Exercise 18

Preprocessing: Obeys commands that begin with #, and removing comments, expanding macros and included files.

Compiling: The compiler turns the preprocessed code into assembly code for specific processor. The C code becomes a set of instructions that directly correspond to actions.

Assembling: The assembler will convert the assembly code into binary object code.

Linking: The linker merges all the object codes into a single executable file.

Exercise 19

In int main(), the value will return an integer by default.

In void main(), there is no value return to main.

In char main(), there will return a character.

Therefore, the return value depends on what type of main function we utilize.

Exercise 21

(a)140

(b)4

(c)24

Exercise 22

(a)0.000000

(b)0.666667

(c)0.000000

(d)3

(e)3

(f)3.000000

Exercise 27

1.Comparing the program with the logic chart. If the logic chart is correct and the program is wrong, it is easy to find out. For example, the formula is (a+b)*c and you type a+b*c.

2. If you really can't find the error, you can check by each segment by using conditional compilation which won't delete all the printf after debugging. Setting up several printf

function in different paragraph of the program. By observing the output values, we can know this segment whether is correct and keep narrow down to specific line.

3. If there is no problem be found in the program, we can check the logic chart for errors which means the algorithm might have error.

Exercise 28

```
#include <stdio.h>
#define MAX_YEARS 100

typedef struct {
  double inv0;
  double growth;
  int years;
  double invarray[MAX_YEARS+1];
} Investment;

int getUserInput(Investment *invp);
void calculateGrowth(Investment *invp);
void sendOutput(double *arr, int years);

int main(void) {

  Investment inv;

  while(getUserInput(&inv)) {
    inv.invarray[0] = inv.inv0;
    calculateGrowth(&inv);
    sendOutput(inv.invarray,
               inv.years);
  }
  return 0;
}

void calculateGrowth(Investment *invp) {

  int i;

  for (i = 1; i <= invp->years; i= i + 1) {

    invp->invarray[i] = invp->growth * invp->invarray[i-1];
  }
}

int getUserInput(Investment *invp) {
  int valid;

  printf("Enter investment, growth rate, number of yrs (up to %d): ",MAX_YEARS);
  scanf("%lf %lf %d", &(invp->inv0), &(invp->growth), &(invp->years));

  valid = (invp->inv0 > 0) && (invp->growth > 0) &&
    (invp->years > 0) && (invp->years <= MAX_YEARS);
  printf("Valid input?  %d\n",valid);

  if (!valid) {
    printf("Invalid input; exiting.\n");
  }
  return(valid);
}

void sendOutput(double *arr, int yrs) {

  int i;
  char outstring[100];

  printf("\nRESULTS:\n\n");
  for (i=0; i<=yrs; i++) {
    sprintf(outstring,"Year %3d:  %10.2f\n",i,arr[i]);
    printf("%s",outstring);
  }
  printf("\n");
}
```

Exercise 30

(a)3

(b)4

(c)2

(d)6

(e)error

(f)unknown

(g)2

Exercise 31

5, 3*1((5>1) true <<1) + (k=2) <<2 + (k==6) <<0

Exercise 32

(a)0xf2

(b)0x01

(c)0x0f

(d)0x0e

(e)0x01

(f)0x68

(g)0x00

Exercise 34

```c
#include <stdio.h>

int main (){
    int i ;
    printf("ASCII from 33 to 127\n");
    for(i=33; i<=127; i++){

        printf("%d : %c \t",i,i);

    }
    return 0;
}
```

Exercise 35

```c
#include <stdio.h>
#include <string.h>
#define MAXLENGTH 100       // max length of string input

void getString(char *str);  // helper prototypes
void printResult(char *str);
int greaterThan(char ch1, char ch2);
void swap(char *str, int index1, int index2);

int main(void) {
  int len;                  // length of the entered string
  char str[MAXLENGTH];      // input should be no longer than MAXLENGTH
  // here, any other variables you need
  int i,j;

  getString(str);
  len = strlen(str);        // get length of the string, from string.h
  // put nested loops here to put the string in sorted order

  for(j=0; j<len-1; j++){
      for(i=0; i<len-1; i++){
          if(greaterThan(str[i],str[i+1])){
              swap(str, i, i+1);
          }
      }
  }
  printResult(str);
  return(0);
}

// helper functions go here
void getString (char *str){
    printf ("Enter the string you would like to sort (up to %d): ", MAXLENGTH);
    scanf("%s",str);
}

void printResult (char *str){
    printf("Here is the sorted string: %s\n", str);
}

int greaterThan(char ch1, char ch2){
    return(ch1>ch2);
}

void swap(char *str, int index1, int index2){
    char s;
    s=str[index1];
    str[index1]=str[index2];
    str[index2]=s;
}
```