CS1100
Introduction to Programming

Other Basic Data Types: Enum etc
Arrays (1D, 2D, Matrices)
Strings

Recap

- **Conditional Statements:**
  - **if:** execute me **if** you are satisfied.
  - **if-else:** execute then/else
  - **switch:** execute one of the many
- **Repetitive statements:**
  - **for**: used typically **for** count based looping
  - **while**: used for sentinel-based looping
  - **do-while**: **do** use it once in a **while**

BASIC TYPES IN C

Data, Types, Sizes, Values

- *int, char, float, double*
- *char* – one byte, capable of holding one character
- *int* – an integer, different kinds exist!
  - Integer Qualifiers – short and long
  - *short int* – 16 bits, *long int* – 32 bits (Typical)
  - Size is compiler dependent
    - based on the underlying hardware

The Char, Signed and Unsigned Types

- Char: data type that is stored in 1 byte (8 bits)
- ASCII Character Set uses 7 bits ➔ 128 characters
- signed or unsigned *char*
  - Unsigned numbers are non-negative
- Signed *char* holds numbers between –128 and 127
  - Whether *char* is signed or unsigned depends on the system. Find out on your system.
  - Print integers between 0 to 255 as characters, (and also integers between –128 to 127) on your system.

ASCII Character Set

Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1 Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2 Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3 Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4 Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Character Constants …

- …are written as one character within single quote
  - 'a', 'D', 'x', '1', '2' etc.
- Value of a character constant is the numeric value of the character in the machine's character set.
  - For example, '1' has value 49 in ASCII; 'A' is 65
- Character constants can participate in arithmetic
  - char ch='A'; ch += 32; value in ch is now 'a'
  - What does '1'+ '2' hold? (not '3'!)
    - Understand this distinction
  - Character arithmetic is used mainly for comparisons

Characters – escape sequences

\a	alert (bell)	\\	backslash
\b	backspace	\?	question mark
\f	formfeed	\'	single quote
\n	newline	\"	double quote
\r	carriage return	\ooo	octal number
\t	horizontal tab	\xhh	hexadecimal
\v	vertical tab		

Non-printable characters

CONSTANTS

Constants

- At run time, each variable holds a value, which changes from time to time
- Constant has a value that does not change
- 1234 is of type int
- 123456789L is a long constant
- 123456789ul is an unsigned long constant
- 123.4 is a floating point constant, so is 1e-2 which denotes .01. Their type is double.
- If suffixed by an f, or by l, the type is float or long double, respectively

Constants Expressions

- Expressions all of whose operands are constants
  - These can be evaluated at compile time
- Examples:

```
#define NUM_ROWS 100
#define NUM_COLS 100
#define MAXVAL 10000000L
#define MINVAL 240UL
#define PI 3.15149L // double precision
#define VARIANCE 2.3456E-7
#define NUM_ELTS  NUM_ROWS*NUM_COLS
```

- ***#define*** is preprocessor directive (recall ***#include***)
- Try running cpp file.c and observe output

Enumerated Constants

By default enum values begin with 0

- ***enum*** boolean {No, Yes};
  - defines two constants No = 0, and Yes = 1.
- ***enum*** months {jan = 1, feb, march, april, may, jun, jul, aug, sep, oct, nov, dec};
  - when a value is explicitly specified (jan=1) then it starts counting from there
  - ***enum*** months2 {jan=11, feb, mar, apr, may, jul=21, aug, sep, oct, nov};
- ***enum*** escapes {BELL = '\a', BACKSPACE = '\b', TAB = '\t', NEWLINE ='\n'};

enum and #define

- Better than ***#define***, the constant values are generated for us
  - Values start from 0 unless specified otherwise
  - Not all values need to be specified
  - If some values are not specified, they are obtained by increments from the last specified value
- Variables of ***enum*** type may be declared
  - but the compilers need not check that what you store is a valid value for enumeration

Declaring Constants

- The qualifier ***const*** applied to a declaration specifies that the value will not be changed.
  - Example: ***const int*** J = 25;
    /* J is a constant through out the program */
- Response to modifying J depends on the system. Typically, a warning message is issued while compilation.
  - Example: ***const char*** MESG[] = "how are you?";
  - The character array MESG is declared as a constant which will store "how are you?"

Variable Initialization

- Variables may be initialized either at the time of declaration
  - Example:
```
#define MAXLINE 200
char esc = '\\';
int i = 0;
int limit = MAXLINE + 1;
float eps = 1.0e-5;
```
- Or they may be assigned values by assignment statements in the program
- Otherwise they contain some random values

Exercise Problem

- Write a program to read 10 characters and print their ASCII value.
- Write a program to repeatedly read a character and print its ASCII value till the character Z is entered.
- Write a program to print the complete ASCII table.

ARRAYS AND STRINGS

Reading 10 numbers

int marks1, mark2, marks3, …., marks10;

scanf("%d%d%d….%d", &marks1, &marks2, &marks3, …, & marks10);

Instead, we can use an array type:

int marks[10];

An Array

- A data structure containing items of same data type
- Declaration: array name, storage reservation
  - ***int*** marks[7] = {22,15,75,56,10,33,45};
    - a contiguous group of memory locations
    - named "marks" for holding 7 integer items
  - elements/components - variables
    - marks[0], marks[1], … , marks[6]
    - marks[*i*], where *i* is an index (0<=i<=6)
  - the value of marks[2] is 75
  - new values can be assigned to elements
    - marks[3] = 36;

22	0
15	1
75	2
56	3
10	4
33	5
45	6

Fun question.

Attendance question: What is the output of the following program?

```
#include <stdio.h>
int main()
{
  char c = 70;
  char mesg[]="After";
  printf("%d %c %c", c, c, mesg[1]);
}
```

Example using Arrays

Read ten numbers *into an array* and compute their average

```
#include <stdio.h>
Int main( ){
   int numbers[10], sum = 0, i;
   float average;
   for ( i = 0; i < 10; i++)
        scanf("%d", &numbers[i]);
   for ( i = 0; i < 10; i++)
        sum = sum + numbers[i];
   average = (float) sum/10;
   printf("The average of numbers is: %f\n", average);
   return 0;  /* good to have this in all programs */
}
```


Counting Digits in Text (Kernighan & Ritchie, pp. 59)

```
#include<stdio.h>
main( ){
   int c, i, nWhite, nOther, nDigit[10];
   nWhite = nOther = 0;
   for(i=0;i<10;i++) nDigit[i]=0;
   while((c = getchar( ))! = EOF){
    switch(c){
       case'0':case'1':case'2':case'3':case'4':case'5':
       case'6':case'7':case'8':case'9': nDigit[c-'0']++;
                       break;
```

An array of ten integers

nDigit[0] – stores count of 0

nDigit[7] – stores count of 7

'7' -> index 7?

c – '0'

ASCII for '7' is 55; ASCII for '0' is 48

'7' – '0' = 55 – 48 = 7

… Counting digits

```
       case' ':
       case'\n':
       case'\t': nWhite++; break;
       default : nOther++; break;
    }
  }
  printf("Digits: \n")
  for(i=0;i<10;i++)
       printf("%d occurred %d times \n", i, nDigit[i]);
  printf("White spaces: %d, other: %d\n", nWhite,
                  nOther);
}
```

Polynomial Evaluation (Self-Reading)

Evaluate

$p(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots a_1 x + a_0$

at a given x value.

Computing each term and summing up

$n + (n-1) + (n-2) + \ldots + 1 + 0 = n(n+1)/2$ multiplications and n additions

Improved Method:

$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \ldots + x(a_{n-2} + x(a_{n-1} + xa_n))\ldots)))$

for instance, $p(x) = 10x^3 + 4x^2 + 5x + 2$

$= 2 + x(5 + x(4 + 10x))$

n multiplications and n additions – will run faster!

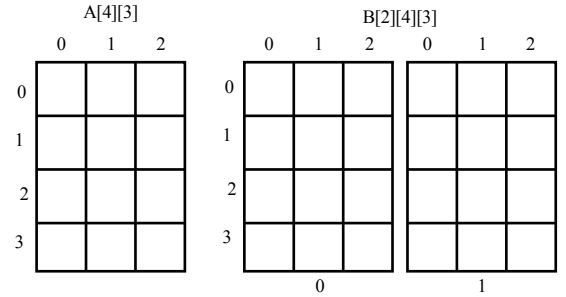
… Polynomial Evaluation

```
#include <stdio.h>
main( ){
  int coeff[20], n, x, value, i;        /*max. no. of coeff 's is 20*/
  scanf("%d%d", &n, &x);   /*read degree and evaluation point*/
  for(i = 0; i <= n; i++)
     scanf("%d", &coeff[i]);          /* read in the coefficients */
                                      /* a0, a1, … , an  */
  value = coeff[n];                    /* an */
  for(i = (n-1); i >= 0; i--)          /* evaluate p(x) */
     value = x*value + coeff[i];
  printf("The value of p(x) at x = %d is %d\n", x, value);
}
```


MULTI-DIMENSIONAL ARRAYS

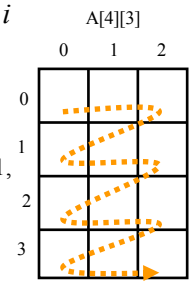
Multi-Dimensional Arrays

- Arrays with two or more dimensions can be defined



Two Dimensional Arrays

- Declaration: int A[4][3] : 4 rows and 3 columns, 4ξ3 array
- Elements: A[*i*][*j*] - element in row *i* and column *j* of array A
- Rows/columns numbered from 0
- Storage: row-major ordering
  - elements of row 0, elements of row 1, etc
- Initialization:
  int B[2][3]={{4,5,6},{0,3,5}};



Two Dimensional Arrays

- int array2d[5][4];
  - 2D array with 5 rows and 4 columns = 20 elements
  - Row index: 0 .. 4 and Column index: 0 .. 3
  - array2d[1][3] = 4; array2d[4][2] = 81;
  - a = b + array2d[2][1];

	0	1	2	3
0	4	7	2	8
1	9	11	1	4
2	6	8	12	24
3	1	4	5	9
4	13	27	81	34

Initializing 2D arrays

- int a[3][2] = { { 1 , 4 }, { 5 , 2 }, { 6 , 5 } };
  - **Recommended – initialize all values explicitly**
- int a[3][2] = {1 , 4 , 5 , 2 , 6 , 5 };
  - Stored **in row-major order** (Risky to assume)
  - a[0][0] = 1; a[0][1] = 4;
  - a[1][0] = 5; a[1][1] = 2; etc.
- int a[3][2] = { { 1 }, { 5 , 2 }, { 6 } };
  - Some elements are not initialized explicitly
  - They are initialized to 0; a[0][1] = 0; a[2][1]=0;

Higher order dimensional arrays

- double array3d[100][50][75];
- double array4d[60][100][50][75];
  - Requires 60*100*50*75*8 = 171.66 MB
- Array4d[3][34][45][56] = 4.12;
- Find out how many dimensions your system/compiler can handle.

Matrix Operations

- An *m*-by-*n* matrix *M*: *m* rows and *n* columns
- Rows: 1, 2, … , *m* and Columns: 1, 2, … , *n*
- *M*(*i*, *j*): element in i^{th} row, j^{th} col., *1*<= *i* <=*m*, *1*<=*j* <=*n*
- Array indexes in C language start with 0
- Use (*m*+1)ξ(*n*+1) array and ignore cells (0,*i*), (*j*,0)
- Programs can use natural convention – easier to understand

Using Matrix Operations

```
main( ){
   int mat[11][11], mat2[11][11]; /*max size: 10 by 10 */
   int mat3[11][11];
   int rows, cols;
   scanf("%d%d", &rows, &cols);

  for (int i = 1; i <= rows; i++)
     for (int j = 1; j <= cols; j++)
        scanf("%d", &mat[i][j]);
for (int i = 1; i <= rows; i++)
     for (int j = 1; j <= cols; j++)
        scanf("%d", &mat2[i][j]);
```


Writing a Matrix

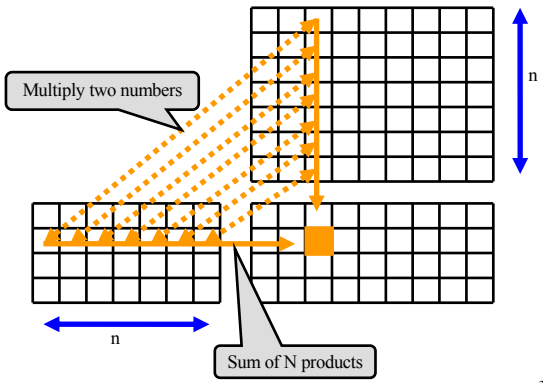
```
 for (int i = 1; i <= rows; i++){
     for (int j = 1; j <= cols;  j++)    /* print a row */
        printf ("%d ", mat[i][j]);    /* notice missing \n */
    printf ("\n");              /* print a newline at the end a row
 */
   }

 for (int i = 1; i <= rows; i++){
     for (int j = 1; j <= cols;  j++)
      mat3[i][j]=mat2[i][j]+mat[i][j]; // Addition
}
```

Attendance Q for Sep 20: Modify this line to subtract mat from mat2

```
} // End main
```


Matrix Multiplication



Matrix Multiplication: m x n and n x p matrices

```
int mat1[11][6], mat2[6][16],  mat3[11][16];
int m=10, n=5, p=15, i, j, k;
for (i =1; i <= m; i++)// Ignore 0th row
    for (j = 1; j <= p; j++)//Ignore 0th column
      mat3[i][j] = 0;
for (i =1; i <= m; i++)
    for (j = 1; j <= p; j++)
       for (k = 1; k <= n; k++)
           mat3[i][j] += mat1[i][k]*mat2[k][j];
```


STRINGS

Strings

- A string is a **array** of **characters** terminated by the **NULL character, '\0' – ASCII value of 0**
- A string is written in double quotes
  - Example: "This is a string"
- char str[4]="cat";
  - str[0] contains 'c'
  - str[1] contains 'a'
  - str[2] contains 't'
  - str[3] contains '\0' – VERY IMPORTANT
- " " – empty string (with NULL character)

Strings, contd.

- char strex[3] = "bar";
  - INCORRECT: NO space for NULL terminator
  - Will cause problems for string-handling routines that expect NULL-terminated strings
- char strex[] = "Hey, you!"; is correct and safer
- char c[]={'a','b','c','d','\0'}; is correct
- Anything within single quotes (e.g. 'G') gets a number associated with it based on char. code
  - 'This is rejected by the C Compiler'
- Exercise: difference between 'x' and "x"???

Functions to Handle Strings

- Strings
  - a non basic data type, a constructed data type
  - we require functions to handle them
- Typical functions are:
  - Length; comparison; concatenation, etc.
- Standard functions are provided with *string.h*
  - *strlen*( ); *strcmp*( ); *strcpy*( ); *strncpy*( ); *strcat*( ); *strncat*( );
- To input strings:
  - scanf("%s", str);
  - fgets(), getline(), fgetc, getchar, etc. – **NEVER use gets()**

Reading string using scanf()

```
#include <stdio.h>
int main()
{
  char name[20];
  printf("Enter your name: ");
  scanf("%s",name);
  printf("Your name is %s",name);
  return 0;
}
```

Enter your name: King Kong

OUTPUT will be:
Your name is King

Reason: scanf stops after seeing Space after King
(Kong is stored for next scanf)

Fun Question

What C function can we use to read an entire line of input from the user?

Using getchar() function – stores \n in array

```
#include <stdio.h>
int main(){
   char name[30], ch;
   int i=0;
   printf("Enter name: ");
   while(ch!='\n')    // terminates if user hit enter
   {
      ch=getchar();
      name[i]=ch;
      i++;
   }
   name[i]='\0';      // inserting null character at end
   printf("Name: %s",name);
   return 0;
}
```

Using getchar() function – does not store \n in array

```
#include <stdio.h>
int main(){
   char name[30], ch;
   int i=0;
   printf("Enter name: ");
   while(ch!='\n')    // terminates if user hit enter
   {
      ch=getchar();
      if (ch != '\n')
       {
       name[i]=ch;
       i++;
      }
   }
   name[i]='\0';      // inserting null character at end
   printf("Name: %s",name);
   return 0;
}
```


Using fgets(): reads an entire line till '\n'

```
#include <stdio.h>
#include <string.h>
int main(){
   const int STRLENGTH = 100;
   char inputstr[STRLENGTH];

   printf("Enter a string: ");
   fgets(inputstr, STRLENGTH, stdin);
   printf("String Output: %s, Length: %d", inputstr,
   strlen(inputstr);
   return 0;
}
```


Operations on strings

- Addition (Concatenation)
  - "Hello" + "world!" = "Helloworld!"
  - strcat() and strncat()
- "Hi" and "hi"
- Compare 2 strings
  - strcmp(), strncmp()
- Determine presence of substrings
  - strstr()

Examples with strlen, strcmp, strcat

```
{
…
 if (strcmp(inputstring, str) == 0)
    printf("%s %s\n", inputstring, str2);
  else
    printf("Please change your name\n");

  printf("%s\n", strcat(inputstring, str3));
}
```

Example with enum, array of strings

From Deitel and Deitel

See program shown in class