CS 537 Discussion

13 September, 2023

Agenda

1. Review/ask questions about lecture material
2. Introduction to C programming
3. Project-1 discussion

Why C?

Operating systems, drivers, embedded, high-performance computing.

Examples: Linux kernel, Python, PHP, Perl, C#, Google search engine/Chrome/MapReduce/etc, Firefox

Issues with C

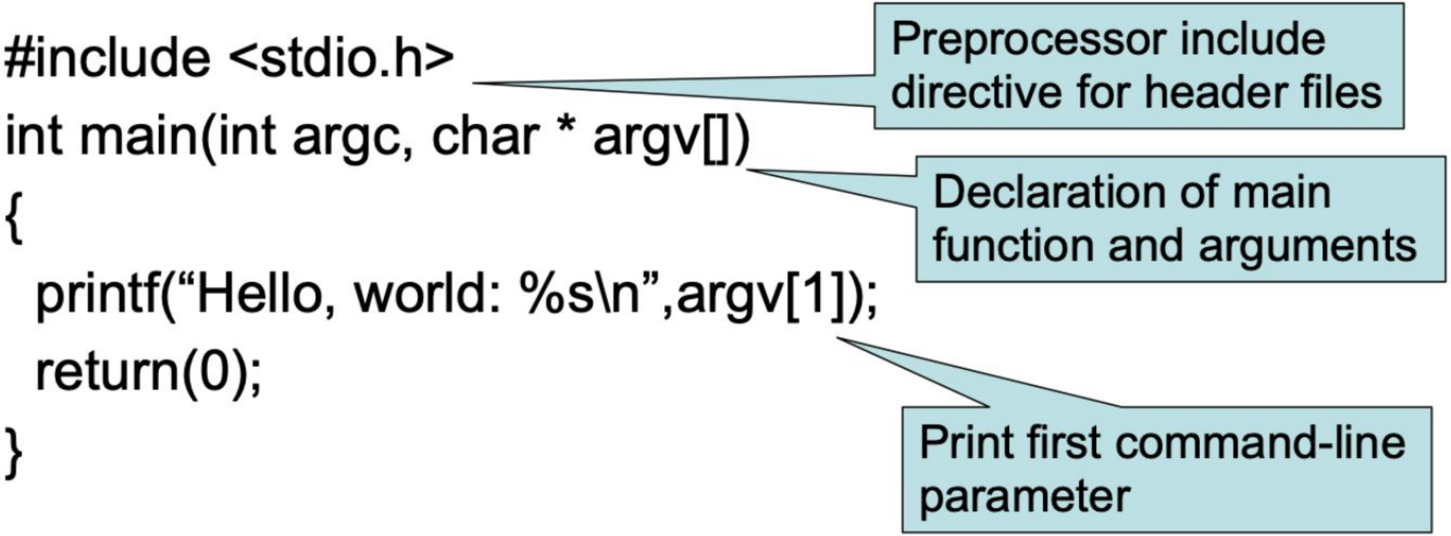
Little hand-holding for programmers

- Manual memory management
- Small standard library
- No native support for threads and concurrency
- Weak type checking

Builtin Types in C

Type	Size	Comment
char	1	ASCII character
int	4	Integer
long int	8	Longer Integer
float	4	Decimal number
double	8	Decimal number
long double	16	Even Longer decimal

C language



```
#include <stdio.h>
int main(int argc, char * argv[])
{
  printf(“Hello, world: %s\n”,argv[1]);
  return(0);
}
```

Preprocessor include directive for header files

Declaration of main function and arguments

Print first command-line parameter

Compiling C code

```
$ gcc hello-world.c
```

Compiling C code

```
$ gcc hello-world.c -Wall -Werror -O3 -g
```

1. -Wall: enables all the warnings about constructions that some users consider questionable, and that are easy to avoid
2. -Werror: Make all warnings into errors.
3. -O[x]:
   a. 0-3: optimization level with 0 being the lowest and 3 being the highest.
   b. s: optimize for binary size
   c. fast: all O3 optimization + some other unsafe optimizations
   d. g: optimize for debugging
4. -g: include debug info in the binary.

Linker

- Linking is required whenever we call functions not defined in the files we are compiling.
- In general, programs are linked against the C standard library, e.g. `glibc'.
- But what if we call functions that we have defined in other files?

Demo - compile separately and then link

What if we have a project with many files?

Makefile

```
 # Makefile
SRCS = myprog.c fn.c
TARG = myprog
CC = gcc
OPTS = -g

OBJS = $(SRCS:.c=.o)
$(TARG): $(OBJS)
$(CC) -o $(TARG) $(OBJS)

%.o: %.c
TAB $(CC) $(OPTS) -c $< -o $@

clean:
TAB rm -f $(OBJS) $(TARG)
```

A few notes:

- Indentations need to be tabs
- Makefiles usually have a bunch of definitions followed by target rules
- `$<': target being generated
- `$@': first prerequisite

Strings

- Strings in C are arrays of bytes.
  - char str[100]
- They are null terminated - so you need to make space for it.
  - str[0] = ‘\0’
  - strlen(str) = 0
- There are a bunch of functions to work with them:
  - strlen, strcpy, strcat

Memory

- You have to manage memory by yourself.
- Fixed-size variables can be allocated on a stack
    - The contents of these variables go away when the function returns:

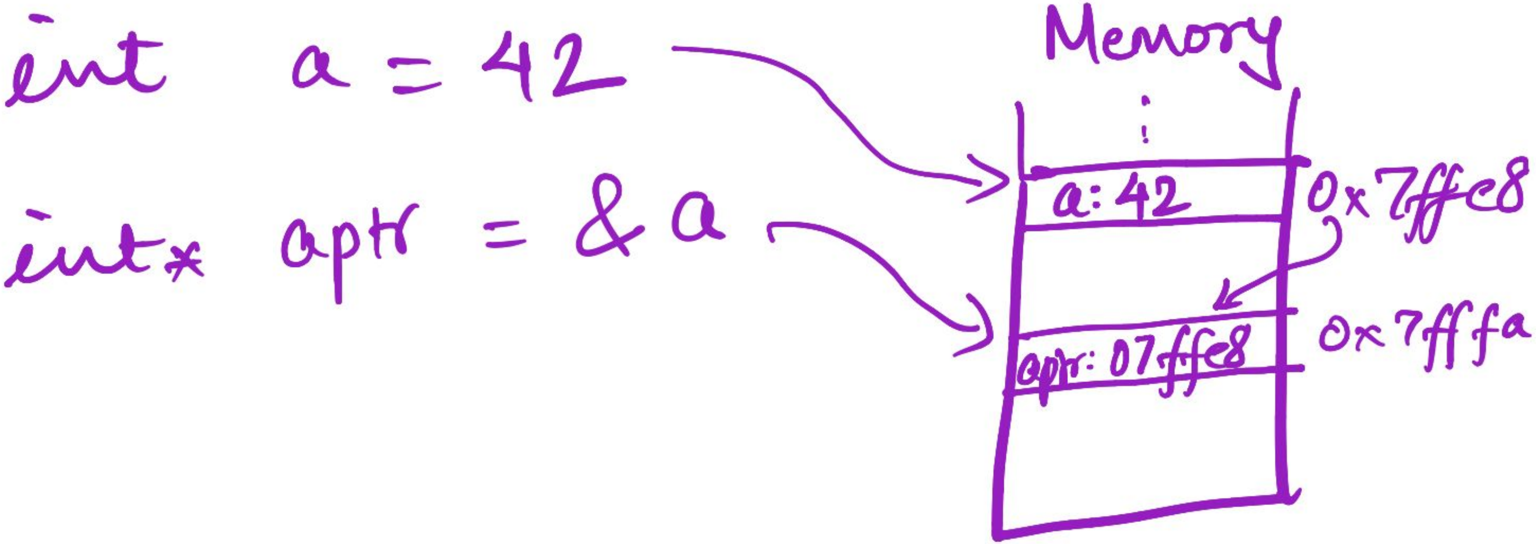
```
char str[100] = “hello, world\n”;
```

- Variable-size variables are allocated using **malloc** similar to new() in Java,
    - Memory from malloc only becomes invalid when you free it.

```
char  *str;

str = malloc(n);

strcpy(str, “hello, world\n”);

free(str)
```

Pointers in C

- Getting the memory (virtual) address of an object.



```
int a = 42
int* aptr = &a
```

File I/O

- Functions for accessing files:
    - struct FILE : represents an open file
    - FILE *f: declares a file pointer to handle and keep track on files being accessed
    - f = fopen(“foo”, “r”): opens file foo for reading
    - fclose(f): closes the file once done with f
    - fgets(buffer, n, f): reads n bytes from f into buffer
    - fputs(buffer, f): writes n bytes to f from buffer
    - fread(buffer, size, count, f): reads size x count bytes from f into buffer
    - fwrite(buffer, size count, f): writes size x count bytes to f from buffer

Demo

How to debug your programs

- Add print statements
    - Print things out all the time to see what is happening
    - Problem: this is hard for large input files
- Use a **debugger**
    - Allows you to stop your program while it is executing and see the contents of the all the your variables
        - You can say where to stop by adding breakpoints
    - GUI debuggers: Visual Studio
        - Shows lots of stuff in windows
    - Command line debuggers: gdb
        - You can enter command to see everything

Debugging using gdb

- Compile with debugging using “-g”: gcc -g hello.c
- Run the program with gdb

```
$ gdb  ./a.out
```

hello.c:

```
#include <stdio.h>
int main(int argc, char *argv[]){
  printf("Hello %s!\n", argv[1]);
  return 0;
}
```

Project-1

Objective:

- Re-familiarize yourself with the C programming language
  - Working with strings
  - Reading and Writing files
  - Working with structs
- Familiarize yourself with a shell / terminal / command-line of UNIX
- Learn about how UNIX command line utilities are implemented

Project-1 overview

- Topic: Unix Utilities
- Due Date: September 19th, at 11:59pm
- Implement the following utils
  - wman
  - wapropos
  - wgroff

Examples Demo

CSL machine

Login to CSL machine:

1. Connect to VPN
2. ssh <cs-login>@best-linux.cs.wisc.edu

Project submission

- Copy your files to ~cs537-1/handin/cslogin/P1.

  Example: cp wman.c ~cs537-1/handin/sunaina/P1/

- Files to submit:
  - Three .c files: wman.c, wapropos.c, wgroff.c
  - Compile successfully with -Wall and -Werror flags.
  - Add a README.md describing your implementation.

What does this C code do?

```
int minval(int A[], int n) {
  int cmin;
  for (int i=0; i<n; i++)
    if (A[i] < cmin)
      cmin = A[i];
  return cmin;
}
```

Find the issue

```
if (x = 0)
  y == 7; // assign y as 7 if x was 0
/********************************************************************************************/
int A[10];
int sum = 0;
for (int i = 0; i <= 10; i++) sum += A[i]; // sum of array `A`
/********************************************************************************************/
```