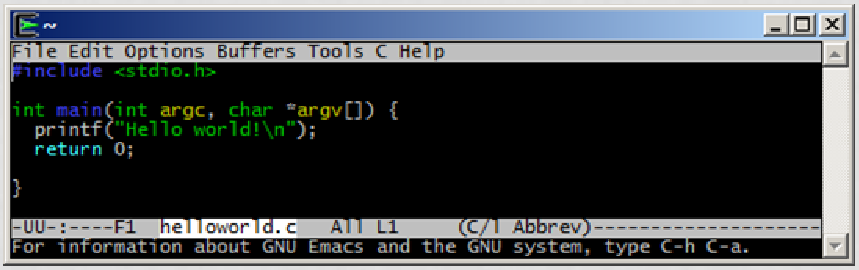
- Example: compile and run hello.c

```
$ gcc hello.c
$ a.out
Hello, World!

$ gcc hello.c –o hello
$ ./hello
Hello, World!
```

HANDS ON

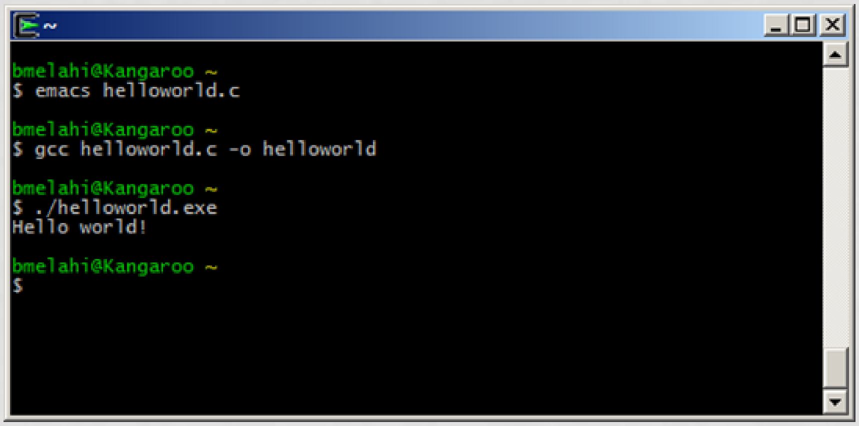
- Demo:
  - 1. Write code
  - 2. Compile
  - 3. Run



```
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("Hello world!\n");
  return 0;

}
-UU-:----F1  helloworld.c    All L1      (C/l Abbrev)--------------------
For information about GNU Emacs and the GNU system, type C-h C-a.
```



```
bmelahi@Kangaroo ~
$ emacs helloworld.c

bmelahi@Kangaroo ~
$ gcc helloworld.c -o helloworld

bmelahi@Kangaroo ~
$ ./helloworld.exe
Hello world!

bmelahi@Kangaroo ~
$
```

SOME OPTIONS

- Some useful command line options:
  - [-o file]: specifies the output file for object or executable
  - [-Wall]: show all warnings (highly recommended)
  - [-l libnam]: Links the library libname, e.g., -lxnet
    - If you get errors saying the library cannot be found, make sure the path is correctly set, and you *do* have the libraries you need.

LIBRARIES

- #include <stdio.h>
- C provides a set of standard libraries for

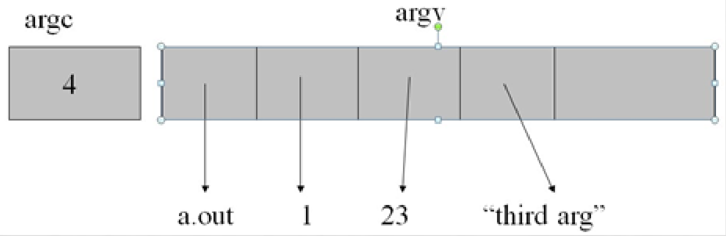
numerical math functions	`<math.h>`	`-lm`
character strings	`<string.h>`	
character types	`<ctype.h>`	
I/O	`<stdio.h>`	

- `#include <math.h>`
  - careful: `sqrt(5)` without header file may give wrong result!
- `gcc -o a main.c -lm`

MAIN ARGUMENTS

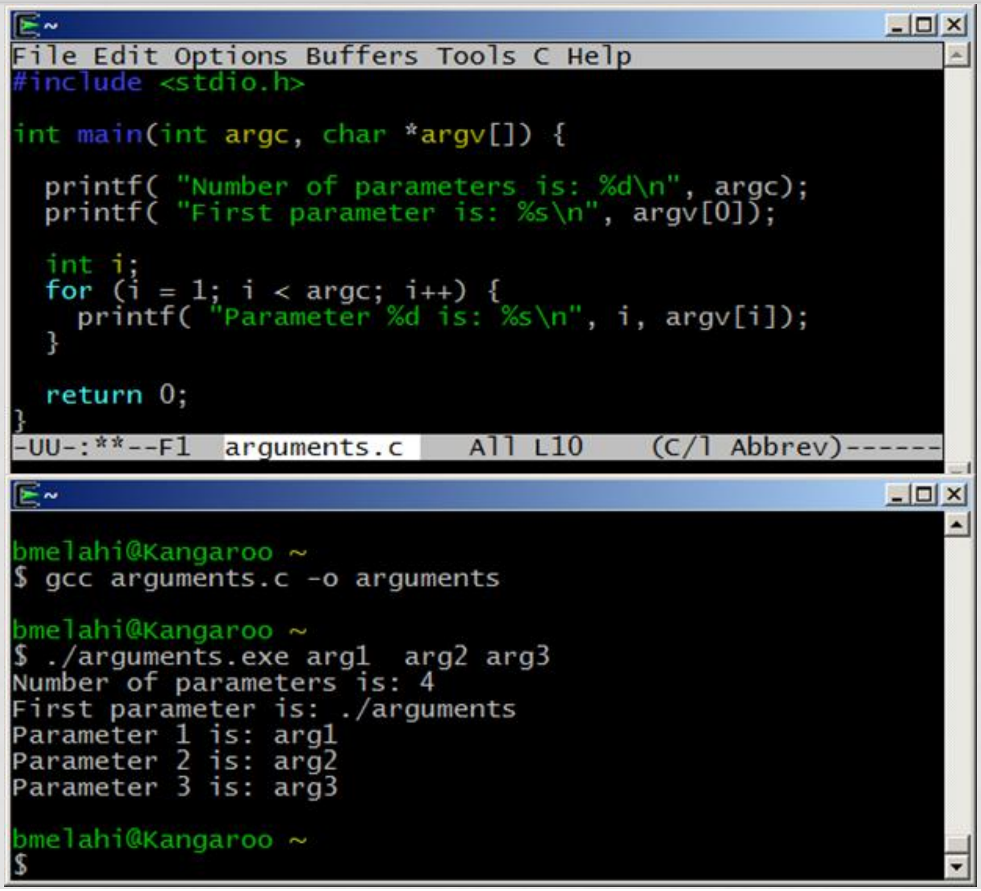
```
int main(int argc, char *argv[])
```

- argc: number of arguments passed to the program
- argv: pointers that list all of the arguments
  - Name of executable + space-separated arguments
  - Name of executable is stored in argv[0]
- $ a.out 1 23 'third arg'



- The return value is `int`
  - convention: 0 means success, > 0 some error

PASSING ARGUMENTS EXAMPLE


~
File Edit Options Buffers Tools C Help
-UU-:**--F1 arguments.c   All L10   (C/l Abbrev)------
~

```c
#include <stdio.h>

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

  printf( "Number of parameters is: %d\n", argc);
  printf( "First parameter is: %s\n", argv[0]);

  int i;
  for (i = 1; i < argc; i++) {
    printf( "Parameter %d is: %s\n", i, argv[i]);
  }

  return 0;
}
```

```
bmelahi@Kangaroo ~
$ gcc arguments.c -o arguments

bmelahi@Kangaroo ~
$ ./arguments.exe arg1  arg2 arg3
Number of parameters is: 4
First parameter is: ./arguments
Parameter 1 is: arg1
Parameter 2 is: arg2
Parameter 3 is: arg3

bmelahi@Kangaroo ~
$
```

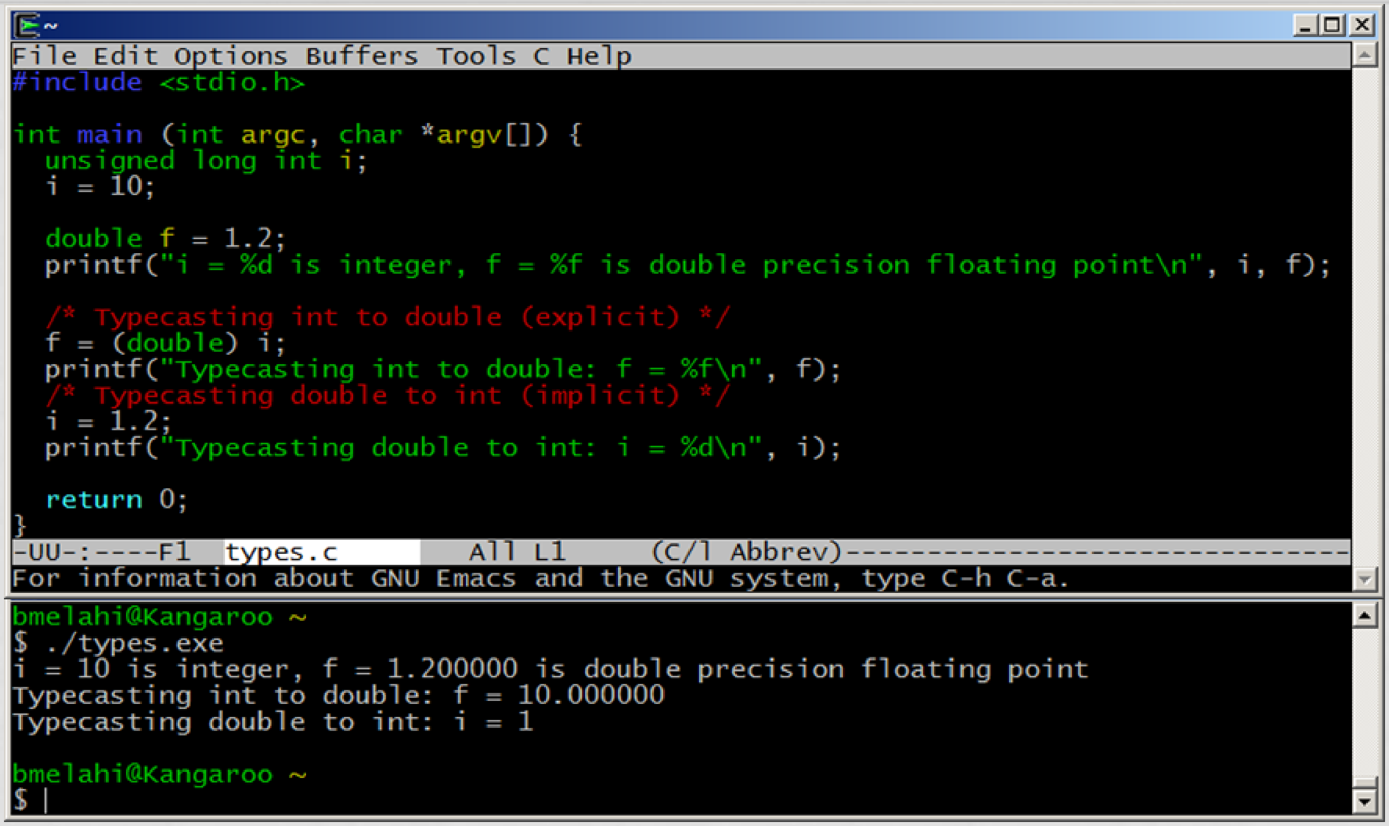
PRIMITIVE DATA TYPES

Name	Description	Size* (32bit)	Range* (32bit system)
char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1byte	true or false
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)

**Size* and *Range* depend on the system the program is compiled for.

From: http://www.cplusplus.com/doc/tutorial/variables/

TYPECASTING EXAMPLE



```
#include <stdio.h>

int main (int argc, char *argv[]) {
  unsigned long int i;
  i = 10;

  double f = 1.2;
  printf("i = %d is integer, f = %f is double precision floating point\n", i, f);

  /* Typecasting int to double (explicit) */
  f = (double) i;
  printf("Typecasting int to double: f = %f\n", f);
  /* Typecasting double to int (implicit) */
  i = 1.2;
  printf("Typecasting double to int: i = %d\n", i);

  return 0;
}
-UU-:----F1  types.c        All L1      (C/l Abbrev)------------------------------
For information about GNU Emacs and the GNU system, type C-h C-a.
```

```
bmelahi@Kangaroo ~
$ ./types.exe
i = 10 is integer, f = 1.200000 is double precision floating point
Typecasting int to double: f = 10.000000
Typecasting double to int: i = 1

bmelahi@Kangaroo ~
$ |
```

Caution: be careful with typecasting, especially implicit conversions.

IF AND LOOPS

- **IF statement:**

```
if ( TRUE ) { /* Execute these statements if TRUE */ }
 else { /* Execute these statements if FALSE */ }

if ( age < 100 ) { /* If the age is less than 100 */
 printf ("You are pretty young!\n" ); /* Just to show you it
   works... */ }
else if ( age == 100 ) { /* I use else just to show an example */
   printf( "You are old\n" ); }
else { printf( "You are really old\n" ); /* Executed if no other
   statement is */ }
```

IF AND LOOPS

- C has several control structures for **repetition**

Statement	**repeats an action…**
while(c) {}	zero or more times, while condition is ≠ 0
do {…} while(c)	one or more times, while condition is ≠ 0
for (start; cond; upd)	zero or more times, with initialization and update

```
for ( x = 0; x < 10; x++ ) {}
```

ARRAYS

- Array declaration: **`int a[100];`**

- C/C++ arrays have no length parameter!
  - Note: when passing an array to a function, typically you have to pass the array size as a separate argument as well.

- You have to take care of array bounds yourself

```
int input[10];
input[10] = 20; // out of bound!
input[-1] = 5;  // out of bound!
```

  - This code could compile and run, but most likely, you'll see unexpected behavior or crash your program.

- Array's name is a pointer to its first element

STRUCTURES

- C `struct` is a way to *logically* group related types
  - Is very similar to (but not same as) C++/java **classes**
  - store many different values in variables of potentially different types under the same name.
- A `struct` component can be of any type (including other `struct` types
- Example:

```
struct database {
 int id_number;
 int age;
 float salary; };
```

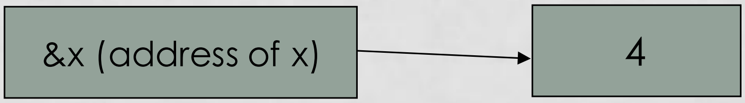
```
int main() {
 struct database employee;
employee.age = 22;
employee.id_number = 1;
employee.salary = 12000.21;
 }
```

POINTERS

- A pointer, they "point" to locations in memory
  - Another variable
  - Some dynamically allocated memory
  - Some function
  - **NULL**

```
int *p
p = &x;
```

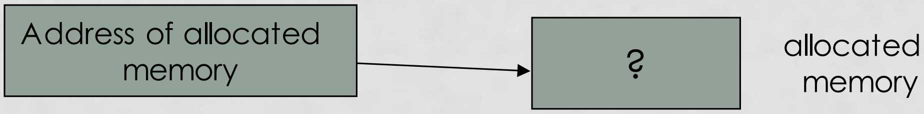
```
int x = 4;
```



&x (address of x)

4

```
int *p = malloc (sizeof int);
```



Address of allocated memory

?

allocated memory

POINTERS IN C

- Declaration: using "*****" symbol before variable name.

```
int * ptr = NULL; //creates pointer to integer
```

- Allocation: allocate new memory to a pointer using the keyword **`malloc`** in C (`new` in C++)

```
int *p = malloc(sizeof(int));
```

- Deallocation: clear the allocated memory when you are done using it. Otherwise, Memory Leak!!!

```
free(p);
```

- Dereferencing: accessing data from the pointer `x = *p;`

STRING

- In C, string is an array of **`char`** terminated with “\0” (a null terminator: ‘\0’)
  - **`“hello” = hello\0`**

- Declaring and initialize a string

```
char str1[10];               // a string of 10 characters
char str2[10]={“hello”}; //initialized string

char *strp1;                // a char pointer

char *strp2 = malloc(sizeof(char)*10);
    // a char pointer initialized to point to a chunk of memory.
```

STANDARD C LIBRARY

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

- Formatted I/O

```
int scanf(const char *format, ...)
```

- read from standard input and store according to format.

```
int printf(const char *format, ...)
```

- write to standard output according to format

Example:

```
int this_is_a_number;
printf( "Please enter a number: " );
scanf( "%d", &this_is_a_number );
printf( "You entered %d", this_is_a_number );
```

STANDARD C LIBRARY

- File I/O: **`FILE *`**

```
FILE *fopen(const char *path, const char *mode)
```

- open a file and return a FILE pointer. Can use the FILE pointer perform input and output functions on the file

```
FILE *fp;
 fp=fopen("c:\\test.txt", "r");
```

```
int fclose(FILE *stream)
```

- close the file; return 0 if successful, EOF if not

```
fclose(fp);
```

LETS WRITE SOME CODE!

- Sample C program:
  - Input: list of grades of student homework.
  - Output: The computed final marks.

REFERENCES

- C for Java programmers:

http://faculty.ksu.edu.sa/jebari_chaker/papers/C_for_Java_Programmers.pdf
http://www.cs.columbia.edu/~hgs/teaching/ap/slides/CforJavaProgrammers.ppt

- C tutorial:

http://www.cprogramming.com/tutorial/c-tutorial.html

- Socket programming with C: (for next session)
  - Beej's Guide to Network Programming Using Internet Sockets
    http://beej.us/guide/bgnet/output/html/multipage/index.html