Introduction to C Programming

Course: Introduction to Programming and Data Structure

Laltu Sardar

Institute for Advancing Intelligence (IAI),
TCG Centres for Research and Education in Science and Technology (TCG Crest)

tcg crest
Inventing Harmonious Future

August 11, 2022

tcg crest
Inventing Harmonious Future

The first C program

```
1 //FileName: hello.c
2 //Printing Hello world
3 #include <stdio.h>
4 main()
5 {
6   printf("hello, world\n");
7
8 }
```

1. Compilation: gcc hello.c /*a.out file will be generated*/
2. Run ./a.out /* default output file*/

Description

- stdio.h: standard input-output library
- printf: a library function
- input string
- \n - newline

Output your Name

```c
//FileName: namePrint.c
//Prints given name
#include <stdio.h>
main()
{
  char name[] = "Your Name"
  printf("hello, %s\n", name);
}
```

1. Compilation: gcc -g -Wall namePrint.c -o prog2.out
   - gcc → GNU Compiler Collection
   - gcc -g → generates debug info to be used by GDB debugger
   - -Wall → Show all warnings
2. Run: ./prog2.out
3. ".out" – not mandatory

tcg crest
Inventing Harmonious Future

Output your Name

Description

- char: variable type
- name: variable name
- %s : string output format specifier
- Commenting a line with /**/ and //: Not read by the compiler

tcg crest
Inventing Harmonious Future

Variables and Arithmetic Expressions

```c
/* filenme: FahToCel.c
   print Fahrenheit-Celsius table
   for fahr = 0, 20, ..., 300
*/
#include <stdio.h>
main()
{
  int fahr, celsius;      //variable Declaration
  int lower, upper, step;
  lower = 0; /* lower limit of temperature scale */ // variable assignment
  upper = 300; /* upper limit */
  step = 20; /* step size */
  fahr = lower;
  while (fahr <= upper) {     //while loop
    celsius = 5 * (fahr-32) / 9;
    printf("%d\t%d\n", fahr, celsius);
    fahr = fahr + step;
  }
}
```

Description

- Variable declaration
- Assign value to a variable
- Each variable must have a format specifier in printf

Building block of a Programming Language

1. Memory = space for calculations, rough work, etc.
2. Variables = names given to memory locations for convenience
3. Instructions = each step in the procedure

tcg crest
Inventing Harmonious Future

Naming rules of variables

Naming Rule of variables

1. Span: letters and digits
2. 1st character must be a letter
3. set of letters = { a, b, . . ., z, A, B, . . ., Z, _ }
4. The underscore "_" is count as letter
5. names are case sensitive.

Traditional C practice

- use lower case for variable names
- use all upper case for symbolic constants.

Variable-Name Examples

- abc_123 $\rightarrow$ valid
- _abc123 $\rightarrow$ valid
- _123 $\rightarrow$ valid
- _123abc $\rightarrow$ valid
- 123_abc $\rightarrow$ invalid

Tips

Variable name should be given in such a way that usage of the variable can be guessed easily from its name.

Should not be unnecessary long

tcg crest
Inventing Harmonious Future

Output format specifiers

Format Specifiers

- Format specifiers define the type of data to be printed on standard output.
- You need to use format specifiers whether you're printing formatted output with printf() or accepting input with scanf().

Some frequently used format specifiers

1. %d – decimal integer
2. %6d – decimal integer, at least 6 characters wide
3. %f – floating point
4. %6f – floating point, at least 6 characters wide
5. %.2f – floating point, 2 characters after decimal point
6. %6.2f – floating point, at least 6 wide and 2 after decimal point

Symbolic Constants

```
1  #include <stdio.h>
2  #define LOWER 0 /* lower limit of table */
3  #define UPPER 300 /* upper limit */
4  #define STEP 20 /* step size */
5  /* print Fahrenheit-Celsius table */
6  main()
7  {
8    int fahr;
9    for (fahr = LOWER; fahr <= UPPER; fahr = fahr + STEP)
10   printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
11 }
```

```
1  #define name replacement list
```

- symbolic constants are *string of characters*:
- They are not variables
- they do not appear in declarations
- In compiled files, they do not exists
- Conventionally written in upper case only

If

```
1 if (condition) {
2   // block of code to be executed
3   //if the condition is true
4 }
```

Example:

```
1 int a = 10;
2 int b = 2;
3 if (a > b) {
4   printf("a is greater than b");
5 }
```

tcg crest
Inventing Harmonious Future

If-Else

```
1  if (condition) {
2    // block of code to be executed
3    //if the condition is True
4  }else{
5    // block of code to be executed
6    //if the condition is False
7  }
```

```
1  int a = 10;
2  int b = 2;
3  if (a > b) {
4    printf("a is greater than b");
5  }else{
6    printf("a is less than b");
7  }
```

If-Else in a single line:

```
1    condition ? expression-true : expression-false
```

```
1    int a = 10, b = 2;
2    (a > b)? printf("a is greater than b"): printf("a is less than b");
```

Else-If

```
if (test expression1) {
  // statement(s)
}
else if(test expression2) {
  // statement(s)
}
else if (test expression3) {
  // statement(s)
}
.
.
else {
  // statement(s)
}
```

```
if (marks > 85) {
  printf("First Class with Distinction");
}
else if(marks > 60) {
  printf("First Class");
}
else if (marks>40) {
  print("Passed");
}
else {
  print("Failed");
}
```

Switch: Psudocode

```
1  switch (expression)
2  {
3    case constant1:
4    // statements
5    break;
6
7    case constant2:
8    // statements
9    break;
10   .
11   .
12   .
13   default:
14   // default statements
15 }
```

Switch: Example

```c
char operation;
double n1, n2;
printf("Enter an operator (+, -, *, /): ");
scanf("%c", &operation);
printf("Enter two operands: ");
scanf("%lf %lf",&n1, &n2);

switch(operation)
{
  case '+':
  printf("%.1lf + %.1lf = %.1lf",n1, n2, n1+n2);
  break;

  case '-':
  printf("%.1lf - %.1lf = %.1lf",n1, n2, n1-n2);
  break;

  case '*':
  printf("%.1lf * %.1lf = %.1lf",n1, n2, n1*n2);
  break;

  case '/':
  printf("%.1lf / %.1lf = %.1lf",n1, n2, n1/n2);
  break;

  // operator doesn't match any case constant +, -, *, /
  default:
    printf("Error! operator is not correct");
}
```

For and While

```
1 for ( init; condition; increment ) {
2   statement(s);
3 }
```

```
1 int i;
2
3 /* for loop execution */
4 for( i = 1; i < 10; i = i + 1 ){
5   printf("value of i: %d\n", i);
6 }
```

```
1 while(condition) {
2   statement(s);
3 }
```

```
1 int i = 1;
2
3 /* while loop execution */
4 while( i < 10 ) {
5   printf("value of i: %d\n", i);
6   i++;
7 }
```

for and while loop

Some frequently used format specifiers

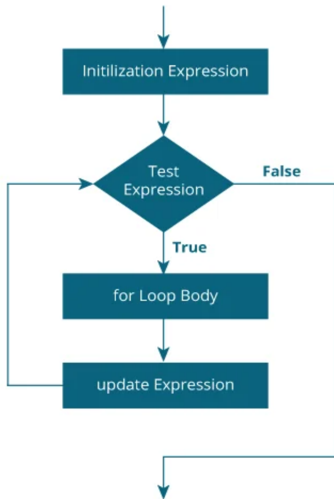
1. The for statement is a loop– a generalization of the while.
2. Three parts— separated by semicolons.
3. The first part– the initialization
4. The second part– Loop controller/ loop terminator
5. The third part– condition re-evaluation

'For' or 'while': which to use?

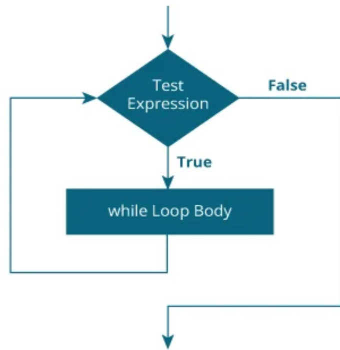
- whatever you want
- 'for' is more compact. It keeps the loop control statements together in one place

tcg crest
Inventing Harmonious Future

Flowchart of for



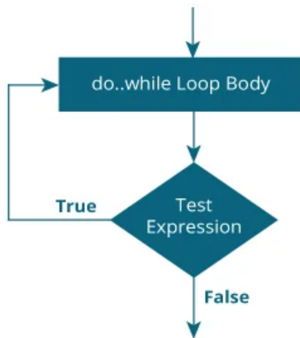
Flowchart of For loop



Flowchart of For while loop

tcg crest
Inventing Harmonious Future

Do-While



Flowchart of Do-while loop

Pseudocode:

```
1 do {
2   // the body of the loop
3 }
4 while (testExpression);
```

Example:

```
1     int i;
2
3     /* for loop execution */
4     i = 1;
5     do{
6       printf("value of i: %d\n", i);
7       i = i + 1 ;
8     } while(i < 10);
```

Question: What to use For or While or Do-While?

Break and Continue

`break` statement **terminates** a loop

```
1 for (int i = 1; i <= 40; i++) {
2   printf("value of i: %d\n", i);
3   if (i == 10) {
4     break;  // terminates the loop
5
6   }
7 }
```

`continue` **skips** a current iteration of a loop.

```
1 for (int i = 1; i <= 10; i++{
2   printf("value of i: %d\n", i);
3   if (i == 3) {
4     continue;
5   }
6 }
```

Functions

- So far, we have used printf, open, etc
- We know, we have to pass some parameter, it returns some values
- We don't have to know *how* it is defined
- We only have to know what is defined, and whats are its outputs

Why?

When we do same things with different values, we keep it in functions

tcg crest
Inventing Harmonious Future

Functions

```
1 return-type function-name(parameter declarations, if any)
2 {
3   declarations
4   statements
5 }
```

Functions

```
return-type function-name(parameter declarations, if any)
{
  declarations
  statements
}
```

```
#include <stdio.h>
int power(int m, int n); //declaration needed
/* test power function */
main()
{
  int i;
  for (i = 0; i < 10; ++i)
  printf("%d %d %d\n", i, power(2,i), power(-3,i));
  return 0;
}
/* power: raise base to n-th power; n >= 0 */
int power(int base, int n)
{
  int i, p;
  p = 1;
  for (i = 1; i <= n; ++i)
  p = p * base;
  return p;
}
```

TOP Secret to be an Expert in programming

Only Secret: Practice!

- Practice code/program writing
- Practice to solve daily eligible problems with coding
- Practice to take new coding challenges

tcg crest
Inventing Harmonious Future

topics tp be covered in some next class

- Character Arrays
- External Variables and Scope

tcg crest
Inventing Harmonious Future

Character array or String

```
1 char str[]    = "TCG_Crest";
2 char str[50]  = "TCG_Crest";
3 char str[]    = {'T','C','G','_','C','r','e','s','t','\0'};
4 char str[14]  = {'T','C','G','_','C','r','e','s','t','\0'};
```

str =

T	C	G	_	C	r	e	s	t	\0
0x12345	0x12346	0x12347	0x12348	0x12349	0x12350	0x12351	0x12352	0x12353	0x12356

- Accessing characters: str[0] = T, str[2] = G, etc
- '\0' is null character, terminating character

tcg crest
Inventing Harmonious Future