C/C++ Programming Review

FATEMEH HOSSEINI
UNIVERSITY OF CALGARY – CPSC 441

Text Editor

- There are several editors for C/C++ programming in Linux like: Netbeans, Eclipse, Microsoft's Visual Studio, VI/VIM and so on
- Choose the one that corresponds best to your habits and your tasks
- Easiest way is just to open gedit, write your program and save it
- Then compile and run the executable file

Create a C/C++ file

C

```
#include<stdio.h>
int main(int argc, char *argv[])
{
   printf("Hello World\n");
   printf("Welcome to CPSC 441\n");
   return 0;
}
```

C++

```
#include<iostream>
using namespace std;
int main(int argc, char *argv[])
{
cout<<"Hello World\n";
cout<<"Welcome to CPSC 441\n";
return 0;
}
```

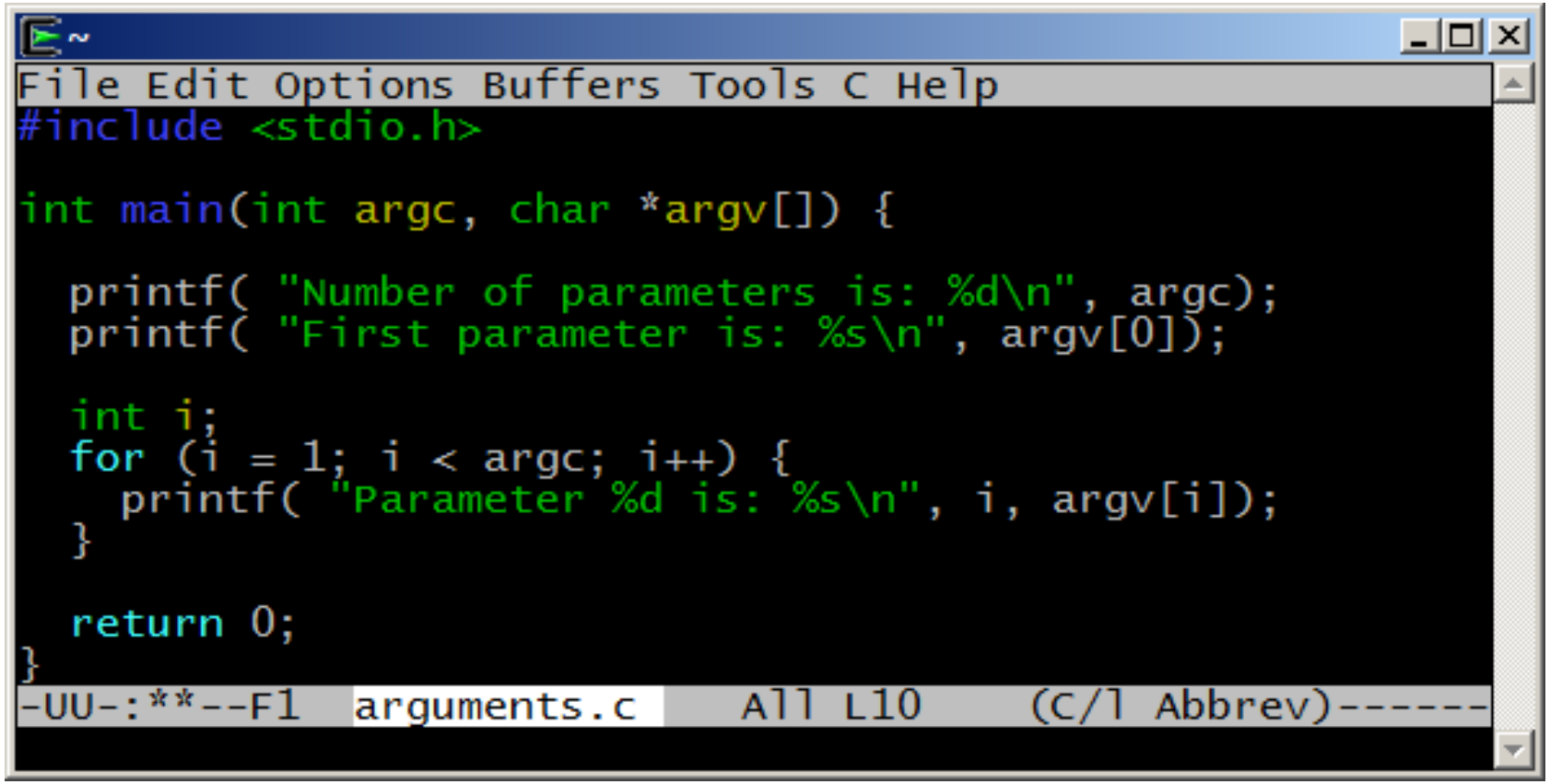
Compile and Run the program

- One way to compile and run the program: **gcc program.c or g++ program.cpp**
  ./a.out

  Another way: **gcc program.c -o program**
  ./pogram

- Output of the program:
  Hello World
  Welcome to CPSC 441

Passing Arguments Example



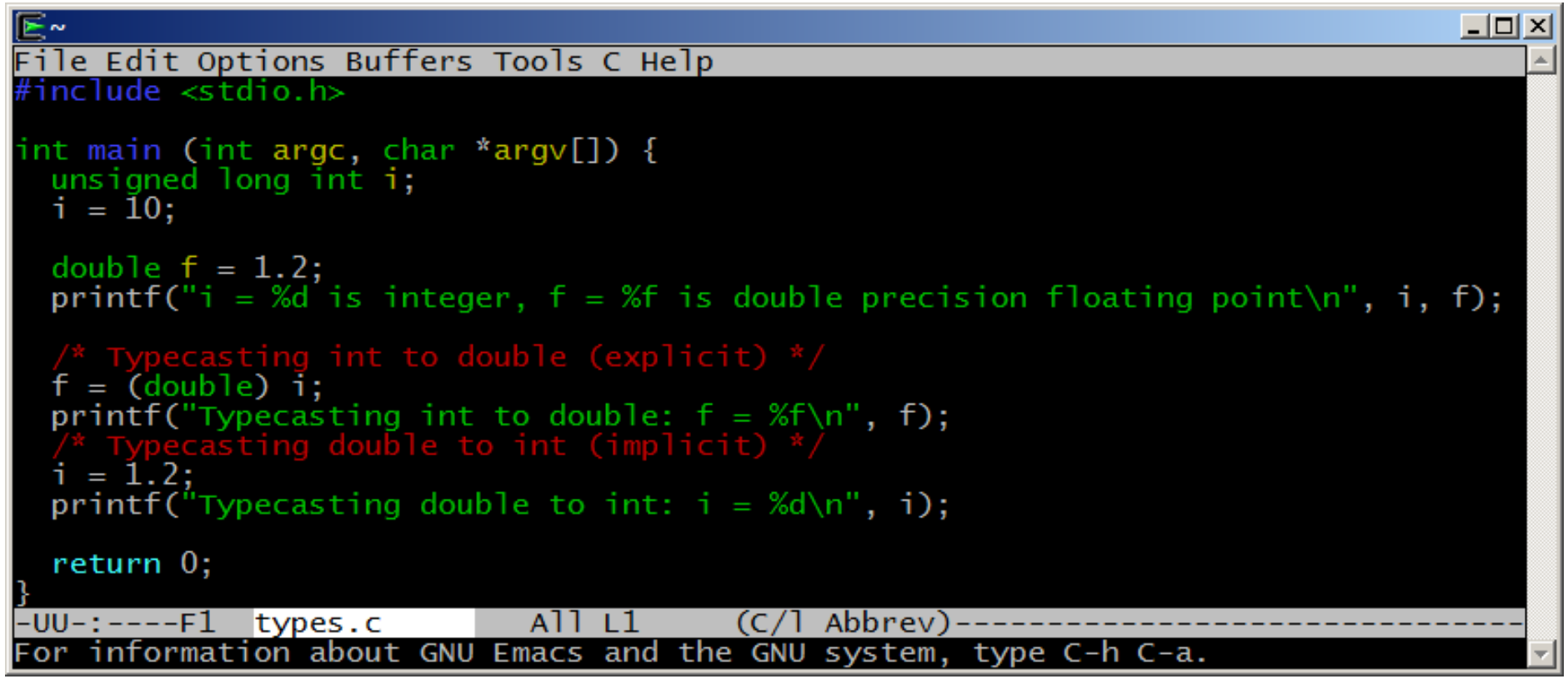
Passing Arguments Example

```
fatemeh@slave:~/c_examples$ ./program argv1 argv2 argv3
number of input arguments:4
executable file:./program
Argument1: argv1
Argument2: argv2
Argument3: argv3
```

Data Types

Name	Description	Size (Bytes)	Range
char	Character or small integer.	Typically 1	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	>=2	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer. **Most efficient**.	>=2; Typically 4	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	>=4	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long long int	long long integer.	>=8	signed: -9.2e18 to 9.2e18 unsigned: -1.8e19 to 1.8e19
float	Floating point number.	4	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8	+/- 1.8e +/- 308 (~15 digits)
long double	Long double precision floating point number.	>=8; Typically 16	+/- 1.2e +/- 4932 (~34 digits)

Type Casting Example



Type Casting Example

```
fatemeh@slave:~/c_examples$ gcc type.c -o type
fatemeh@slave:~/c_examples$ ./type
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
```

Arrays

// Array declaration by specifying size
int arr[10];

// Array declaration by initializing elements
int arr[] = {10, 20, 30, 40};

// Compiler creates an array of size 4.
// above is same as "int arr[4] = {10, 20, 30, 40}"

No Index Out of bound Checking in C, For example the following program compiles fine but may produce **unexpected output** when run:

```
int main(){
  int arr[2];
  printf("%d ", arr[3]);
  printf("%d ", arr[-2]);
  return 0;}
```

Struct

```
typedef struct {
   type    member1;
   type    member2;
} struct_alias;
```

```
int main(void) {

/* Define a type point to be a struct with integer members x, y */
typedef struct {
   int    x;
   int    y;
} point;
```

Struct

```
/* Define a variable p of type point, and initialize all its members inline! */
   point p = { 1, 3 };

/* Define a variable q of type point. Members are uninitialized. */
   point q;

/* Assign the value of p to q, copies the member values from p into q. */
   q = p;

/* Change the member x of q to have the value of 3 */
   q.x = 3;
```

Struct

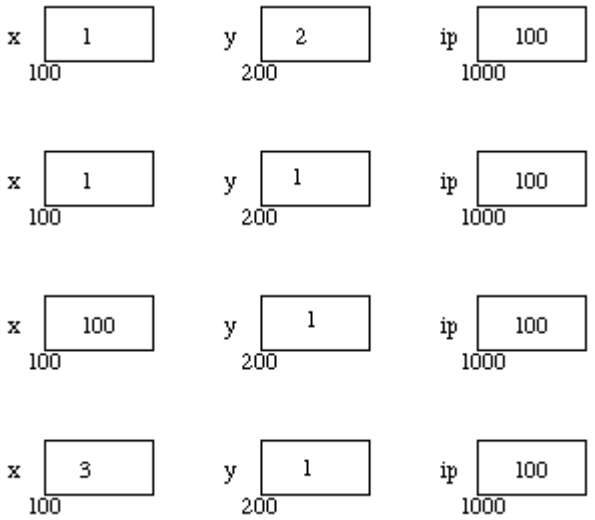
```
/* Demonstrate we have a copy and that they are now different. */
    if (p.x != q.x) printf("The members are not equal! %d != %d", p.x,
q.x);

/* Define a variable r of type point. Members are uninitialized. */
    point r;


/* Assign values using compound literal (ISO C99/supported by GCC > 2.95) */
    r = (point) { 1, 2 };

    return 0;
}
```

Pointers

- A pointer is a variable which contains the **address** in memory of another variable. We can have a pointer to any variable type.

Example:

```
int x = 1, y = 2;
int *ip;
ip=&x;
y=*ip;
x=ip;
*ip=3;
```



Strings

- String: array of characters terminated by NULL character
- String in/output:
  printf("%s",S), scanf("%s",S)
- string.h: collection of functions for string manipulation

remember: strings are arrays!

```
char *strp2 =
malloc(sizeof(char)*4);
```

```
#include<stdio.h>
#include<string.h>
 #define MAX_STRING_LEN 4
 int main() {
 /* strings are array of characters *
terminated by the NULL character *
which is different from '0' */
char S[MAX_STRING_LEN];
int l, i;
S[0] = 'a';
S[1] = 'b';
S[2] = 'c';
S[3] = 'd;
l = strlen(S);
printf("S:\t%s\n",S);
printf("length:\t%d\n",l);
 /* print characters in S */
printf("forward\n");
for (i = 0; i < l; ++i)
 printf("S[%d] = %c\n",i,S[i]);
```

Equality of Strings

```
#include<stdio.h>
#include<string.h>
#define MAX_STRING_LEN 80
int main() { /* Warning: Strings aren't regular variables * You've to be careful with
string comparison * and assignments. */
char* S1 = "AAAA";
char* S2 = "AAAA";
int cmp1, cmp2, cmp3, cmp4;
cmp1 = (S1 == S2);
cmp2 = strcmp(S1,S2);
printf("S1:\t%s\n",S1);
 printf("S2:\t%s\n",S2);
printf("S1 == S2:\t%d\n",cmp1);
printf("strcmp(S1,S2):\t%d\n",cmp2);
```

→ Returns 1, they are equal

→ Returns 0 , they are equal

Equality of Strings

```
S2 = S1;
cmp3 = (S1 == S2);
cmp4 = strcmp(S1,S2);
printf("\nafter assignment\n");
printf("S1:\t%s\n",S1);
printf("S2:\t%s\n",S2);
printf("S1 == S2:\t%d\n",cmp3);
printf("strcmp(S1,S2):\t%d\n",cmp4);
}
```

→ Returns 1, they are equal

→ Returns 0 , they are equal

String Library in C

#include <string.h>

Functions:

char *strcpy(char *dest, char *source)
- copies chars from source array into dest array up to NULL

char *strncpy(char *dest, char *source, int num)
- copies chars; stops after num chars if no NULL before that; appends NULL

int strlen(const char *source)
- returns number of chars, excluding NULL

char *strchr(const char *source, const char ch)
- returns pointer to first occurrence of ch in source; NULL if none

char *strstr(const char *source, const char *search)
- return pointer to first occurrence of search in source

String Library in C++

#include <string>

String operations:

string substr (size_t pose, size_t len=npos)

- The returned substring is the portion of the object that starts at character position *pos* and spans *len* characters (or until the end of the string, whichever comes first).

size_t copy(char* s, size_t len, size_t pos=0)

- Copies a substring of the current value of the string object into the array pointed by *s*. This substring contains the *len* characters that start at position *pos*. The function does not append a null character at the end of the copied content.

size_t find(const string& str, size_t pos=0)

- Searches the string for the first occurrence of the sequence specified by its arguments. When *pos* is specified, the search only includes characters at or after position *pos*, ignoring any possible occurrences that include characters before *pos*.

String Library in C++

#include <string>

String operations:

int compare(const string& str)

- Compares the value of the string object (or a substring) to the sequence of characters specified by its arguments. The *compared string* is the value of the string object or -if the signature used has a *pos* and a *len* parameters- the substring that begins at its character in position *pos* and spans *len* characters. This string is compared to a *comparing string*, which is determined by the other arguments passed to the function.

size_t size() or size_t length()

- Returns length of string in terms of byte.

Reading and Parsing the Input

```
#include<string.h>
#include<stdio.h>
#define MAX_STRING_LENGTH 100
int main() { /* this example reads an input line and extract the first * four words. */
char S[MAX_STRING_LENGTH];
char A0[MAX_STRING_LENGTH];
char A1[MAX_STRING_LENGTH];
char A2[MAX_STRING_LENGTH];
char A3[MAX_STRING_LENGTH];
int n;
/* gets reads an entire line from the input */
gets(S);
/* read four strings from array of character S */
n = sscanf(S,"%s %s %s %s",A0,A1,A2,A3);
printf("strings read:\t%d\n",n);
printf("A0:\t%s\n",A0);
printf("A1:\t%s\n",A1);
printf("A2:\t%s\n",A2);
printf("A3:\t%s\n",A3); }
```

Split String to Tokens

```
#include <stdio.h>
#include <string.h>
int main () {
	char str[] ="- This is, a sample string.";
	char* pch;
	printf ("Splitting string \"%s\" into tokens:\n",str);
	pch = strtok (str," ,.-");
	while (pch != NULL)
	{
		printf ("%s\n",pch);
		pch = strtok (NULL, " ,.-");
	}
	return 0;
}
```

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

- File I/O: **FILE ***
  FILE *fopen(const char *path, const char *mode)
  open a file and return the file descriptor

  int fclose(FILE *stream)
  close the file; return 0 if successful, EOF if not

Standard C Library

#include <stdio.h>

- Other I/O operations:

  int getchar()

  read the next character from stdin; returns EOF if none

  char *fgets(char *buf, int size, FILE *in)

  read the next line from a file into buf

  int fputs(const char *str, FILE *out)

  output the string to a file, stopping at ‘\0’
  returns number of characters written or EOF

References

C tutorial: http://www.cprogramming.com/tutorial/c-tutorial.html
C++ Reference: http://www.cplusplus.com/reference/

Questions