POINTERS IN C

Now begins the fun part…He he he… >:)

First, recall...

- Computers have *memory*
- Memory is divided into *memory cells* or *memory locations*
- Each memory cell has a *single, unique address*
- When a variable is declared, it is allocated to a memory cell which will hold its value

WHAT ARE POINTER VARIABLES?

- Variables that can hold addresses of other variables
- To declare pointers:

```
<data type> *<variable name>;
```

- Examples:
  - `int *p;`
  - `float *q;`
  - `char *r;`

RECALL: ADDRESS OPERATOR

- Used to extract a variable's address
- Usage: &<variable name>
- Example:

```
int x, *p;
p = &x
```

- Can be interpreted as "the address of the variable <variable name>"

RECALL: INDIRECTION OPERATOR

- Used to access variables through pointers
- Usage: *<variable name>
- Example:

```
int x, *p;
p = &x;
(*p) = 5;
```

- Can be interpreted as “the value in the address in <variable name>”

REMARKS

- Address and indirection operators cancel each other out
  - `&(*x) = *(&x) = x`
- Pointers can only access variables of the same data type
  - Example:

```
int *p; //p can only access
        //integer variables
float *q; //q can only access
          //float variables
```

POINTER INITIALIZATION

- Recall that…
  - C does not initialize variables when they are declared
  - Uninitialized variables often contain *garbage values*
- With pointers, garbage values may be interpreted as memory addresses
- Can lead to run-time errors due to access violations
- Always assign a valid address to a pointer before using it
- If no address can be assigned, use **NULL**

SO, WHAT ARE POINTERS FOR?

- The usual BSCS student answer is…

"Ma'am, para pahirapin ang buhay KOMSAY."

- Naaaaaaaaah. :p
- Pointers are used for *pass-by-reference parameter* passing and *dynamic memory allocation*
- But more on that later. ☺

POINTERS TO POINTERS

- Pointers can also point to other pointers
- Example:

```
int x, *p, **q;
p = &x;
q = &p;
```

q is a double int pointer; it can point to integer pointers

p is an integer pointer; it can point to integer variables

- Given the address table, answer the ff.:
  - What is the value of *p?
  - What is the value of *q?
  - What is the value of **q?

Address	Value
12	6638687131384335 4
13	12
14	13

DEREFERENCE TYPE COMPATIBILITY

- *Dereference Type* – data type of the variable the pointer is referencing
- Invalid to assign a pointer of one type to a pointer of another type
- Example:

```
int *p; float *q;
p = q; //INVALID, since p is
       //an integer pointer and
       //q is a float pointer
```

- Exception: Pointers to ***void***

VOID POINTERS

- Pointers declared as

```
void *<variable name>;
```

- Generic pointer
- Can be used for *assignment statements* with all other pointer types
- Example:

```
int x, *p; void *q;
p = &x;
q = (void *) p;
```

- However, void pointers *cannot be dereferenced*

RETURNING POINTERS FROM FUNCTIONS

- Do NOT return a pointer to a local variable of the called function
- Example:

```
main()
{
  int *x;

  x = getInput();
}
```

x receives the address return by getInput

```
int *getInput()
{
  int x;

  printf("1. Madali ang CMSC 21.\n");
  printf("2. Papasa ako sa CMSC 21.\n");
  printf("Enter your opinion: ");
  scanf("%d", &x);
  return &x;
}
```

getInput returns an integer pointer

returns the address of the LOCAL VARIABLE x