

CSI333 Lecture 3

What we did.

Lect 01

A non-trivial algorithm to program which uses the core data structures (1) stacks and (2) dictionary.

Lab 01

Intro. to knowledgeable usage of our virtual Unix lab with multiple windows for developing C programs.

Lect 02

Review of basic Object Oriented structuring of the Project 1 program.

Programming Project 2

- ▶ Start using C on Unix.
- ▶ Input 32-bit words from the user in several different ways.
- ▶ See those bits in two common ways: Hexadecimal and the bits as 0s and 1s.
- ▶ Output (i.e., interpret) the 32-bit words in several different ways.
- ▶ Challenge: Print the number in English.

Loop until commanded to exit:

1. Print a menu and receive an input choice, or command to use currently stored data.
2. (exit, or) Run an input operation according to that choice.
3. Print the input bits.
4. Print a menu and receive an output choice.
5. Run an output operation according to that choice.

Formatted Input and Output in C—very simple

```
#include <stdio.h> /*REASON: get declarations of
scanf & printf   from SYSTEM HEADER FILE stdio.h.
Those declarations are approx.
extern int scanf(const char *, ... );
extern int printf(const char *, ... );
*/
... main ( ... ) /* declare and */
{               /* DEFINE the function main */
int Ivar;      /*declare and define Ivar*/
scanf("%d", &Ivar);
printf("Hello. Your int*2 is %d\n", 2*Ivar );
return 0;     /* success return to shell */
}
```

with `scanf(...);`

```
int Ivar;           /*declare and define Ivar*/  
scanf("%d", &Ivar);  
printf("Hello. Your int*2 is %d\n", 2*Ivar );
```

1. The **format string** (like "%d") specifies HOW to interpret fields of the input stream, including their conversion into bits.
2. You must provide **ADDRESSES, i.e., pointer values** to specify **where** the data will be delivered.

C unlike C++ has no REFERENCE parameters

- ▶ We mimick them: pass **the address of** the argument variable where we want data delivered. **&theVariable** means “address of variable **theVariable**”.
- ▶ The function **dereferences** the address to access the argument variable: **YOU** must code `*pointerParameter` to access what that address addresses!

```
void MakeMe23( int *pInt)  { *pInt = 23; }  
...  
int iVar;  
printf("%d\n", iVar ); /*prints GARBAGE*/  
MakeMe23(      &iVar );  
printf("%d\n", iVar ); /*prints 23*/
```

char I/O

scanf/printf character format

```
char cVar;  
scanf("%c", &cVar);  
printf("You typed \" %c \".\n", cVar);  
printf("Next in the ASCII code is %c\n", cVar + 1);
```

getchar()

```
printf("Type a char:");  
printf("Hey, you typed %c!\n",  getchar( ) );
```

??? getchar() returns an int value, printf interprets it as a char and prints that.

Unformatted char, ie., bitwise fread and fwrite

```
int iVar;  
fread( &iVar, 1, 4, STDIN ); /*Read ONE RECORD of 4 BYTES  
                             copy into iVar*/  
fwrite(&iVar, 1, 4, STDOUT); /*Write out the 4 BYTES  
                             in iVar*/
```

C-string format with scanf/printf

```
char myCString[4];
int intVar;
scanf("%3s", &intVar ); /*reads up to 3 chars
and stores them PLUS \0 in the 4-byte var. intVar*/
scanf("%3s", myCString); /*DITTO into the 4-byte
byte array*/
/* DIFFERENT from other output formats! */
printf("%s", &intVar); /* Print a C-String! */
printf("%s", myCString); /* Another C-string */
```

C strings are null-terminated char arrays

- ▶ They go by the address of their first char.
- ▶ In C/C++, with array `char myCharArray[56];`
`myCharArray` (no brackets!) denotes the (const) ADDRESS OF the first character.
- ▶ `myCharArray` is **equivalent to** `&myCharArray[0]`