#01 C Review

242-207 FUNDAMENTAL PROGRAMMING I

DEPARTMENT OF COMPUTER ENGINEERING, PSU

v1.1

Contents

- C Program Components
- Variables
- Flow Controls
  - if/else
  - while, for
- Functions
- Arrays
- Strings
- Structures

C Program Components

```
#include<stdio.h>               //pre-processor
#define INC_BY
int inc(int x);                 //function prototype

int main(){
  int x;                        //variable declaration
  scanf("%d", &x);              //function calling
  printf("%d\n", inc(x));       //expression
  return 0;                     //return statement
}
int inc(int x){                 //function definition
  return x + INC_BY;
}
```

Variables

- Store values corresponding to its type
- Possible C Types: `char`, `int`, `long`, `float`, `double`

```
int x;              //declaration
... x + 1;          //read
x = ...;            //write
```

- Declaration Syntax: type variable-identifier;

Identifier

- To name variables or functions.
- Basic rules:
  - Not C keywords (e.g, `int`, `return`, `break`)
  - composed of letters (both uppercase and lowercase letters), digits and underscore '_' only.
  - first letter of identifier should be either a letter or an underscore.
- Good Practice:
  - choose meaningful name for an identifier

Control: if/else

- Conditional execution

```
if(x >= 0){
  printf("Positive");
}else{
  printf("Negative");
}
```

- Condition can be
  - Logical Expression using logical operators
    - ==, !=, >, <, >=, <=
  - True/False via integer expression
    - 0 => false, true otherwise.
- `switch/case` is also available.

Control: for, while

- execute until condition is false

```
while(x >= 0){
  ...
}

for(i = 0; i < 10; i++){
  ...
}
```

- `break` statement allows you to exit a loop from any point
- `continue` statement forces the next iteration of the loop to take place
  - skipping any code in between itself and the test condition
- `do/while` is also available.

Case study I

```
main(){
  int sum = 0;
  int x;
  do{
    printf("N: ");
    scanf("%d", &x);
    sum += x;
  }while(x != 0);
}
```

Find total of inputted numbers until user input 0.

Possible output:

```
N: <3>
N: <2>
N: <7>
N: <0>
= 12
```

Functions

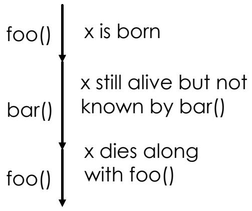
- Prototype: for compiler to know before hand about
  - name
  - parameter lists (only types of parameters are required)
  - return type
- Definition: define how function work
  - name
  - parameter lists (variable declaration separated by colon)
  - return type
  - function body { }
- Call statements: invoke function with matched parameters
  - interpreted as expression with type corresponding to function return type.

Variable Scope & Life Cycle

- Declare in a function, known only in the function and live as long as the function lives.

```
void foo(){
  int x;
  ...

  bar();

}
```

```
void bar(){
  ...

  .. I don't
know any x..

}
```



foo() x is born

bar() x still alive but not known by bar()

foo() x dies along with foo()

Recursions

- Factorial n (n!)
  - n! = 1, if n<=1
  - = n*(n-1)! Otherwise

```
long int factorial(long int n)
  {
      if(n<=1)
         return 1;
      else
         return (n * factorial(n-1));
   }
```

Case study II (The Fibonacci Series)

```
fib n = 0                      if n = 0
      = 1                      if n = 1
      = fib(n-1) + fib(n-2)   if n > 1

long int fib(long int n) {
  if(n==0 || n==1)
    return n;
  else
    return fib(n-1) + fib(n-2);
}
```

Arrays

- fixed-size sequential collection of elements of the same type
- Declaration
  - `int x[3];`
- Initialization (along with declaration)
  - `int x[] = {3, 1, 7};`
- Access
  - use subscription with integer expression as an index (starts with 0)
  - `x[2] = 3, y = x[1] + 2`

Case study III

```
main(){
  int d[NUM], i;
  for(i = 0; i < NUM; i++){
    printf("N%d:", i + 1);
    scanf("%d", &d[i]);
  }
  printf("=" + sum(d, NUM));
}

int sum(int a[], int n)
{
  int sum = 0, i;

  for(i = 0; i < n; ++i)
      sum += a[i];
  return sum;
}
```

Find total of five numbers inputted by user.

Possible output:

```
N1: <3>
N2: <2>
N3: <7>
N4: <0>
N5: <4>
= 16
```

Strings

- A sequence of characters which end with the `NULL` character `'\0'`

```
char color[] = "blue";

char color[] = {'b', 'l', 'u', 'e', '\0'};
```

- Sample code block to process string

```
... foo(char s[]){//or (char* s)
  int i;
  for(i = 0; s[i] != '\0'; i++)
    //access s[i] for each character
  ...
}
```

Case study III

```
main(){
  char s[128];
  scanf("%s", s);
  s_inv(s);
  printf("=" + s);
}
void s_inv(char s[]) {
  int i;
  for(i = 0; s[i] != '\0'; i++)
    s[i] = c_inv(s[i]);
}
char c_inv(char c){
  if(c >= 'a' && c < 'z') return (c – 'a') + 'A';
  else if(c >= 'A' && c < 'Z') return (c –'A') + 'a';
  return c;
}
```

Invert character case of given string. Do not change the character if it is not English letter.

Possible output:

```
S: <Hello World>
= hELLO wORLD
```

**DO NOT use ASCII number in your code.*

String manipulations

- Copy a string:

  `char *strcpy (char *dest, char *src);`

- Concatenating strings:

  `char *strcat(char *dest, const char *src);`

- Comparing strings:

  `int strcmp(char *s1, char *s2)`

  - return < 0 if s1 is less than s2, > 0 otherwise.
  - return 0 if s1 is equal to s2

- String Length:

  `int strlen(char *s);`

- many more...

Structures

- A collection of items of different types
- Type Declaration

```
struct student {
    char firstName[20];
    char lastName[20];
    float gpa;
};
typedef struct student Student;
```

- Variable Declaration

  `Student s = {"john", "doe", 2.55};`

- Access

```
s.gpa = 2.58;
strcpy(s.firstname, "John");
```