Step 13: struct, typedef, malloc, realloc

This assigment is done in class on (monday)04-14-14 AND (friday) 04-10-14.

This assignment is due for monday class 04-14-14 AND for the firday class 04-10-14.

Getting Started

Create a new directory called transistors:

```
cd cse251
mkdir transistors
cd transistors
```

Create this main program file transistors.c:

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

#include "transistors.h"

/*
 * Name :  < insert name here >
 * Description : Simple transistor description example program
*/



int main()
{
    printf("transistors!\n");


}
```

And create this empty header file transistors.h:

```
#ifndef TRANSISTORS_H
#define TRANSISTORS_H


#endif
```

Next, create a Makefile that builds this program from its component parts.

Be sure it compiles and runs. I'll not indicate specific places to compile very often. You will need to do so regularly.

1. A struct

We are going to create a program that lets us input common transistors. I need a way to describe a transistor. For that I will use a struct. Add this code to transistors.h:

```
/* Types */
#define NPN 1
#define PNP 2

/* Cases */
#define TO18 1
```

```
#define TO92A 2
#define TO92C 3
#define TO220 4
#define TO39 5

struct Transistor
{
    char number[10];
    int type;       /* NPN, PNP, etc. */
    int caseStyle;  /* TO18, etc. */
    double pmax;    /* Maximum power dissipation watts */
    double icmax;   /* Maximum collector current amps */
};
```

Of course, this goes between the include guards. Don't put it before the #ifndef or after the #endif.

Now add this variable to the function **main**:

```
int main()
{
    struct Transistor t1;

    printf("transistors!\n");


}
```

Be sure it compiles and runs okay. We're not doing anything yet, but be sure the syntax is right.

I have created a variable named **t1** of type **struct Transistor**. We can populate it like this:

```
    strcpy(t1.number, "2N3053");
    t1.type = NPN;
    t1.caseStyle = TO39;
    t1.pmax = 0.500;    /* 500mw */
    t1.icmax = 0.700;   /* 700ma */
```

Add this code to the end of the main function and be sure it compiles and runs.

Notice how I have to copy a string into the struct. I can't assign it like this: t1.number = "2N3053". I have to use strcpy, which copies the string.

DisplayTransistor

Now, create a new file io.c and add this function to it:

```
void DisplayTransistor(struct Transistor tran)
{
    printf("Number: %s\n", tran.number);
    switch(tran.type)
    {
    case NPN:
        printf("Type: NPN\n");
        break;

    case PNP:
        printf("Type: PNP\n");
        break;
    }

    printf("pMax: %.3f\n", tran.pmax);
    printf("icMax: %.3f\n", tran.icmax);
}
```

We are going to put our input/output functions into this .c file. You are going to have to #include <stdio.h> in io.c.

Put the declaration for this function into transistors.h after the Transistor struct has been created:

```
void DisplayTransistor(struct Transistor tran);
```

Add io.c to your Makefile and be sure it compiles and runs okay.

Now add this line of code to the end of the **main function** to display the transistor we just entered. Click here if you are unsure where this goes.

```
    DisplayTransistor(t1);
```

When you run this program you will see the transistor you entered displayed:

```
cbowen@ubuntu:~/cse251/transistors$ ./transistors
transistors!
Number: 2N3053
Type: NPN
pMax: 0.500
icMax: 0.700
```

InputTransistor

I would like to have code to enter a transistor. One of the great things about a struct is we can not only pass it to a function, we can return it from a function.

Add this function to io.c and put the declaration in transistors.h:

```
struct Transistor InputTransistor()
{
    struct Transistor t1;

    strcpy(t1.number, "2N3053");
    t1.type = NPN;
    t1.caseStyle = TO39;
    t1.pmax = 0.500;    /* 500mw */
    t1.icmax = 0.700;   /* 700ma */

    return t1;
}
```

Because this function uses strcpy, you are going to have to #include <string.h> in io.c.

Now change the main function to this:

```
int main()
{
    struct Transistor t1;

    printf("transistors!\n");

    t1 = InputTransistor();

    DisplayTransistor(t1);
}
```

Notice how we have moved the code to create a transistor to InputTransistor(). I put in dummy code initially to get the function written, compiling, and working. Now we'll fill in the code to input a transistor. In InputTransistor, replace this line:

```
    strcpy(t1.number, "2N3053");
```

with:

```
    scanf("%s", t1.number);
```

When you run the program you should now be able to input the transistor number. The rest is all still generated automatically.

Safe and Robust Input

A problem with this type of input is that it is not very safe or robust. Let's add a couple more input operations. Replace these two lines of code:

```
    t1.pmax = 0.500;    /* 500mw */
    t1.icmax = 0.700;   /* 700ma */
```

with this:

```
    printf("Input pMax: ");
    scanf("%lf", &t1.pmax);
    printf("Input icMax: ");
    scanf("%lf", &t1.icmax);
```

Now run the program. When it asks for pMax, enter "a". See what happens. Because it is looking for a floating point number, when it sees a, it stops. Then the next scanf also stops. Also, if you enter a transistor number greater than 9 characters long, you should cause the program to fail. Try running the program and type this for the number:

```
2N2222 0.5 0.7
```

You will notice it continues to get numbers from that first line.

Let's create a more robust and safe way to input a string. Add this function to io.c:

```
void InputString(char *str, int max)
{
    char buffer[100];

    /* Get a line of up to 100 characters */
    fgets(buffer, sizeof(buffer), stdin);

    /* Remove any stray newlines from the buffer */
    while (buffer[0] == '\n')
        fgets(buffer, sizeof(buffer), stdin);

    /* Remove any \n we may have input */
    if(strlen(buffer) > 0)
        buffer[strlen(buffer)-1] = '\0';

    /* Copy up to max characters to our string */
    strncpy(str, buffer, max);
    str[max-1] = '\0';
}
```

Read this carefully and be sure you understand how it works. Now, change this line in InputTransistor:

```
    scanf("%s", t1.number);
```

to:

```
    InputString(t1.number, sizeof(t1.number));
```

You should be able to enter any string you want for the transistor number with no problems.

Now we'll make a more robust function to input a double value:

```
double InputPositiveValue(char *prompt)
{
    char buffer[100];
    double value = 0;

    printf("%s", prompt);

    /* Get a line of up to 100 characters */
    fgets(buffer, sizeof(buffer), stdin);

    /* Remove any \n we may have input */
    if(strlen(buffer) > 0)
        buffer[strlen(buffer)-1] = '\0';

    sscanf(buffer, "%lf", &value);

    return value;
}
```

This function gets a line of text. It then uses the function sscanf to scan off the value. sscanf works like scanf, except is reads from a string instead of the standard input device.

Replace these two lines in InputTransistor:

```
    printf("Input pMax: ");
    scanf("%lf", &t1.pmax);
```

with:

```
    t1.pmax = InputPositiveValue("Input pMax: ");
```

Notice how we use a function to create a more robust and safe way to input the maximum power.

Of course, this does not ensure the value is positive. For that we need to add a loop to the InputPositiveValue function:

```
double InputPositiveValue(char *prompt)
{
    char buffer[100];
    double value = 0;

    do
    {
        printf("%s", prompt);

        /* Get a line of up to 100 characters */
        fgets(buffer, sizeof(buffer), stdin);

        /* Remove any \n we may have input */
        if(strlen(buffer) > 0)
            buffer[strlen(buffer)-1] = '\0';

        sscanf(buffer, "%lf", &value);
    } while(value <= 0);
    return value;
}
```

We now have a more robust way to enter values.

Tasks

At this point do the following tasks:

1. Change the input for t1.icmax to use InputPositiveValue.

2. Create a safe and robust function to input the transistor type. It should look something like this:

```
int InputTransistorType(char *prompt)
{
    /* ... */
}
```

And you should use it in InputTransistor like this:

```
    t1.type = InputTransistorType("Input type: ");
```

3. Add code to display the case style. The options can be found in transistors.h.

4. Create a safe and robust function to input the transistor case style.

typedef

Many programmers get tired of typing "struct". We can use typedef to avoid that problem. Change the definition of struct Transistor to:

```
typedef struct Transistor
{
    char number[10];
    int type;       /* NPN, PNP, etc. */
    int caseStyle;  /* TO18, etc. */
    double pmax;    /* Maximum power dissipation watts */
    double icmax;   /* Maximum collector current amps */
} Tran;
```

This should compile and run exactly the same as before. But, now we can refer to our transistor using the type Tran. Change the declaration of the variable t1 in the main function to this:

```
int main()
{
    Tran t1;

    printf("transistors!\n");

    t1 = InputTransistor();

    DisplayTransistor(t1);
}
```

This should work exactly the same, but should avoid you having to type "struct" all of the time. For example, we can change the function DisplayTransistor to this:

```
void DisplayTransistor(Tran tran)
```

Try changing it to that. Be sure to change both the function and the declaration in transistors.h.

Arrays of Structs

Change the main function to this:

```
int main()
{
    int i;
    Tran trans[3];

    printf("transistors!\n");

    for(i=0;  i<3;  i++)
    {
        trans[i] = InputTransistor();
    }

    printf("\nThe transistors:\n");
    DisplayTransistor(trans[0]);
}
```

When you run this you should be able to enter three transistors. If you are wanting some real values to use, here are some examples:

NPN transistors

Code	Type	**Case style**	**I_C max.**	**V_{CE} max.**	**h_{FE} min.**	**P_{tot} max.**	**Category (typical use)**
BC107	NPN	TO18	100mA	45V	110	300mW	Audio, low power
BC108	NPN	TO18	100mA	20V	110	300mW	General purpose, low power
BC108C	NPN	TO18	100mA	20V	420	600mW	General purpose, low power
BC109	NPN	TO18	200mA	20V	200	300mW	Audio (low noise), low power
BC182	NPN	TO92C	100mA	50V	100	350mW	General purpose, low power
BC182L	NPN	TO92A	100mA	50V	100	350mW	General purpose, low power
BC547B	NPN	TO92C	100mA	45V	200	500mW	Audio, low power
BC548B	NPN	TO92C	100mA	30V	220	500mW	General purpose, low power
BC549B	NPN	TO92C	100mA	30V	240	625mW	Audio (low noise), low power
2N3053	NPN	TO39	700mA	40V	50	500mW	General purpose, low power
BFY51	NPN	TO39	1A	30V	40	800mW	General purpose, medium power
BC639	NPN	TO92A	1A	80V	40	800mW	General purpose, medium power
TIP29A	NPN	TO220	1A	60V	40	30W	General purpose, high power
TIP31A	NPN	TO220	3A	60V	10	40W	General purpose, high power
TIP31C	NPN	TO220	3A	100V	10	40W	General purpose, high power
TIP41A	NPN	TO220	6A	60V	15	65W	General purpose, high power
2N3055	NPN	TO3	15A	60V	20	117W	General purpose, high power

PNP transistors

Code	**Structure**	**Case style**	**I_C max.**	**V_{CE} max.**	**h_{FE} min.**	**P_{tot} max.**	**Category (typical use)**
BC177	PNP	TO18	100mA	45V	125	300mW	Audio, low power
BC178	PNP	TO18	200mA	25V	120	600mW	General purpose, low power
BC179	PNP	TO18	200mA	20V	180	600mW	Audio (low noise), low power
BC477	PNP	TO18	150mA	80V	125	360mW	Audio, low power
BC478	PNP	TO18	150mA	40V	125	360mW	General purpose, low power
TIP32A	PNP	TO220	3A	60V	25	40W	General purpose, high power
TIP32C	PNP	TO220	3A	100V	10	40W	General purpose, high power

Add a loop to output the three transistors as well.

As you can see, arrays work the same for structs as they did for other values.

Variable Input, malloc(), and free()

A problem with this approach is that we may not know how may transistors to input. Suppose I want to be able to input a variable number of transistors. For this purpose, we are going to use *dynamic memory allocation*.

Change the main function to this version:

```
int main()
{
    int i;
    Tran *trans;
    int numTrans = 0;

    printf("transistors!\n");

    /* Allocate space for one transistor */
    trans = malloc(sizeof(Tran));
    numTrans = 1;

    /* Input the transistor */
    trans[0] = InputTransistor();

    /* Output the transistors */
    printf("\nThe transistors:\n");
    for(i=0;  i<numTrans;  i++)
    {
        DisplayTransistor(trans[i]);
    }

    /* Free the memory */
    free(trans);
}
```

You will have to #include <stdlib.h> to use malloc and free.

The function **malloc** allocates memory to your program. The size we are allocating is the size of one of our transistor structs. I'm also keeping track of the fact I have space for one transistor in numTrans. After allocation, the space works just like an array, in this case of size 1. So, this code works the same as if I had declared:

```
    Tran trans[1];
```

But, what is interesting here is that I can now increase that allocate any size I want. For example, if I need space for 100 transistors, I would allocate it like this:

```
    trans = malloc(sizeof(Tran) * 100);
    numTrans = 100;
```

At the end of the program you will see a function called **free**. The function malloc has asked the operating system for some memory. It has given it to you. It expects you to return it when you are through with it. That's what free does. Only call free on a pointer to memory you have allocated.

realloc()

Suppose I have 10 transistors, but I want to add one. When we use malloc, we can make the allocated memory larger using realloc. Add this code after the call to InputTransistor:

```
    /* Increase the space by one transistor */
    trans = realloc(trans, sizeof(Tran) * (numTrans + 1));
    numTrans++;

    trans[numTrans-1] = InputTransistor();
```

What Do We Have Now?

Now we have the ability to save records, such as the description of a student or an inventory item. And, because we now use malloc and realloc, we can store any number we want to.

Final Tasks

1. Be sure you completed the tasks mentioned earlier. You should be able to input and output all of the fields that describe our transistors.

2. Make sure your Makefile has a clean rule that will delete the file **transistors** (the executable) and the **.o** files (*.o)

3. Add code to the function main to input any number of transistors. After you enter each transistor, it should ask this question:

```
Would you like to enter another transistor (Y/N)?
```

If the users enters "Y" or "y" (either case), input one more transistor. When the user enters "N" or "n", display all of the transistors that have been entered. Be sure your input for the yes/no response is safe and robust.

When you are done, do the following:

A: Do a make clean to remove the executable and intermediate object files.

B: Type these commands:

```
cd ..
tar cvzf transistors.tar.gz transistors
```

This will create a file transistors.tar.gz. This is the file you will turn in.

Turn in transistors.tar.gz via http://secure.cse.msu.edu/handin/

CSE 251