CS 50: Introduction to Computer Science I
Harvard College
Fall 2007

Scribe Notes
Week 1 Friday
Anjuli Kannan

Announcements (0:00-13:00)

- Remember to section by 7 PM
- Problem set 1 is now online
- Virtual office hours will begin this week (see explanatory handout on website)
- Problem sets will now come in two varieties: standard edition (for most if not all of you) and hacker edition (for those with significant previous experience). You may choose which to do.

Variable Types and Standard Input Functions from the CS50 Library (13:00-23:00)

- The cs50 library contains some variable types and functions that are not by default included in C.
- We have included these to make life easier for you—for now. Eventually, you will be expected to write your programs without them, but for now they are available for your use.
- In order to use them, you must put this line at the top of your program: #include <cs50.h>
- You must also compile in the following manner: gcc –o hello3 hello3.c –lcs50
- Two types included in this library are string and bool
- A string is an ordered set of zero or more characters (i.e., a letter, word, or sentence)
- Here is an example of using strings to record text in our programs (hello2.c):

```
int
main(int argc, char * argv[])
{
    string name = "David";
    printf("hello, %s\n", name);
}
```

- The cs50 library also includes some standard input functions:
    - char GetChar() - will retrieve a character entered by the user
    - string GetString() – will retrieve a string entered by the user

```
int
main(int argc, char * argv[])
{
    string name;
    printf("State your name: ");
    name = GetString();
    printf("hello, %s\n", name);
}
```

    - float GetFloat() – will retrieve a floating point number entered by the user
    - int GetInt() – will retrieve an integer entered by the user

```
int
main(int argc, char * argv[])
{
    int x, y;

    /* ask user for input */
```

```
            printf("Give me an integer: ");
            x = GetInt();
            printf("Give me another integer: ");
            y = GetInt();

            /* do the math */
            printf("The sum of %d and %d is %d!\n", x, y, x + y);
        }
```

  - double GetDouble() – will retrieve a double entered by the user
- These functions are equipped with error-checking. For instance, if asked for an integer and you give a letter, it will ask you go retry. This saves you a lot of work!
- Home exercise: Make a program that asks for a temperature in Fahrenheit and tells the user the temperature in Celsius.

Conditions (23:00-33:00)

- If-else: do something in one case, do something else in all other cases

```
if (condition)
{
      /*do this*/
}
else
{
      /*do that*/
}
```

- Example: conditions1.c

```
int
main(int argc, char * argv[])
{
    int n;

    /* ask user for an integer */
    printf("I'd like an integer please: ");
    n = GetInt();

    /* analyze user's input (somewhat inaccurately) */
    if (n > 0)
        printf("You picked a positive number!\n");
    else
        printf("You picked a negative number!\n");
}
```

- What's the bug? Code doesn't deal with case in which n is 0
- If-else if-else: have several different cases

```
if (condition)
```

```
{
      /*do this*/
}
else if (condition)
{
      /*do that*/
}
else
{
      /*do this other thing*/
}
```

- Example: replace the if-else in conditions1.c with the following

```
if (n > 0)
        printf("You picked a positive number!\n");
    else if (n == 0)
        printf("You picked zero!\n");
    else
        printf("You picked a negative number!\n");
```

- Notice how we check for equality. We use == because = alone is the assignment operator. Remember that == will not work for strings!
- Notice we can exclude the squiggly braces if the contents of the if or else is only one line
- This is simply a matter of style. Placement of brackets, spacing, and indentation are all aspects of style. Every student has her own style, but just make sure your code is consistent and readable.

Boolean Expressions (33:00-38:00)

- if (condition1 || condition2) means if condition1 is true or condition2 is true
- if(condition1 && condition2) means if condition1 is true and condition2 is true
- Example: nonswitch.c

```
int
main(int argc, char * argv[])
{
    int n;

    /* ask user for an integer */
    printf("Give me an integer between 1 and 10: ");
    n = GetInt();

    /* judge user's input */
    if (n >= 1 && n <= 3)
        printf("You picked a small number.\n");
    else if (n >= 4 && n <= 6)
        printf("You picked a medium number.\n");
    else if (n >= 7 && n <= 10)
```

```
            printf("You picked a big number.\n");
        else
            printf("You picked an invalid number.\n");
    }
```

- Notice <= and >= syntax

Switch Statements (38:00-43:00)

- If you have an int or char that you want to check the value of and do different things depending on its value.
- You could do a whole bunch of if else ifs, but switch statements may be more readable and understandable.
- Example: switch1.c

```
int
main(int argc, char * argv[])
{
    int n;

    /* ask user for an integer */
    printf("Give me an integer between 1 and 10: ");
    n = GetInt();

    /* judge user's input */
    switch (n)
    {
        case 1:
        case 2:
        case 3:
            printf("You picked a small number.\n");
            break;

        case 4:
        case 5:
        case 6:
            printf("You picked a medium number.\n");
            break;

        case 7:
        case 8:
        case 9:
        case 10:
            printf("You picked a big number.\n");
            break;

        default:
           printf("You picked an invalid number.\n");
    }
}
```

- When the computer gets to a condition that is true, it executes whatever is under that case, as well as whatever is underneath all cases below it, until it reaches a break.
- If no condition is true, it will carry out the default case.
- It is very important to include a break if you want the computer to execute the instructions under one case but not under later cases. If there is no break written, the execution will "fall" down to later cases.
- It is not necessary to have a default case.
- You can also do a switch with chars, but you must use single quotes around them (while strings use double quotes)
- Example: switch2.c

```
int
main(int argc, char * argv[])
{
    char c;

    /* ask user for a char */
    printf("Pick a letter grade: ");
    c = GetChar();

    /* judge user's input */
    switch (c)
    {
        case 'A':
        case 'a':
            printf("You picked an excellent grade.\n");
            break;

        case 'B':
        case 'b':
            printf("You picked a good grade.\n");
            break;

        case 'C':
        case 'c':
            printf("You picked a fair grade.\n");
            break;

        case 'D':
        case 'd':
            printf("You picked a poor grade.\n");
            break;

        case 'E':
        case 'e':
            printf("You picked a failing grade.\n");
            break;

        default:
           printf("You picked an invalid grade.\n");
    }
```

```
        }
```

Loops (43:00-53:00)

- for loop:

  ```
  for (initializations; condition; updates)
  {
          /*do this again and again*/
  }
  ```

- Example: progress1.c

  ```
  for (i = 0; i <= 100; i++)
  {
      printf("Percent complete: %d%%\n", i);
      sleep(1);
  }
  ```

- Three components
  - i = 0. Initialization of the iterative variable.
  - i <=100. Condition that indicates when the contents of the loop should be done.
  - i++. Update that indicates what should be done after each run of the loop. i++ means i = i +1.
- while loop:
  while (condition)

  ```
  {
          /*do this again and again */
  }
  ```

- Example: progress3.c

  ```
  while (i <= 100)
  {
      printf("\rPercent complete: %d%%", i);
      fflush(stdout);
      sleep(1);
      i++;
  }
  ```

- Contains only a condition that indicates when the contents of the loop should be executed.
- Now you have to do updates and initialization yourself. Notice that these elements are still in our program, they are just not part of the while loop syntax.
- Remember to include an update step or you'll end up with an infinite loop!
- do-while loop:

```
do
{
        /*do this again and again */
}
while(condition)
```

- Use when you want to execute something at least once, and then start checking condition
- Useful when we are getting user input, because you will always ask once, but may ask multiple times if input is invalid.