Introduction to C (part 1)

PHYS 795/895

W. Douglas Cramer

C Versions

- Versions
  - Kernighan & Ritchie C (1978)
  - C89(ANSI C)/C90(ISO C) (1989/1990) ← what we will use
  - C99
  - C11
- Compiling
  - GNU C: gcc <options> <c file list>
  - Options
    - '-ansi −pedantic': to enforce ANSI (C89/C90) standard
    - '-o <executable name>': default executable name is 'a.out'

Intro to C Language

- Big differences from FORTRAN
  - All routines are functions (default return type integer)
    - standard (int): 0=success, non-zero=failure code
  - Library functions not automatically included (must include)
  - Function arguments are pass-by-value, not pass-by-ref
  - Arrays are zero-based (start at 0, not 1) and row-major
  - 'void' type
  - Pointers (to memory addresses)!
  - Obviously, many syntax differences
    - Case sensitive
    - Loose formatting − columns, lines, and whitespace not important
      - Statement separator ';', Code blocks within '{…}'
    - Etc., Etc., Etc.

Simple Program

- Main routine must be called “main”
  - `int main(void) {…}`
  - `int main(int argc, char *argv[]) {…}` – to pass arguments
- Comments enclosed in `/*...*/`
- Compile with gcc (or other C compiler)

```
/* simple.c */
int main(void)
{
  return(16);
}
```

```
$ gcc –ansi –pedantic basic.c
$ ./a.out
$echo $?
16
```

Data Types/Variables

- Basic types
  - Integers ([unsigned/signed] int, short [int], long [int]) – 15, 17u, -312L, 077, 0xff (type of literals: smallest integer type able to hold)
  - Floating-point (float, double, long double) - 1.2e-3F, 4.5e26L
  - Characters ([unsigned/signed] char) – 'a'
  - Void (void) – explicitly defines lack of type/value
  - No logical (0=false, non-zero=true) or complex type
- Derived types (typedef, struct, union, enum)
- Variables
  - identifier any combination of a-z, A-Z, _, 0-9 (except reserved words)
- Declare (at beginning of block): <type> <variable>[=<value>][,...] (e.g., float var1)
- Type conversion: (<type>)<variable> (e.g., (float)ivar)
- No implicit type

Basic Input/Output

- Must include standard library io functions
  - `#include <stdio.h>` - 'h' means 'header' file, <> signifies library
- Some commonly used functions
  - `int putchar(int c)` – print character to stdout
  - `int getchar(void)` – read character from stdin
  - `int printf(char *format, arg1, arg2, ...)` – print to stdout
  - `int scanf(char *format, arg1, arg2, ...)` – read from stdin

```
/* basicio.c */
#include <stdio.h>
int main(void)
{
  int inchar, inval;
  printf("Input a character: ");
  inchar=getchar();
  printf("Here it is: "); putchar(inchar); printf("\n");
  printf("Input an integer: ");
  scanf("%d",&inval);
  printf("Here it is: %d\n",inval);
}
```

```
$ gcc basicio.c -o basicio
$ ./basicio
Input a character: d
Here it is: d
Input an integer: 45
Here it is: 45
$ echo $?
0
```

Operators

- Arithmetic: `-`, `*`, `/`, `%`, `+`, (unary) `-`, (incr/decrement) `++`, `--`
- Relational: `<`, `<=`, `>`, `>=`, `==`, `!=`
- Logical: `!`, `&&`, `||`
- Bitwise: `~`, `<<`, `>>`, `&`, `^`, `|`
- Assignment: `=`, `<oper>=` (e.g., `a+=1`)
- Pointer: `&` (address of), `*` (value at)
- Be careful of order of operations!!! (use parentheses)

```
/* operator.c */
#include <stdio.h>
#include <math.h>
int main(void)
{
  int val=3+5;
  printf("%d\n",val++);
  val=sqrt(val);
  printf("%d\n",++val);
}
```

```
$ gcc operator.c
$ ./a.out
8
4
```

Arrays

- List of values: {value1, value2, ...} (e.g., `{1,2,3,4}`)
- Declaration: `<type> <name>[dim1,[dim2...]] [={<init>}]`
  - e.g., `int array[5]={1,2,3,4,5}` (Note: uninitialized elements set to zero/null)
- Access: `<name>[<index1>[,<index2>...]]`
  - zero-based (e.g., `array[0]=1, array[4]=5`)
- Strings are character arrays
  - can initialize with double quotes (e.g., `char str[10]="hello\0"`)
  - must end with `'\0'`

```
$ gcc arrays.c
$ ./a.out
10 2
hello
```

```
/* arrays.c */
#include <stdio.h>
int main(void)
{
  int array[5]={1,2,3,4,5};
  char str[10]="hello\0";
  array[0]+=9;
  printf("%d %d\n",array[0],array[1]);
  printf("%s\n",str);
}
```

Pointers

- Values reside in memory
- A pointer is the memory address of the value
- Declaration: `<type> *<name>` (e.g., `int *ptr`)
- Operators
  - `&<variable>`: address of variable (1-dim array names are pointers)
  - `*<pointer>`: value at pointer (pointer type should match value being pointed at!!)

```
/* pointers.c */
#include <stdio.h>
int main(void)
{
  int val=20;
  int array[5]={1,2,3,4,5};
  int *ptr;
  ptr=&val;
  printf("%p %d\n",ptr,*ptr);
  printf("%p %d\n",array,*array);
}
```

```
$ gcc pointers.c
$ ./a.out
0x7fff5e3264fc 20
0x7fff5e326500 1
```

Program Control

- Decisions
  - `if(test)/else if(test)/else` – decisions
  - `switch/case test:/default:` – multi-case decisions
- Loops
  - `for(init;test;incr)` – loop until test fails
  - `while(test)` – loop while test (at start) passes
  - `do/while(test)` – loop while test (at end) passes
- Early exit
  - `break` – exit immediately
  - `continue` – in loops, skip to next iteration

Program Control Examples

```c
/* decisions.c */
#include <stdio.h>
int main(void)
{
  int inval, retval=1;
  printf("Enter a number between 1 and 10 :");
  scanf("%d",&inval);

  if (inval < 1 || inval > 10) {
    printf("Error: invalid number\n");
  }
  else {
    switch(inval) {
      case 2:
      case 3:
      case 5:
      case 7:
        printf("%d is a prime number\n",inval);
        break;
      default:
        printf("%d is NOT prime\n",inval);
        break;
    }
  }

  return retval;
}
```

```c
/* loops.c */
#include <stdio.h>
int main(int argc, char *argv[])
{
  int index;

  for (index = 0; index < argc; index++)
    printf("%s\n",argv[index]);

  index = 0;
  while (index < argc)
    printf("%s\n",argv[index++]);

  index = 0;
  do {
    printf("%s\n",argv[index++]);
  } while (index < argc);

  return 0;
}
```

Functions

- Header: `[return type] name([type arg1[,...]])`
  - Default return type is `int`
- Arguments are pass-by-value (i.e., a copy is made)
  - to alter an argument, must pass a pointer
- Execution: `[retval =]name([arg1,[...]])`
  - function or prototype/stub must precede use (for compiler)
- Return value: `return [expression]`
  - if `expression` not equal to return type, auto-converted

```
/* add.c */
float add(float x, float y)
{
  return x+y;
}
```

Including External Functions

- #include <stdio.h> or #include “basic.h”
  - # indicates preprocessor command
  - #include: include file contents prior to compiling
  - <>: library files; “”: user-defined files
    - common lib: stdlib.h, stdio.h, math.h, string.h
  - stdio.h contains library input/output functions
- File extension conventions
  - ‘.h’ files contain function stubs/prototypes
  - ‘.c’ files contain actual functions

```
/* include.c */
#include <stdio.h>
#include ”add.h"
int main(void)
{
  printf("%f\n",add(5,8));
}
```

```
/* add.h */
float add(float x, float y);
```

```
/* add.c */
float add(float x, float y)
{
  return(x+y);
}
```