Pointers

Summary Chapter 6

Normal - int a | Pointer - int *a

a = value | *a = value

&a = address | a = address

&a = address of pointer.

Double Indirection

```
int **pp;
int *p;
pp = &p;
```

Chapter 5.1 1D Array

Variables

Primitive → Store values. eg. int, char, float

Reference → Store addresses. eg. pointer, array, char string

Array name → pointer constant → address.

days

days = &days[0] | *days = days[0]

days+1 = &days[1] | *days+1 = days[1]

Address. | Values

Array as func arg.

→ void fn(int table[], int n);

→ void fn(int table[TABLESIZE]);

→ void fn(int *table, int n);

Notations

→ Index Implementation

→ Array base addr

→ pointer var. notation

Chp 5.2 Multi-dimensional Array

Memory Layout

a[row][column]

↑ always first

Pointers

→ Stored sequentially.

ar[2][1] = *(*ar+index)

index = (row*(col.size)+col)

Or

ar[2][1] = *(*(ar+m)+n)

= *(*(ar+2)+1)

Initialization

```
int x[2][2] = {{1,2},
               {6,7}};

int x[2][2] = {1,2,6,7};
```

Partial Init

```
int exam[3][3] = {{1,2},
                  {4},{5,7}};
```

* Can omit outermost dimension.

Recursion

→ Returning Value

```
int multi(int m, int n)
{
  if(n==1)
    return m;
  else{
    return m+multi(m,n-1);
  }
}
```

→ Call by Reference

```
void multi(int m, int n, int *prod)
{
  if(n==1)
    return m;
  else{
    multi(m,n-1, prod);
    *prod = *prod + m;
  }
}
```

→ Recursion in Arrays

```
int recursiveSum(int a[], int size)
{
  if(size==1)
    return a[0];
  else
    return a[size-1] + recursive
    Sum(a, size-1);
}
```

→ Design Recursive Funcs

→ Key step (Recursive condition)

→ Stopping Rule (Terminating condition)

Recursion or Iteration?

Advantages

→ Short & clear code

Disadvantage

→ Expensive (memory) to run.

* Any problem solved recursively can be solved Iteratively

Strings 1

str == &str[0] # Similar to array, if contains an address

Declaration → Ending with '\0' = NULL

Array: char str1[]

Pointer: char *str1

Not Allowed ∈ ++str1

str1 = <new Addr> → Allowed

Using scanf, & is neglgent because it is an address.

String Input
→ gets(name)
→ printf("%s", name)

String Output
→ puts(name)
→ scanf("%s", name)

String Functions

strcat()

strncat() → Append s2 to end of s1

strchr()

strrchr()

strcmp() → Compares strings s1 & s2

strncmp()

strcpy() → Copies s2 content to s1

strncpy()

strcspn()

strstr()

strlen() → Returns length of string

strpbrk()

strtok()

Strings 2

ctype.h

Boolean Functions

isalnum → Alphanumeric
isalpha → Alphabet
iscntrl → Control Character (Ctrl B
isdigit → Numeric
isgraph → Any print char, exclude space
islower → lowercase char.
isprint → Print Char
ispunct → punctuation. (Any print char excluding alpha-numeric & space
isspace → whitespace char \n, \t, ...
isupper → Uppercase char
isxdigit → Hexadec. char.

Character Conversion

toupper() → All lowercase char to uppercase

tolower() → All uppercase to lowercase

stdlib.h → Str to Num Conversion

atof() → String to Double

atoi() → String to Int

Array of Char Strings

Ragged Array → save storage

char *nameptr[4]

Rectangular Array

char name[4][8]