More on C Programing

- Hong Liu
- HPC Consultant
- NICS

- Program control
- Advanced data types
- File handling

Program control

- **C provides two styles of flow control:**
  - Branching (**If** , **if-else** and **switch** structures)
  - Looping (**while loop, do loop and do...while loop**)

- **Branching** is deciding what actions to take. We call it ***branching*** because the program will choose to follow one branch or another.

- **Looping** is deciding how many times to take a certain action.

if statement

- **if** statements take the following form:

if (expression)
statement;
	if (expression)
	{
	Block of
	statements;
	}
		if (expression)
		{
		Block of
		statements;
		}else {
		Block of
		statements;
		}
			if (expression)
			{
			Block of statements;
			}else if(expression)
			{
			Block of statements;
			}else {
			Block of statements;
			}

```
#include <stdio.h>
main() {
    int cows = 6;
    if (cows > 1)
        printf("We have cows\n");
     if (cows > 10)
        printf("lots of them!\n");
        else
        printf("Executing else part...!\n");
    if (cows == 5 )
        printf("We have 5 cows\n");
    else if(cows == 6 )
        printf("We have 6 cows\n");
    }
```

switch statement:

- **The switch statement is much like a nested if .. else statement**

- switch (<variable>) {
- case this-value: Code to execute if <variable> == this-value
  - break;
- case that-value: Code to execute if <variable> == that-value
  - break; ...
- default: Code to execute if <variable> does not equal the value following any of the cases
  - break; }

```
#include <stdio.h>
main() {
    int Grade = 'A';
    switch( Grade ) {
        case 'A' : printf( "Excellent\n" );
            break;
        case 'B' : printf( "Good\n" );
            break;
        case 'C' : printf( "OK\n" );
            break;
        case 'D' : printf( "Mmmmm....\n" );
            break;
        default : printf( "What is your grade anyway?\n" );
            break;
        }
    }
```

Looping

- Loops provide a way to repeat commands and control how many times they are repeated.

- **while loop**
- **for loop**
- **do...while loop**

While Loop

```
while ( expression )
{
    Single statement
        or
    Block of statements;
    }
```

```
#include <stdio.h>
main()
{
    int i = 10;
     while ( i > 0 )
     {
            printf("Hello %d\n", i );
                i = i -1;
      }
}
```

For LOOP

```
for( expression1;
expression2; expression3)
{
    Single statement
            or
     Block of statements;
}
```

```
#include <stdio.h>
main() {
    int i; int j = 10;

    for( i = 0; i <= j; i ++ ) {
        printf("Hello %d\n", i );
    }
}
```

Do…While LOOP

```
do {
   Single statement
         or
   Block of statements;
}while(expression);
```

```
#include <stdio.h>
main() {
   int i = 10;
   do{
        printf("Hello %d\n", i );
         i = i -1;
        }
    while ( i > 0 );
}
```

break and continue statements

- C provides two commands to control how we loop:
- ***break*** -- exit form loop or switch.
- ***continue*** -- skip 1 iteration of loop.

```
#include <stdio.h>
main(){
    int i;    int j = 10;
        for( i = 0; i <= j; i ++ )    {
                if( i == 5 )      {        continue;      }
                printf("Hello %d\n", i );
    }
}
```

```
#include <stdio.h>
main(){
    int x ;
    for ( x=0 ; x<=100 ; x++) {
        if (x%2) { continue;}
        printf("%d\n" , x);
    }
}
```

- **a % b**
- produces the remainder when **a** is divided by **b**; and zero when there is no remainder.

Advanced Data Types

- Arrays
- Pointers
- string

```
main(){
  int al,a2,a3,a4,a5;
  scanf("%d %d %d %d %d",&a1,&a2,&a3,&a4,&a5);
  printf("%d %d %d %d %d",a5,a4,a3,a2,a1);
}
```

Arrays

- **type array[size]**

- **int a[5]**
- Declares an array called **a** with five elements. The first element is **a[0]** and the last **a[4]**.

- **int a[5] = {1,2,3,4,5};**
- Declares an integer array and initializes it so that **a[0]= 1**, **a[1] = 2** and so on

- If you omit the size of the array, an array just big enough to hold the initialization is created.
- **double** balance[ ] = {1000.0, 2.0, 3.4, 17.0, 50.0};

- Following is an example of array declaration, assignment.

```
#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; /*SetElement 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;
}
```

Pointers

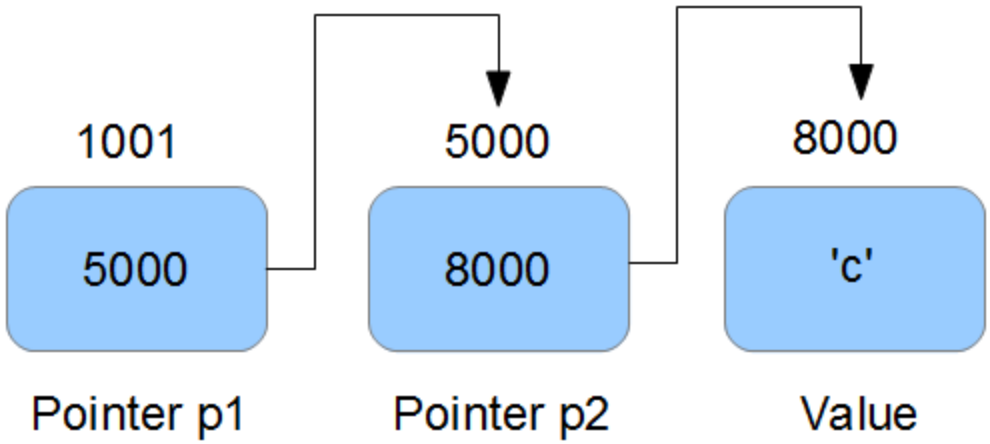
Every memory location has its address which can be accessed using ampersand (&) operator.

```
#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;
}
```

Pointers

- A ***pointer*** is a variable that stores location of memory.
- In more fundamental terms, a pointer stores the *address* of another variable .
- In more picturesque terms, a pointer points to another variable.

- Lets suppose we have a pointer ‘p1’ that points to another pointer ‘p2’ that points to a character ‘ch’. In memory, the three variables can be visualized as :



- So we can see that in memory, pointer p1 holds the address of pointer p2. Pointer p2 holds the address of character ‘ch’.

- A pointer has to be declared just like any other variable.

- **int *p;**
- **p** is the name of our variable. The '*' informs the compiler that we want a pointer variable, i.e. to store an address in memory. The **int** says that we intend to use our pointer variable to store the address of an integer. Such a pointer is said to "point to" an integer.

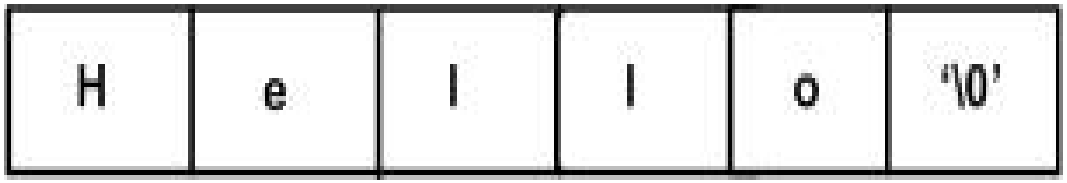
- **To declare a pointer add * in front of its name.**
- **To obtain the address of a variable use & in front of its name.**
- **To obtain the value of a variable use * in front of a pointer's name**

- **int *a , b, c;**
- Declares a pointer to an **int** and two **int** variables, not 3 pointers.
- **b = 10;**
- **a = &b;**
- The **&** operator returns the address of a variable. It stores the address of **b** in **a**. So **a** points to **b**
- **c = *a;**
- Stores the value in the variable pointed to by **a** in **c**. As **a** points to **b**, its value 10 is stored in **c**.
- In other words, this is a long way of writing
- **c = b;**
-

- #include <stdio.h>
- int main() {
  - int x; /* A normal integer*/
  - int *p; /* A pointer to an integer ("*p" is an integer, so p must be a pointer to an integer) */
  - p = &x; /* Read it, "assign the address of x to p */
  - Printf(“Please give x a value: ”);
  - scanf("%d", &x); /* Put a value in x*/
  - printf(“ Address is %d\n", p); /* Note the use of the * to get the value */
- }

string

- A ***string*** is just an array of characters which is terminated by a **null** character '\0'.
- **char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};**
- you can write the above statement as follows:
- **char greeting[] = "Hello";**
- Following is the memory presentation of above defined string



```
#include <stdio.h>
int main () {
   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
   printf("Greeting message: %s\n", greeting );
   return 0;
}
```

- C supports a wide range of functions that manipulate null-terminated strings:

- **strcpy(s1, s2);**
  Copies string s2 into string s1.
- **strcat(s1, s2);**
  Concatenates string s2 onto the end of string s1.
- **strlen(s1);**
  Returns the length of string s1.

```
#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;
}
```

File Handling

- C communicates with files using a new data type called a file pointer. This type is defined within stdio.h, and written as FILE *.
- A file pointer called f is declared in a statement like **FILE *f;**

- #include<stdio.h>

  int main(){
  - FILE *f;
    return 0;
- }

- open a file for reading or writing
- read/write the contents of a file
- close the file

- **Opening a file with fopen**
- The program must open a file before it can access it.
- #include<stdio.h>
  int main(){
  - FILE *f;
    f = fopen("test.txt","w");
    return 0;
- }

- "r" Open file for reading
- "w" Open file for writing
- "a" Open file for appending

- **Closing a file using fclose**
- The fclose command can be used to disconnect a file pointer from a file.
- **fclose(file);**

- If files are still open when a program exits, the system will close them for you. However it is usually better to close the files properly.

Input /Output with file pointers

- To work with text input and output, you use **fprintf** and **fscanf,** both of which are similar to their friends **printf** and **scanf** except that you must pass the FILE pointer as first argument.

- #include <stdio.h>
- main() {
  - FILE *fp;
  - fp = fopen(“ test.txt", "w");
  - fprintf(fp, "This is testing...\n");
  - fclose(fp);
- }

fscanf

- **Here is an example which will be used to read from a file:**
- ***fscanf*** stops reading after the first space character encounters
- #include <stdio.h>
- main() {
  - FILE *fp;
  - char buffer[20];
  - fp = fopen("test.txt", "r");
  - fscanf(fp, "%s", buffer);
  - printf("Read Buffer: %s\n", buffer);
  - fclose(fp);
- }

- **It is also possible to read (or write) a single character or a string to a file by**

- fputc fputc(int c, FILE *fp);
- fputs fputs(const char *s, FILE *fp);

  fputs("This is testing for fputs...\n", fp);
- fgetc fgetc (FILE *fp);
- fgets char *fgets(char *buf, int n, FILE *fp);

  fgets(buff, 255, (FILE)fp);*

- #include <stdio.h>
- main() {
  - FILE *fp;
  - fp = fopen("test.txt", "w");
  - fputs("This is testing for fputs...\n", fp);
  - fclose(fp);
- }

Questions