Introduction to C

Instructor: Vinicius Prado da Fonseca, PhD (vpradodafons@online.mun.ca)

Introduction

- Originally designed for and implemented on the UNIX OS DEC PDP-11 by Dennis Ritchie
- Not tied to any hardware or system
- Easy to compile and run in other systems
- General-purpose programming language
- Not strongly typed but strong type-checking
- Variety of data types (characters, integers and float point numbers)
- Derived data types created with pointers, arrays, structures and unions

Introduction

- Fundamental control-flow (if-else, switch, while, for, do, break...)
- Functions may return basic types, structures, unions or pointers
- Supports recursive function calls
- Preprocessing performs macro substitution
- Relatively low level, C deals with same objects that most computers would use (assembly)
- High level mechanisms need to be provided by explicitly-called functions
- **Book:** *C Programming Language (2nd Edition) by Brian W. Kernighan, Dennis M. Ritchie, 1988. (Pearson)*

Environment

- Ubuntu

  ```
  $ sudo apt install build-essential
  ```
- Windows

  Brightspace instructions (not tested)
- Mac

  Xcode, brew (package manager), gcc, clang (native)

What are we covering?

- Tutorial (ch1)
- Functions and program structure (ch4)
- Pointers (ch5)
- Structures and unions (ch6)

1.1 Hello World!

- Environment test
  - Windows, Mac, Linux, VS Code
  - Text editor and shell (terminal)
  - # include <stdio.h>
  - int main()

```c
#include <stdio.h>

int main () {
    printf("hello world!\n");
}
```

To compile

```
gcc hello_world.c
```

To run

```
./a.out
```

1.2, 1.3 Fahrenheits to Celsius

- C = (5/9) * (F - 32)
- 0, 20, 40, 60...300 F ---> C?
- While loop version
- Truncated integers
  - float - float point
  - char - character, single byte
  - short - short integer
  - long - long integer
  - double - double-precision float point

```
#include <stdio.h>

int main() {

    int F, C;
    int min_value = 0;
    int max_value = 300;
    int step = 20;

    F = min_value;
    while (F <= max_value) {
        C = (5.0/9.0)*(F-32.0);
        printf("%d\t%d\n", F, C);
        F = F + step;
    }
}
```

1.2, 1.3 Fahrenheits to Celsius

- C = (5/9) * (F - 32)
- 0, 20, 40, 60...300 F ---> C?
- C and F are now floats
  - printf %f

```
#include <stdio.h>

int main() {

    float F, C;
    int min_value = 0;
    int max_value = 300;
    int step = 20;

    F = min_value;
    while (F <= max_value) {
        C = (5.0/9.0)*(F-32.0);
        printf("%3.0f\t%6.1f\n", F, C);
        F = F + step;
    }

}
```

1.2, 1.3 Fahrenheits to Celsius

- For loop version
- Symbolic constants

```
#include <stdio.h>

#define name replacement text
#define LOWER  0
#define UPPER  100
#define STEP   5

int main() {
  int celsius;

  printf("Celsius-Fahrenheit table\n");
  /* for (<initialisation>; <expression>; <expression>) */
  for (celsius = LOWER; celsius <= UPPER; celsius += STEP) {
    printf("%3.0f %6.1f\n", (float)celsius, (celsius * 9.0 /
5.0) + 32.0);
  }
}
```

1.5 Character input/output

- One character at a time
  - c = getchar()
  - putchar(c)
- ./a.out
  - type something
  - ctrl+d
- ./a.out < some_file

```
#include <stdio.h>

int main() {
    int c;

    c = getchar();
    while (c != EOF) {
        //printf("%d\n", c);
        putchar(c);
        c = getchar();
    }
}
```

1.6 Arrays

- Arrays declaration
  - int ndigit[10]
- Array indexing starts with zero
- Counts
  - each digit
  - white spaces
  - other characters
- Uses the representation of numbers to index the array
  - INCLUDE THE ASCII TABLE
  - chars = 1 byte integers

```
#include <stdio.h>

int main() {
  int c, i, nwhite, nother;
  int ndigit[10];

  nwhite = nother = 0;
  for (i = 0; i < 10; ++i) ndigit[i] = 0;

  while ((c = getchar()) != EOF)
    if (c >= '0' && c <= '9')
      ++ndigit[c - '0'];
    else if (c == ' ' || c == '\n' || c == '\t')
      ++nwhite;
    else
      ++nother;

  printf("digits = ");
  for (i = 0; i < 10; ++i) printf(" %d", ndigit[i]);
       printf(", white space = %d, other = %d\n", nwhite, nother);
}
```

1.7 Functions

- Subroutine, procedure
- Encapsulate some computation
  - Ignore how is done
  - know what is done
- printf, getchar
- C has no power operator
  - Function power()
  - print 2^i and $(-3)^i$
  - Not practical
  - Only positive powers, small integers
- pow(x, y) -> x^y
- Arguments, variable copy, local

```
#include <stdio.h>

int power(int base, 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;
}

int power(int base, int n) {
    int i, p;

    p = 1;
    for (i = 1; i<= n; ++i) {
        p = p * base;
    }

    return p;
}
```

4.1 Grep

- Print the lines that contains a "pattern"
- Split the problem in small parts
- getline (chapter 1); printf()
- strindex(s, t)
  - Return the index in the string s where t begins
  - -1 if s does not contain t

```
while (there's another line)
    if (the line contains the pattern)
            print  it
```

4.1 Grep

- Print the lines that contains a “pattern”
- Split the problem in small parts
- getline (chapter 1); printf()
- strindex(s, t)
    - Return the index in the string s where t begins
    - -1 if s does not contain t

```
int mygetline(char s[], int lim) {
    int c, i;

    i = 0;
    while (--lim > 0 && (c=getchar()) != EOF && c!= '\n')
        s[i++] = c;
    if (c == '\n')
        s[i++] = c;
    s[i] = '\0';
    return i;
}
```

4.1 Grep

- Print the lines that contains a “pattern”
- Split the problem in small parts
- getline (chapter 1); printf()
- strindex(s, t)
    - Return the index in the string s where t begins
    - -1 if s does not contain t

```
int strindex(char s [], char t []) {
    int i, j, k;

    for (i=0; s[i] != '\0'; i++) {
        for (j=i, k=0; t[k] != '\0' && s[j] == t[k]; j++, k++)
            ;
        if (k > 0 && t[k] == '\0')
            return i;
    }
    return -1;
}
```

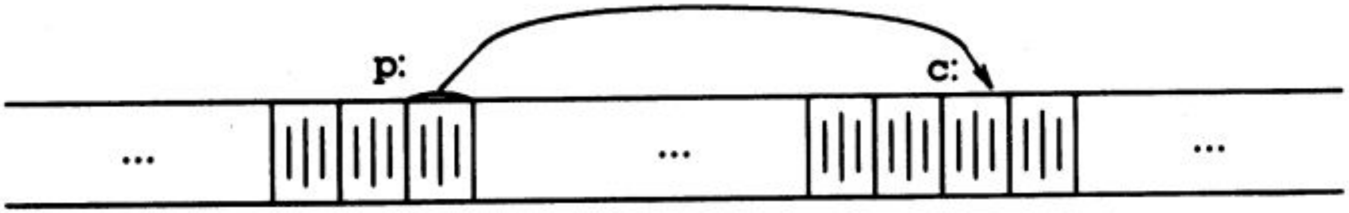
4.1 Grep

- Print the lines that contains a “pattern”
- Split the problem in small parts
- getline (chapter 1); printf()
- strindex(s, t)
  - Return the index in the string s where t begins
  - -1 if s does not contain t
- cat loren.txt
- ./a.out < loren.txt

```
int main() {
    char line[MAXLINE];
    int found = 0;

    while (mygetline(line, MAXLINE) > 0)
        if (strindex(line, pattern) >= 0) {
            printf("%s", line);
            found++;
        }
    return found;
}
```

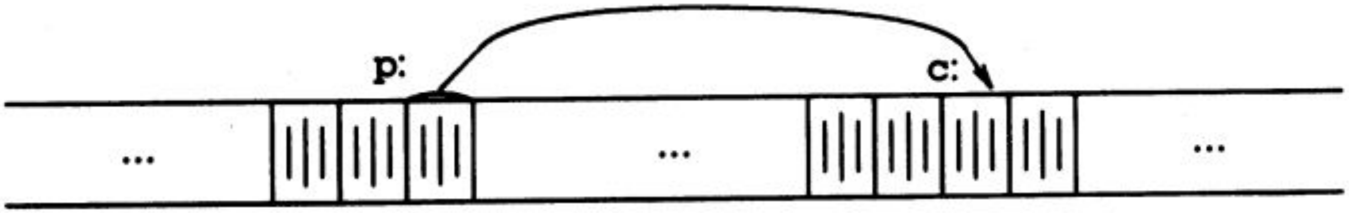
5.1 Pointers

- Memory is an consecutive numbered memory cells
- Pointer is a group of those cells that holds an address
- Operator & gives the address of something p = &c;
- Operator * access the object that pointer is pointing to



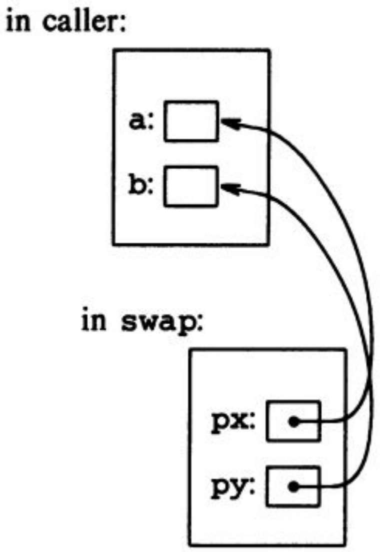
5.1 Pointers

```
int x = 1, y = 2;
int *ip;            // It reads "pointer to an integer"
                    // The expression "*ip" is an int
ip = &x;            // ip receives the address of x
y = *ip;            // copy the content of whatever ip points to y
*ip = *ip + 10;     // *ip affects x
```



5.2 Pointers and function arguments

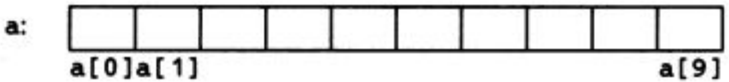
- C passes arguments to functions by value
- No direct way to alter the variable in the calling function (scope)



```
int swap(int x, int y) {
    int temp;

    temp = x;
    x = y;
    y = temp;
}

int pswap(int *px, int *py) {
    int temp;

    temp = *px;
    *px = *py;
    *py = temp;
}
```

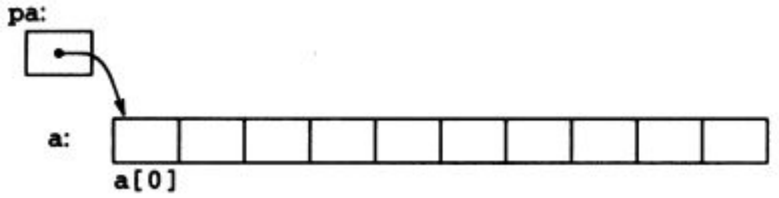
5.3 Pointers and arrays

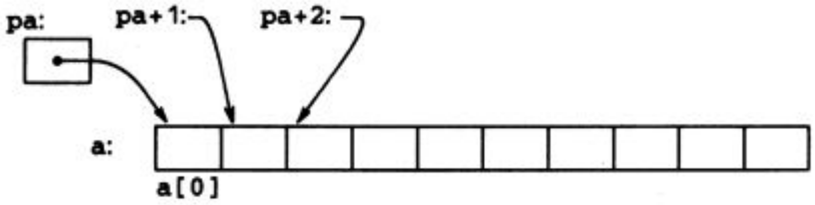
- Strong relationship between pointers and arrays
- Any operation that can be achieved indexing (arr[i]) can also be done with pointers;
- In general it will be faster with pointers



5.3 Pointers and arrays

- Strong relationship between pointers and arrays
- Any operation that can be achieved indexing (arr[i]) can also be done with pointers;
- In general it will be faster with pointers





Continue next class

- Install the environment
  - Test with hello_world
- Try the examples yourself