Introduction

Week 02 – Tutorial

C Language

Outline

- Learn basic C language concepts
- Answer your questions

Variables

A variable can be characterized with:

- name
- type
- address
- value
- lifetime
- scope

Problem 1

1. Can a variable name start with a number?
2. Can a variable name start with a typographical symbol (e.g. #, *, _)?
3. Give an example of a C variable name that would not work. Why doesn't it work?

Problem 1 - Answer

1. No, the name of a variable must begin with a letter (lowercase or uppercase), or an underscore
2. Only the underscore can be used
3. For example, #nm*rt is not allowed because # and * are not the valid characters for the name of a variable

Problem 2

1. List at least three data types in C
2. On your computer, how much memory does each require?
3. Which ones can be used in place of another? Why?
    1. Are there any limitations on these uses?
    2. If so, what are they?
    3. Is it necessary to do anything special to use the alternative?
4. Can the name we use for a data type (e.g. 'int', 'float') be used as a variable?

Problem 2 - Answer (1/3)

1. Data types: int, float, double

Code example:

```
#import <stdio.h>

int main()
{
   int a;
   float b;
   double c;
   printf("%lu, %lu, %lu\n", sizeof(a), sizeof(b),
sizeof(c));
   return 0;
}
```

Problem 2 - Answer (2/3)

2. On my computer:
    1. int: 4 bytes
    2. float: 4 bytes
    3. double: 8 bytes

Problem 2 - Answer (3/3)

3. We can use **int** instead of **float** and **double** and sometimes, vice versa. However, if you want to assign float or double value to int variable, you need to cast it and it will be truncated. Since float and double variables stored in memory as a combination of exponent and significant, their capacity is much greater than int. If a value exceeds maximum int capacity, some weird value will be stored inside the variable
4. We can not use 'int' or 'float' as a variable's name

Problem 3

- What is a structure of a standard C program?

Problem 3 - Answer (1/3)

- Let's consider an example:

```
// This is a sample program
// All rights reserved
```
Documentation section

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

```
#define PI 3.1415926
```
Definition section

```
int global_var;
```
Global declaration section

```
int main() {
  int local_var;
  …
  return 0;
}
```
declaration part

executable part

main() function section

```
void some_function { … }
```
Subprogram section (user-defined functions)

Problem 3 - Answer (2/3)

- **Documentation section** usually contains information about a program: the author, description, copyright etc.
- **Link section** provides instructions to the compiler to link functions from a library
- **Definition section** contains symbolic constants and macros
- **Global declaration section** contains declaration of global variables (explained later) and user-defined functions

Problem 3 - Answer (3/3)

- **main() function** is a function called when a C program runs. Every C program must have one main() function section
  - **declaration part** declares all the variables used in the executable part
  - **executable part** contains function calls, calculations and so on
- Subprogram section consists of all the user-defined functions if the program is a multi-function program
- **Important: all the sections except main() function section are optional**

Problem 4

- What are global variables, local variables and formal parameters?

Problem 4 - Answer (1/2)

- There is a term called **scope**, that means a region of a program inside which a variable exists. Beyond that scope a variable is not accessible
- There are three places where variables can be declared:
  - Outside of all functions: **global variables**
  - Inside a function: **local variables**
  - Definition of function parameters: **formal parameters**

Problem 4 - Answer (2/2)

- Example:

```
int g_var; // global variable                    Global variable

int main(int argc, char *argv[]) {               Formal parameters
  int l_var;                                     Local variable
  l_var = 0;
  g_var = 0;
  func();
  printf("Value of g_var is %d", g_var); // prints 1
}

void func() {
  g_var = g_var + 1;
  // l_var = l_var + 1;   // error
  // argc = argc + 1;   // error
}
```

Problem 5

- How does someone provide input to a C program?

Problem 5 - Answer (1/5)

- There are several ways to provide input to a C program:
  - provide parameters when running a program
  - **gets()** function
  - **scanf()** function
  - **getline()** function
  - **fgets()** function

Problem 5 - Answer (2/5)

- When you want to provide some input values once at the beginning of execution, you might want to pass them as parameters to a program by running your program this way:

```
>./a.out 42 13 12
```

and writing your main function like this:

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

- **argc** contains a number of the arguments and **argv** is a pointer to an array containing values of the arguments. The first parameter is a name of the program

Problem 5 - Answer (3/5)

- **Even if you can use gets() it is unsafe!**
- It may lead to buffer overflow (will be explained at the end of the course, or you can read about it whenever you want)
- During the labs you can use **scanf()** function, but don't use it in real life since it is **vulnerable to buffer overflow attacks too**

Problem 5 - Answer (4/5)

- Example of reading input using **scanf()** function:

```c
#include "stdio.h"

int main(void)
{
    int a;

    printf("Please input an integer value: ");
    scanf("%d", &a);
    printf("You entered: %d\n", a);

    return 0;
}
```

Problem 5 - Answer (5/5)

- Some people suggest using **getline()** function from C POSIX standard. However, in this case you loose portability, since this function is not implemented by Microsoft in their C version
- The only safe solution is to use **fgets()** function since it allows you to restrict size of an input string:

```
char name[10];
printf("Who are you? ");
fgets(name, 10, stdin);
```

Problem 6

- How does someone provide an output from a C program?

Problem 6 - Answer (1/2)

- Basically, there are two ways to send a string to screen:
  - **puts()** and **fputs()** functions
  - **printf()** family of functions
- **puts()** and **fputs()** functions are counterparts of **gets()** and **fgets()** and have almost similar syntax
- The difference between these two methods is that the latter ones provide formatting capabilities

Problem 6 - Answer (2/2)

- Examples of using printf():

```
// printf() will not work without this
#include <stdio.h>

int main() {
  printf("The answer is %d", 6*7);
  printf("PI = %f", 3.1415926);
  printf("Today is %d of %s", 24, "August");
}
```

- Try to run `$ man 3 printf` in bash

Problem 7

- What is the difference between passing arguments by value and by reference?

Problem 7 - Answer (1/4)

- Passing arguments by value:
  - copies **the actual value** of an argument into the formal parameter of a function
  - changes made to the parameter inside the function's body don't affect the argument that is passed

Problem 7 - Answer (2/4)

- Passing arguments by value example

```
int main(void) {
  int arg;
  arg = 0;
  printf("Value of arg is %d\n", arg);
  increment(arg);
  printf("Value of arg is %d\n", arg);
}

void increment(int num) {
  num = num + 1;
}
```

Problem 7 - Answer (3/4)

- Passing arguments by reference:
  - copies the address of an argument into the formal parameter of a function, not a value
  - the address is used inside a function to get the argument value and, possibly, alter it

Problem 7 - Answer (4/4)

- Passing arguments by reference example:

```
int main(void) {
  int arg;
  arg = 0;
  printf("Value of arg is %d\n", arg);
  increment(&arg);
  printf("Value of arg is %d\n", arg);
}

void increment(int *num) {
  *num = *num + 1;
}
```

End

Week 02 – Tutorial

References

- http://www.onlineclassnotes.com/2015/04/write-down-basic-structure-of-c.html
- http://www.tutorialspoint.com/cprogramming/c_scope_rules.htm
- http://stackoverflow.com/questions/3302255/c-scanf-vs-gets-vs-fgets
- https://en.wikibooks.org/wiki/C_Programming/Simple_input_and_output
- https://en.wikibooks.org/wiki/C_Programming/Beginning_exercises