Lecture 2
C Programming Language

Summary of Lecture 2

- Relational and logic operations
- More C data types
- Introduction to arrays and pointers
- Function arguments, main() arguments

Characters

- Characters constants are given in single quotes: ‘a’, ‘B’, ‘$’ are characters
- The characters constants are represented numerically:
  ```
  char c = ‘a’;
  int k = (c < ‘b’); /* =1 if c<‘b’, else = 0 */
  ```
- Numerical values of characters - ASCII - see appendix 4 or any other ASCII table
- There are special characters like:
  ‘\n’ - end of line
  ‘\t’ - tab
  ‘\0’ - null character (to be continued..)

Relational and Logic operations

- Relational expressions:
  a>b, a<b, a<=b, a>=b, a==b, a!=b
  these expressions all have values,
  true or false (0 or 1)
  Thus the following is legal :
  printf(“%d”, a>b);

- Logic expressions:

a||b	a or b
a&&b	a and b
!a	not a

- Note:
  (test) ? stmt1 : stmt2; is equal to:

```
if (test)
      stmt1;
else
      stmt2;
```

Bitwise Operations

- **bitwise expressions:**

  a | b   a “or” b

  a & b   a “and” b

  Example:

  a = 00000110

  b = 00000011

  a | b = 00000111

  a &b = 00000010

- **Shift operations:**

  << left shift

  >> right shift

  Example:

  j=3;   j = 00000011

  k = j<<2;   k = 00001100 (k=12)

  m = j>>2;   m = 00000000 (m=0)

Integer Division

- 5 / 2 = 2 (5 divided by two)
  3 / 2 = 1 (note: ignore remainder)
- 5 % 2 = 1 (5 modulo 2)
  8 % 3 = 2 (remainder of 8/3)
- <u>Example:</u>

```
main() {
        int counter = 0;
        int letter = ‘A’;
        while (letter <= ‘Z’) {
            printf(“%c “,letter);
            counter++; letter++;
            if (counter % 6 == 0)
                    printf(“\n”);
        }
}
```

  This program prints the alphabet in the format of 6 letters in every line.

Implicit / Explicit Conversions

- **Explicit conversion:**
  (type)variable
  Example:
  ```
  int j =3;
  float f = 5.0;
  float d = (float)j / f;  /* d = 0.6 */
  ```
- **Implicit conversion:**
  ```
  int j =3;
  float f = 5.0;
  float d = j / f;  /* d = 0.6 */
  ```
  there will be no integer division, j is implicitly converted to a float
- Not all machines support conversions between doubles and floats, so use either one (only floats or only doubles).

Unsigned Data Types

- Typically half the values represented by a data type are negative (one sign bit)
- Example: char data types hold values from -127 to 127

  unsigned char data types hold numbers from 0 to 255
- unsigned data types should be used with caution :

  ```
  unsigned int j = 0;
  int k = -1;
  if (j > k)
        printf("0 is greater than -1");
  else
        printf("0 is less than or equal to -1");
  ```
- <u>implicit conversion rule</u>: if one of the operands is unsigned int, convert the other one to unsigned int, but when we convert -1 we get INT_MAX-1=big number

Arrays

- Syntax of defining an array:

  ```
          int a[10]; /* array of 10 integers */
  ```

- In C the index starts from 0, so the above definition allocated 10 integer variables:
  a[0], … , a[9]
- There is no allocated integer a[10] !!!!!
  However, the compiler will **not** warn you if you try to access it:

  ```
  int a[10];
  a[10] = 1;  /* crush and burn !! */
  ```

- To initialize an array you can use:

  ```
  a = {1,2,3,4,5,6,7,8,9,10};  /* a[0] == 1 */
  ```

- Multidimensional arrays are defined as follows:

  ```
  int a[10][20];
      /* a is array of 10 rows and 20 columns */
  a = {{1,1,… ,1}, {2,2,… ,2}, … {10,…,10}};
  ```

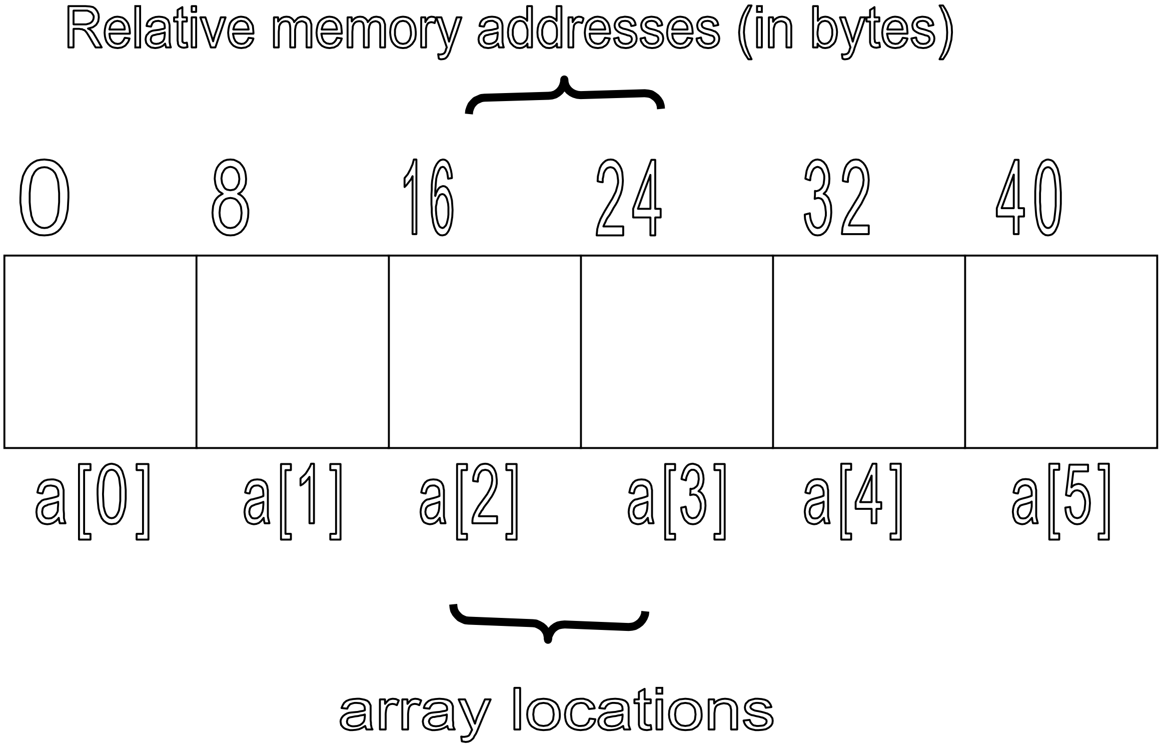
  or

  ```
  a = {1,1,… 1, 2,2,… ,2, … ,10, … ,10};
  ```

- more on multidimensional arrays later

Arrays in Memory

- For the following definition:
  double a[6];
  the compiler interprets the address of a[2] as:
  a[0] + 2*sizeof(double)



Each time an element is referenced, the compiler computes the address:
address = reference + index*sizeof(type)

Pointers - Introduction

- Pointers are special variables that store “the address” of another variable. Definition:
  ```
  <type> * <variable name>;
  ```

-
  ```
  float f1;
  float * pf1;  /* pf1 is a pointer to float */
  pf1 = &f1;
  ```
- & is the address operator:
  &<variable> gives the address of <variable>
  (no matter what <variable> is)
- * is the “value of” operator:
  ```
  float f1 = 1.0, f2 = 2.0;
  float * pf1= &f1;
  f2 = *pf1;   /* now the value of f2 is 1.0 */
  ```
- Pointers in memory (drawing)

Pointers and Arrays

- There is an important relation between pointers and arrays. By defining:
  int a[10];
  "a" by itself is of type (int *) - a pointer to int, and has the value &a[0] (the address of a[0]).
  So we can do the following:
  int *pa = a;
- Since pointers are just **numbers** (i.e. numeric memory addresses) we can do arithmetic operation on them:

```
int *pb = pa+1; /* now pb points to a[1] */
*pb = 1;           /* now a[1] = 1 */
*(pb + 2) = 3;   /* now a[3] = 3 */
```

Pointers - Examples

- Example: Swapping two arrays:

```
int a1[10];
int a2[10];
int *pa1 = a1;
int *pa2 = a2;
int *temp;
/* now pa1[3] = a1[3], for example, and
          *(pa1+3) = a1[3]                    */
temp = pa1;
pa1 = pa2;
pa2 = temp;
```

- Another (not elegant) way to implement array assignment pa1=pa2:

```
int j;
for (j=0;j<10;j++)
        *(pa1++) = *(pa2++);
```

Pointers to Pointers

- Since a pointer is just a **number** which represents an actual memory address of **some variable,** we can assign it the address of a variable which is another pointer.
  However, the syntax changes:

  ```
  int **ptr2ptr;
  int *ptr;
  int i = 1;

  ptr2ptr = &ptr;

  ptr = &i;           /* or: */
  *ptr2ptr = &i;     /* or: */
  *(*ptr2ptr) = i;
                     /* the latter causes *ptr = 1    */
  ```

- See memory drawing

- **We will study pointers later** !!!
  This was just an introduction !

Strings

- Constant string is represented by:
  char name[9] = “Aya Aner”; /* init */
  char name[] = “Aya Aner”; /* init */
  this is actually an array of characters
- Every constant string is terminated by the special null char ‘\0’, so here name is a character array of size 9, 8 letters (and space char) and the 9th character is ‘\0’.
- Only character arrays can be initialized like that.
- Special string manipulation library functions are available by including <string.h>
- more on strings later in this course
  (char *name; is a “special” string)

Arguments

- Until now we have seen examples of the main function calling other functions.
- Main can receive its own arguments, but in a preconditioned way:

  ```
  main(int argc, char **argv) {
          …
  }
  ```

- argc is a counter for the number of arguments given to main.

  argv is an array of strings - the actual arguments. argv[0] is the program name.

-
  ```
  % a.out 1 my_input
  ```

  argc is 3

  argv[0] = “a.out”

  argv[1] = “1”

  argv[2] = “my_input”

- The ability of main to take arguments is useful for passing parameters to a program

argc, argv example

- Computing the square root of an input number:

```
#include <stdio.h>
#include <stdlib.h>

main(int argc, char **argv)
{
    float inp;
    if (argc != 2) {
       printf(”Usage: a.out number \n”);
       exit(0);
    }
    /* atof converts an ascii string to a float
         see <stdlib.h> for atoi, atol etc..        */
    inp = atof (argv[1]);
    printf(“%f\n”,inp*inp);
}
```

Pass by Value vs. Pass by Pointer

-

```
void  test(int  val,  int *ptr)
{
        val = 1;
        *ptr = 1;
}
main()
{
        int i1 = 0, i2 = 0;
        /* i1 is passed by value */
        /* i2 is passed by pointer */
        test( i1, &i2);

        /* i1 is unchanged, i2 was set to 1 */
}
```

Summary of Lecture 2

- Relational and logic operations
- More C data types
- Introduction to arrays and pointers
- Function arguments, main() arguments