LECTURE 4

CME 182 – COMPUTER PROGRAMMING

E2E112 – Introduction to Computer Programming

Arrays

- Arrays are kind of data structure that can store a fixed-size sequential collection of elements of the same type.
- All arrays consist of contiguous memory location. The lowest address corresponds to the first element and the highest address to the last element.

Declaration of an array

```
type arrayName [ arraySize ];
```

Initializing of an array

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

Arrays - Example

```
#include <stdio.h>

int main () {

   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;

   /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ ) {
      n[ i ] = i + 100; /* set element at location i to i + 100 */
   }

   /* output each array element's value */
   for (j = 0; j < 10; j++ ) {
      printf("Element[%d] = %d\n", j, n[j] );
   }

   return 0;
}
```

Output

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109


--------------------------------
Process exited after 0.008071 seconds with
return value 0
Press any key to continue . . .
```

Multi-dimensional Arrays

	Column 0	Column 1	Column 2
Row 0	n[0][0]	n[0][1]	n[0][2]
Row 1	n[1][0]	n[1][1]	n[1][2]
Row 2	n[2][0]	n[2][1]	n[2][2]

Declaration of an multi-dimensional array

```
type name[size1][size2]...[sizeN];
```

Initializing of multi-dimensional array

```
int a[3][4] = {
  {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
  {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
  {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

Multi-Dimensional Arrays - EXAMPLE

```
#include <stdio.h>

int main () {

   /* an array with 5 rows and 2 columns*/
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
   int i, j;

   /* output each array element's value */
   for ( i = 0; i < 5; i++ ) {

      for ( j = 0; j < 2; j++ ) {
         printf("a[%d][%d] = %d\n", i,j, a[i][j] );
      }
   }

   return 0;
}
```

Output

```
a[0][0] = 0
a[0][1] = 0
a[1][0] = 1
a[1][1] = 2
a[2][0] = 2
a[2][1] = 4
a[3][0] = 3
a[3][1] = 6
a[4][0] = 4
a[4][1] = 8


--------------------------------
Process exited after 0.01367 seconds with return value 0
Press any key to continue . . .
```

Characters in arrays

```
#include "stdio.h"
main(int argc, char* argv[])
{
        int i;
        char c[6]={'a','b','c','d','e'};
        for (i=0;c[i]!='\0';i++)
        printf("\n %c",c[i]);
        printf("\n\n");
}
```

Output

```
a
b
c
d
e


--------------------------------
Process exited after 0.00796 seconds with return value 0
Press any key to continue . . .
```

- Characters can be defined in arrays

Pointers

- A pointer is a variable whose value is the address of the another variable.
- It can be direct address of the memory location.
- You must declare a pointer before using it to store any variable address.

Declaration of a pointer

```
type *var-name;
```

- The asterisk * used to declare a pointer is the same asterisk used for multiplication.
- The actual data type of the value of all pointers, whether integer, float, characters, etc..
- The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

Example of declaration

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

Printing out variable address - Example

```
#include <stdio.h>

int main () {

   int  var1;
   char var2[10];

   printf("Address of var1 variable: %x\n", &var1  );
   printf("Address of var2 variable: %x\n", &var2  );

   return 0;
}
```

Output

```
Address of var1 variable: 62fe4c
Address of var2 variable: 62fe40


--------------------------------
Process exited after 0.01153 seconds with
return value 0
Press any key to continue . . .
```

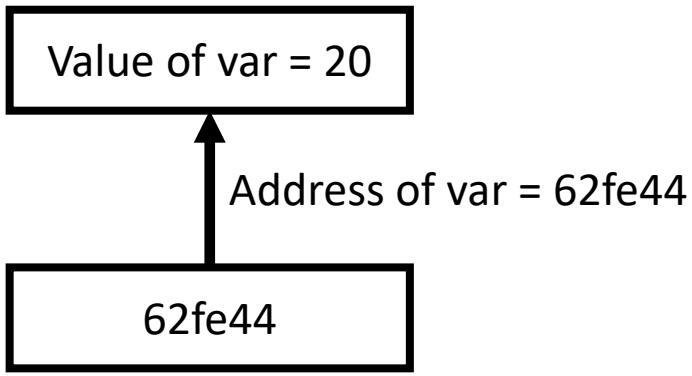
- This code was built up without using pointer.
- Memory address of any variable can be accessed using ampersand (&) operator.

Pointer - Example

```
#include <stdio.h>

int main () {

   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */

   ip = &var;  /* store address of var in pointer
variable*/

   printf("Address of var variable: %x\n", &var  );

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );

   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );

   return 0;
}
```

Output

```
Address of var variable: 62fe44
Address stored in ip variable: 62fe44
Value of *ip variable: 20

--------------------------------
Process exited after 0.01231 seconds with return value 0
Press any key to continue . . .
```



Pointer&Address - Example

```
#include "stdio.h"
int main(int argc, char* argv[])
{
		int x,y,z,m,n,i;
		int *p[5];
		x=200;
		y=300;
		z=7;
		m=56;
		n=65;
		p[0]=&x;
		p[1]=&y;
		p[2]=&z;
		p[3]=&m;
		p[4]=&n;
for(i=0;i<5;i++)
{
		printf("\n%p address - value =
%d",p[i],*p[i]);
}
		return 0;}
```

Output

```
000000000062FE48 address - value = 200
000000000062FE44 address - value = 300
000000000062FE40 address - value = 7
000000000062FE3C address - value = 56
000000000062FE38 address - value = 65
--------------------------------
Process exited after 0.01175 seconds with return value 0
Press any key to continue . . .
```

Null Pointers

- You do not have to assign an exact address for any variable
- This is done at the time of variable declaration.
- A pointer that is assigned NULL is called a null pointer.

```
#include <stdio.h>

int main () {

   int  *ptr = NULL;

   printf("The value of ptr is : %x\n", ptr  );

   return 0;
}
```

Output

```
The value of ptr is : 0


--------------------------------
Process exited after 0.01087 seconds with return value 0
Press any key to continue . . .
```

- In most of the operating systems, programs are not permitted to access memory at address 0 because that memory is reserved by the operating system.

Pointer Arithmetic

- A pointer is address which is a numerical value.
- Thus, we can arithmetic operations on a pointer just as you can on a numerical value.

```
#include <stdio.h>
const int MAX = 3;
int main () {
  int  var[] = {10, 100, 200};
  int  i, *ptr;
  /* let us have array address in pointer */
  ptr = var;
  for ( i = 0; i < MAX; i++) {
    printf("Address of var[%d] = %x\n", i,
ptr );
    printf("Value of var[%d] = %d\n", i, *ptr
);
    /* move to the next location */
    ptr++;
  }
  return 0;
}
```

Output

```
Address of var[0] = 62fe30
Value of var[0] = 10
Address of var[1] = 62fe34
Value of var[1] = 100
Address of var[2] = 62fe38
Value of var[2] = 200


--------------------------------
Process exited after 0.04011 seconds with return value 0
Press any key to continue . . .
```

- **Incrementing Pointer**

Decrementing a Pointer - Example

```
#include <stdio.h>
const int MAX = 3;
int main () {
   int  var[] = {10, 100, 200};
   int  i, *ptr;
   /* let us have array address in pointer */
   ptr = &var[MAX-1];
   for ( i = MAX; i > 0; i--) {
      printf("Address of var[%d] = %x\n", i-1, ptr );
      printf("Value of var[%d] = %d\n", i-1, *ptr );

      /* move to the previous location */
      ptr--;
   }
   return 0;
}
```

Output

```
Address of var[2] = 62fe38
Value of var[2] = 200
Address of var[1] = 62fe34
Value of var[1] = 100
Address of var[0] = 62fe30
Value of var[0] = 10


--------------------------------
Process exited after 0.01128 seconds with return
value 0
Press any key to continue . . .
```

Pointer Comparison

```
#include <stdio.h>
const int MAX = 3;
int main () {
   int  var[] = {10, 100, 200};
   int  i, *ptr;
   /* let us have address of the first element in pointer */
   ptr = var;
   i = 0;
   while ( ptr <= &var[MAX - 1] ) {
      printf("Address of var[%d] = %x\n", i, ptr );
      printf("Value of var[%d] = %d\n", i, *ptr );
      /* point to the previous location */
      ptr++;
      i++;
   }
   return 0;
}
```

Output

```
Address of var[0] = 62fe30
Value of var[0] = 10
Address of var[1] = 62fe34
Value of var[1] = 100
Address of var[2] = 62fe38
Value of var[2] = 200


--------------------------------
Process exited after 0.01098 seconds with return
value 0
Press any key to continue . . .
```

Pointer Array - Example

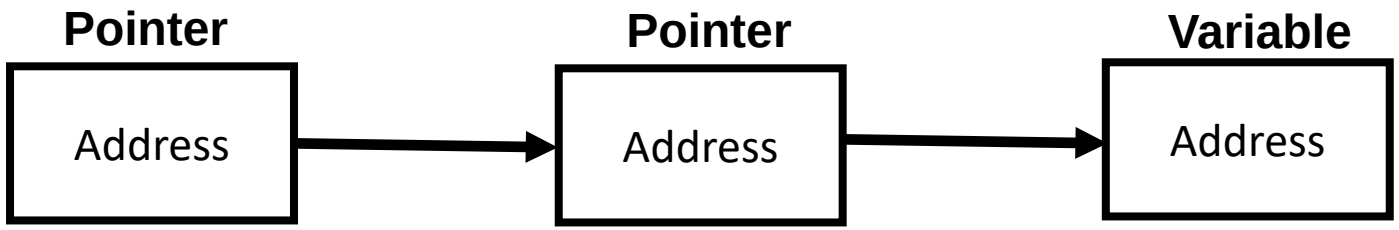
```
#include <stdio.h>

const int MAX = 4;

int main () {

   char *names[] = {
      "Daniel",
      "Jessica",
      "Heather",
      "David"
   };
   int i = 0;
   for ( i = 0; i < MAX; i++) {
      printf("Value of names[%d] = %s\n", i,
names[i] );
   }
   return 0;
}
```

Output

```
Value of names[0] = Daniel
Value of names[1] = Jessica
Value of names[2] = Heather
Value of names[3] = David


--------------------------------
Process exited after 0.0221 seconds with return value 0
Press any key to continue . . .
```

Pointer-to-Pointer

- A pointer to a pointer is a form of multiple indirection, or a chain of pointers.
- In normal use, pointer contains the address of the variable.
- When we define a pointer to pointer, the first pointer contains the address of the second pointer.
- The second pointer contains the address of the variable.



Declaration- pointer to pointer

```
int **var;
```

- A variable that is a pointer to pointer must be declared before it's using.
- Declaration of pointer-to-pointer can be done by placing an additional asterisk in front of its name.

Pointer-to-Pointer - Example

```
#include <stdio.h>
int main () {
   int  var;
   int  *ptr;
   int  **pptr;

   var = 3000;

   /* take the address of var */
   ptr = &var;

   /* take the address of ptr using address of operator & */
   pptr = &ptr;

   /* take the value using pptr */
   printf("Value of var = %d\n", var );
   printf("Value available at *ptr = %d\n", *ptr );
   printf("Value available at **pptr = %d\n", **pptr);
   return 0;
}
```

Output

```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000


--------------------------------
Process exited after 0.02009 seconds with return
value 0
Press any key to continue . . .
```

Strings

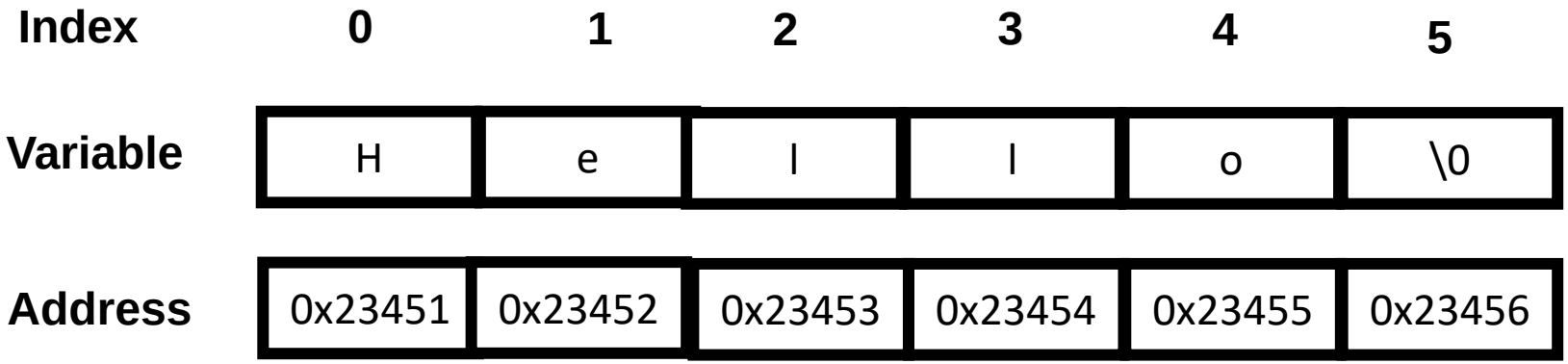
- Strings are actually one-dimensional array of characters terminated by a null character ‘\0’.
- Thus, a null-terminated string contains the characters that comprise the string followed by a null.

Declaration Examples

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'}
```

```
char greeting[] = "Hello";
```

These two declarations type give the same output



Strings - Example

```
#include <stdio.h>

int main () {

   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
   printf("Greeting message: %s\n", greeting );
   return 0;
}
```

Output

```
Greeting message: Hello


--------------------------------
Process exited after 0.01239 seconds with return value 0
Press any key to continue . . .
```

Strings – Several commands

- **strcpy(s1,s2)** copies string s2 into s1
- **strcat(s1,s2)** concatenates strings s2 onto the end of strings s1.
- **strlen(s1)** returns the length of string s1
- **strcmp(s1,s2)** returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
- **strchr(s1,ch)** returns a pointer to the first occurrence of character ch in string s1
- **strstr(s1,ch)** returns a pointer to the first occurrence of s2 in string s1.
- **strrev(s1)** reverses string s1

- Add the library **“string.h”** to use commands above.

String Commands - Example

```
#include <stdio.h>
#include <string.h>
int main () {
   char str1[12] = "Hello";
   char str2[12] = "World";
   char str3[12];
   int  len ;
   /* copy str1 into str3 */
   strcpy(str3, str1);
   printf("strcpy( str3, str1) :  %s\n", str3 );
   /* concatenates str1 and str2 */
   strcat( str1, str2);
   printf("strcat( str1, str2):   %s\n", str1 );
   /* total lenghth of str1 after concatenation */
   len = strlen(str1);
   printf("strlen(str1) :  %d\n", len );
   return 0;
}
```

Output

```
strcpy( str3, str1) :  Hello
strcat( str1, str2):   HelloWorld
strlen(str1) :  10


--------------------------------
Process exited after 0.009395 seconds with return
value 0
Press any key to continue . . .
```