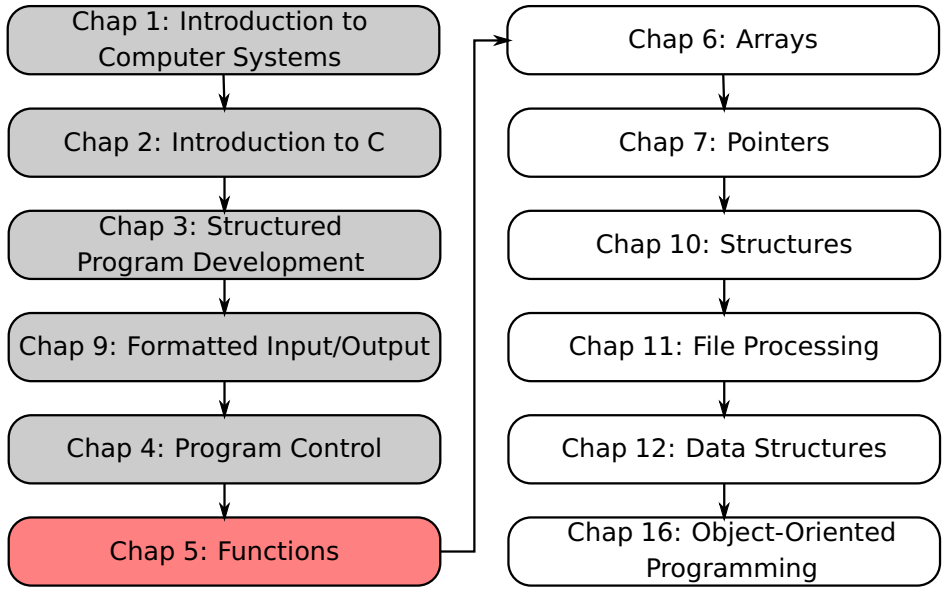
Computer Programming 143 – Lecture 12
Functions III

Electrical and Electronic Engineering Department
University of Stellenbosch

Corné van Daalen
Thinus Booysen
Willie Smit
Willem Jordaan

Module Overview



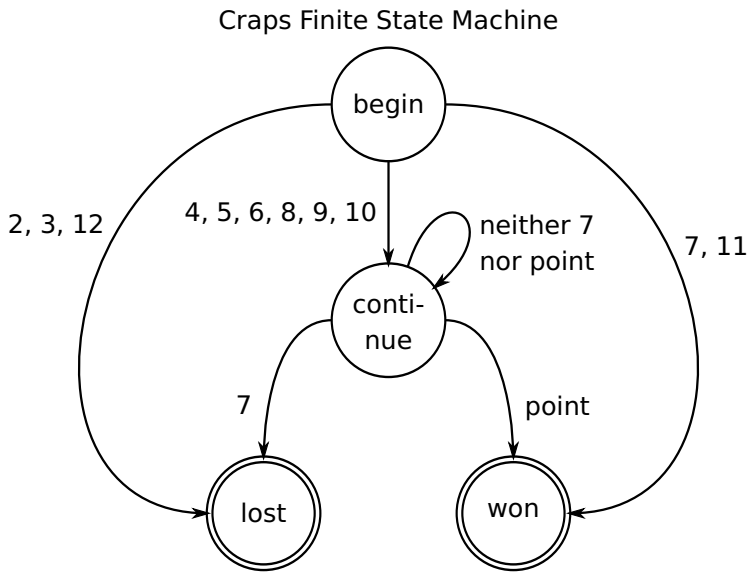
Lecture Overview

5.11 Example: A Game of Chance I

Rules of Craps

A player rolls two dice. Each die has six faces. These faces contain 1, 2, 3, 4, 5, and 6 spots. After the dice have come to rest, the sum of the spots on the two upward faces is calculated. If the sum is 7 or 11 on the first throw, the player wins. If the sum is 2, 3, or 12 on the first throw (called “craps”), the player loses (i.e. the “house” wins). If the sum is 4, 5, 6, 8, 9, or 10 on the first throw, then that sum becomes the player’s “point.” To win, you must continue rolling the dice until you “make your point.” The player loses by rolling a 7 before making the point.

5.11 Example: A Game of Chance II



5.11 Example: A Game of Chance III

C code

Refer to Fig. 5.14 in Deitel & Deitel

5.11 Example: A Game of Chance IV

Sample output 1

```
Player rolled 6 + 1 = 7
Player wins
```

Sample output 2

```
Player rolled 1 + 1 = 2
Player loses
```

Sample output 3

```
Player rolled 1 + 4 = 5
Point is 5
Player rolled 6 + 6 = 12
Player rolled 4 + 1 = 5
Player wins
```


5.11 Example: A Game of Chance V

Sample output 4

```
Player rolled 4 + 1 = 5
Point is 5
Player rolled 2 + 5 = 7
Player loses
```

5.11 Example: A Game of Chance VI

Enumerations

```
enum Status { CONTINUE, WON, LOST };
```

- Creates a user-defined enumeration data type
- Specifies a set of integer constants represented by identifiers

```
enum Status gameStatus;
```

- Declares variable **gameStatus** of enumeration type **Status**
- Variable **gameStatus** may take on values **CONTINUE**, **WON**, or **LOST**

```
gameStatus = WON;
```

- Variable **gameStatus** is assigned the value **WON**

```
if ( gameStatus == WON ) {...}
```

- Tests if variable **gameStatus** has the value **WON**
- Enumerations make code more readable and easily understood

5.12 Storage Classes I

Storage class specifiers

An identifier's storage class determines:

- **Storage duration** – how long a variable exists in memory
- **Scope** – where object can be referenced in program
- **Linkage** – specifies the files in which an identifier is known (more in Chapter 14)

5.12 Storage Classes II

Local variables

- Variables exist for certain section of program (block of code)
- Exist while block is active
- Destroyed when block is exited

Static storage

- Variables exist for entire program execution
- **static**: local variables defined in functions.
  - Keep value after function ends
  - Only known in their own function
- Global variables and functions
  - Known in any function

5.12 Storage Classes III

Example

```c
#include <stdio.h>

void accumulate(void); // function prototype

int main(void)
{
   int cntr, max_cntr;

   setbuf(stdout, 0);

   printf("Enter the maximum number of marks to be added: ");
   scanf("%d", &max_cntr);
   printf("Enter the %d marks to be added\n", max_cntr);
```

5.12 Storage Classes IV

Example (cont...)

```
   for (cntr = 0; cntr < max_cntr; cntr++)
   {
         accumulate(); // function call
   }
   printf("Done!!!!");

   return 0;
} // end function main
```


5.12 Storage Classes V

Example (cont...)

```
void accumulate(void)
{
   static int acc = 0; // static variable definition
   int num; // local variable definition
   printf("Enter a number: ");
   scanf("%d", &num);
   acc += num; // add number to current total
   printf("The current total is %d\n", acc);
}
```


5.13 Scope Rules I

Definition of scope

Portion of the program in which the identifier can be referenced

File scope

- Variable defined outside a function
- Known in all functions (in file)
- Used for global variables, function definitions, function prototypes

Block scope

- Variable declared inside a block ({...})
- Used for variables, function parameters (local variables of function)
- Outer blocks “hidden” from inner blocks if there is a variable with the same name in the inner block

5.13 Scope Rules II

Example

```
/* Example indicating scope of variable */
#include <stdio.h>

void useLocal(void); // function prototype
void useStaticLocal(void); // function prototype
void useGlobal(void); // function prototype

int x = 1; // global variable
```

5.13 Scope Rules III

Example (cont...)

```c
int main(void)
{
    int x = 5; // local variable to main

    printf("local x in scope of main is %d\n", x);
    useLocal(); // useLocal has local x
    useStaticLocal(); // useStaticLocal has static local
    useGlobal(); // useGlobal uses global x
    useLocal(); // useLocal has local x
    useStaticLocal(); // useStaticLocal has static local
    useGlobal(); // useGlobal uses global x
    printf("local x in scope of main is %d\n", x);

    return 0;
}
```


5.13 Scope Rules IV

Example (cont...)

```c
// useLocal reinitializes local variable x during each call
void useLocal(void)
{
    int x = 25; // initialized each time useLocal is called
    printf("\nlocal x in useLocal is %d ", x);
    printf("after entering useLocal\n");
    ++x;
    printf("local x in useLocal is %d ", x);
    printf("before exiting useLocal\n", x);
}
```


5.13 Scope Rules V

Example (cont...)

```c
/*
useStaticLocal initializes static local variable x only
the first time the function is called;
value of x is saved between calls to this function
*/
void useStaticLocal(void)
{
    static int x = 50; // initialized only once
    printf("\nlocal x in useStaticLocal is %d ", x);
    printf("after entering useStaticLocal\n");
    ++x;
    printf("local x in useStaticLocal is %d ", x);
    printf("before exiting useStaticLocal\n");
}
```


5.13 Scope Rules VI

Example (cont...)

```c
// function useGlobal modifies global variable x during each call
void useGlobal(void)
{
    printf("\nglobal x is %d after entering useGlobal\n", x);
    x *= 10;
    printf("global x is %d before exiting useGlobal\n", x);
}
```

Use globals sparingly! Preferable never

5.13 Scope Rules VII

Output:

```
local x in scope of main is 5

local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal

local x in useStaticLocal is 50 after entering useStaticLocal
local x in useStaticLocal is 51 before exiting useStaticLocal

global x is 1 after entering useGlobal
global x is 10 before exiting useGlobal
```


5.13 Scope Rules VIII

Output: (cont...)

```
local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal

local x in useStaticLocal is 51 after entering useStaticLocal
local x in useStaticLocal is 52 before exiting useStaticLocal

global x is 10 after entering useGlobal
global x is 100 before exiting useGlobal

local x in scope of main is 5
```


Perspective

Today

Functions III

- Example: a game of chance
- Storage class
- Scope rules

Next lecture

Functions IV

- Recursion

Homework

1. Study Sections 5.11-5.13 in Deitel & Deitel
2. Do Self Review Exercises 5.6 in Deitel & Deitel
3. Do Exercises 5.27 in Deitel & Deitel