PROGRAMMING WITH C - CONTROL FLOW & FUNCTIONS

UNIVERSITY
OF PETROLEUM
& ENERGY STUDIES

| Aug 2014|

CONDITIONAL STATEMENTS

- Which provide the condition to execute a block.
  - *If* statement
  - *If else* statement
  - *Else if* statement or nested *if else* statement
  - *Switch* statement
  - *Break* statement
  - *Continue* statement
  - *Goto* statement

If statement

- The *if* conditional statement is used to provide the selection control structure that execute a set of statements based on a condition.
- *if (condition)*

```
{
    set of statements;
}
Ex:
if (marks < 30)
{
    printf("Sorry… You fail\n");
```

The if else statement

- ***If else*** conditional statement is used to select any one set of statements between two sets of statements.
- ***if (condition)***

{

set of statements;

}

else

{

another set of statements;

}

next statement;

The if else statement

- Ex:

```
if (marks < 30)
{
        printf(" Sorry … You fail");
        printf("Good luck in the supplementary");
}
else
{
        printf("Congrats .. You pass");
}
```

The else if statement or nested if else

- ***else if*** statement is used for the multi way decision based on several conditions.
- if (condition 1)

```
{
        set of statements(1);
}
else if (condition 2)
{
        set of statements(2);
}
……
else if (condition n)
{
        set of statement – n;
}
else
{
        set of statements – x;
}
```

The else if statement or nested if else

- if (marks >= 75)

```
    printf("passed with grade A \n");
else if (marks >= 60)
    printf("passed with grade B \n");
else if (marks >= 45)
    printf("passed with grade C \n");
else
    printf("fail");
```

The switch Statement

- The switch statement is another form of the multi way decision statements.

```
switch(var)
{
        case value1:
                        statements;
                        break;
        case value2:
                        statements;
                        break;
        default:
                        statements;
}
```

The switch Statement

- switch(c)

  {

  case 'A': capa++;

  case 'a': lettera++;

  default : total++;

  }

Break, Continue, Goto statements

- **The continue statement provides a convenient way to force an immediate jump to the loop control statement. The break statement terminates the execution of the loop.**

- Goto statement for unstructured jumps. It is very rarely recommended.

```
statements;
goto flagname;
statements;
flagname:
statements;
```

Loop Construct…

- A *loop* causes a section of a program to be *repeated* a certain number of times.
- The repetition continues while the *condition* set for it *remains true*.
- When the condition becomes *false*, the loop *ends* and the control is passed to the statement following the loop.
  - for loop
  - while loop
  - do .. while loop

for loop

- The for loop is usually used when the number of iterations is predetermined.

```
for (expr1;expr2;expr3)
{
        s1;
        s2 ;
    }
```

1. expr1 is executed only once before looping.
2. expr2 is a Boolean expression. If not given, it is assumed to be true.
3. If expr2 is false, the loop is terminated.
4. After execution of the repeat section, expr3 is executed
5. The expressions expr1, expr2 and expr3 are all optional.

for loop

- #include <stdio.h>

```
main()
{
    int i,n = 5;

    for(i = 0; i < n; i = i+1)
        {
       printf("the numbers are %d \n",i);
        }
 }
```

while loop

- **while** statement lets you repeat a statement until a specified expression becomes false.

while (*expression* **)** *statement*

```
#include<stdio.h>
main()
{
    int i = 0;
    while (i<5)
        {
        printf(" the value of i is %d\n", i);
        i = i + 1;
        }
}
```

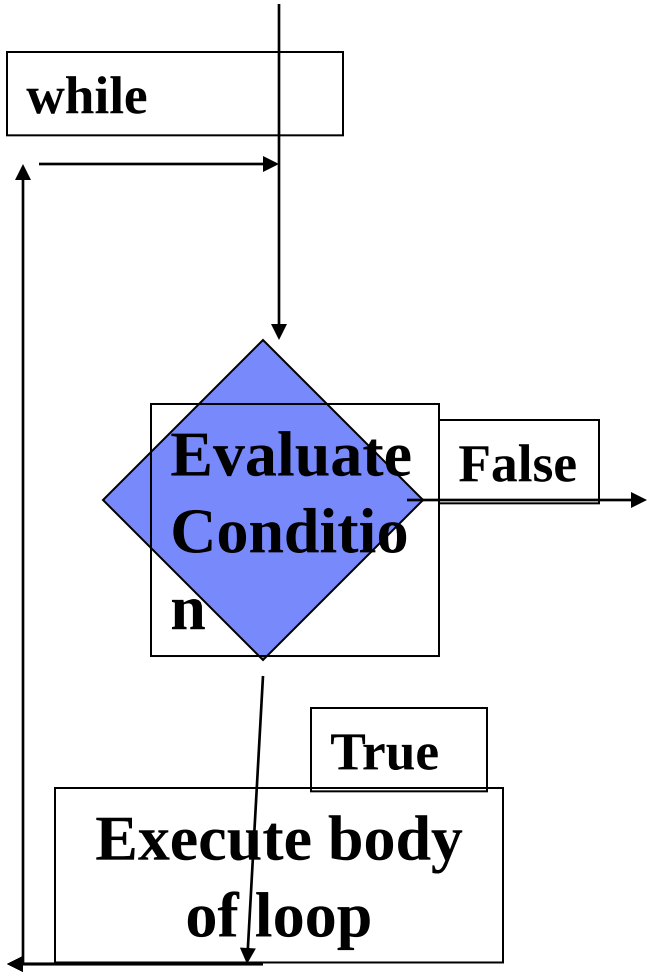
do while loop

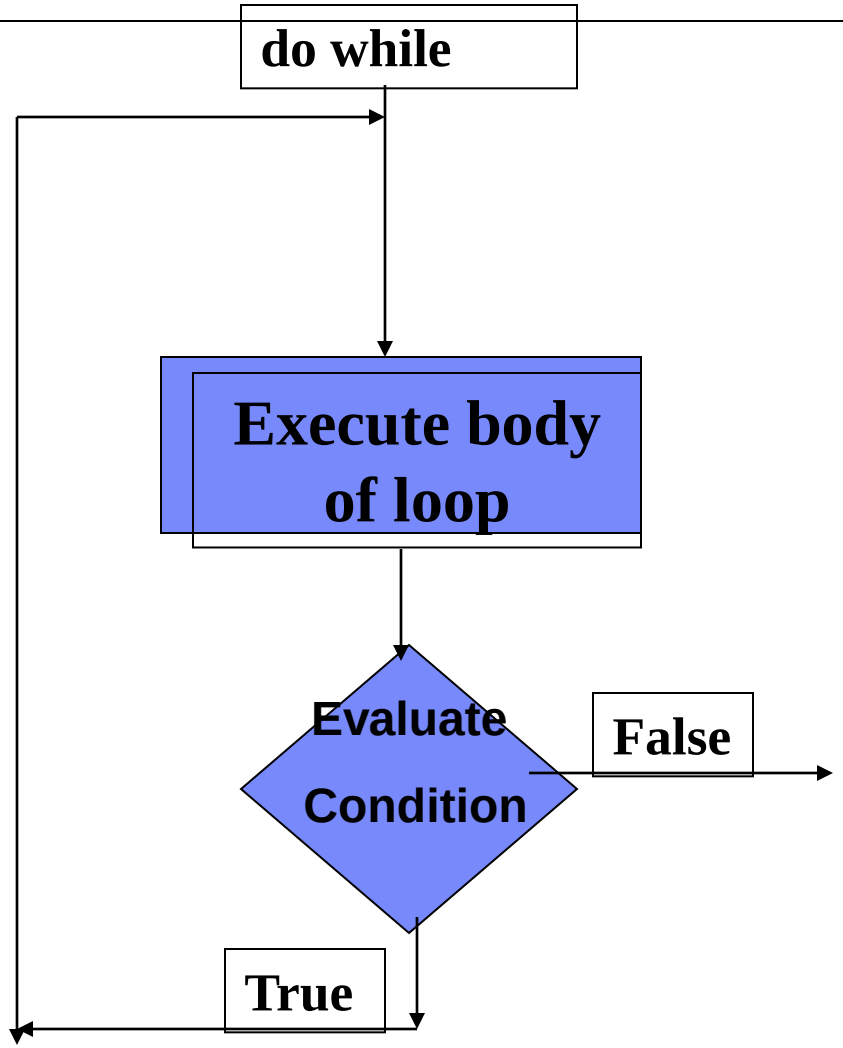
- do-while statement lets you repeat a statement or compound statement until a specified expression becomes false.

do *statement* **while (** *expression* **) ;**

```
do
{
y =  x ;
x--;
} while ( x > 0 );
```

Comparison…





Knowing prefix & postfix…

- *Eg: Prefix= ++ivar;*

int var1=10;
var1=++var1;

This will be translated as:

int var1=10;
var1= var1 + 1;
var2=var1;

Result is both *var1* & *var2* are set to *11.*

- ***Eg: Postfix= ivar++;***

This will be translated as:

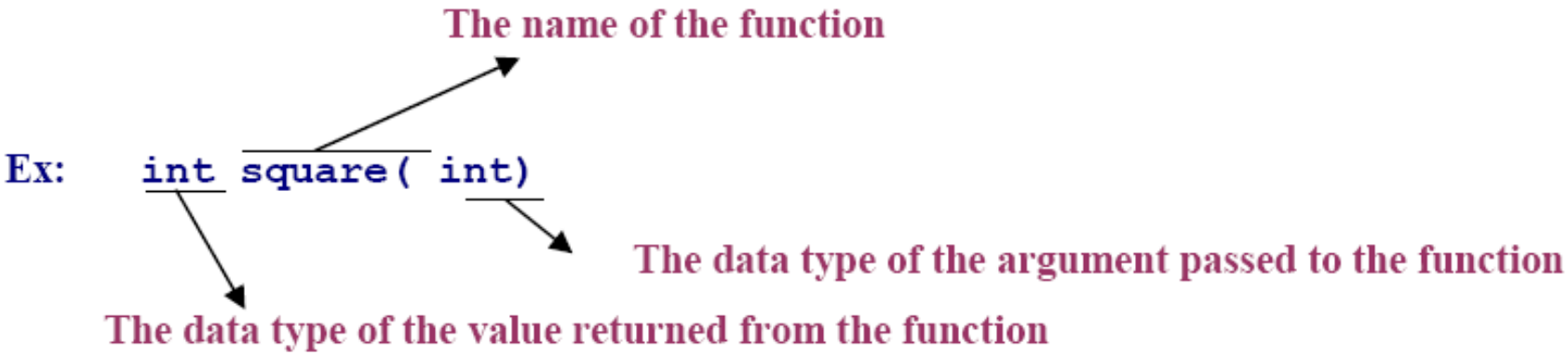
var1 11
var2 10.

Functions

- A function receives zero or more parameters, performs a specific task, and returns zero or one value.
- A function is invoked by its name and parameters.
  - No two functions have the same name in your C program.
  - The communication between the function and invoker is through the parameters and the return value.
- A function is independent:
  - It is “completely” self-contained.
  - It can be called at any places of your code and can be ported to another program.
- Functions make programs reusable and readable.

Function Prototype

- Function prototypes are always declared at the beginning of the program indicating the **name of the function**, the **data type of its arguments** which is passed to the function and the **data type of the returned value** from the function.



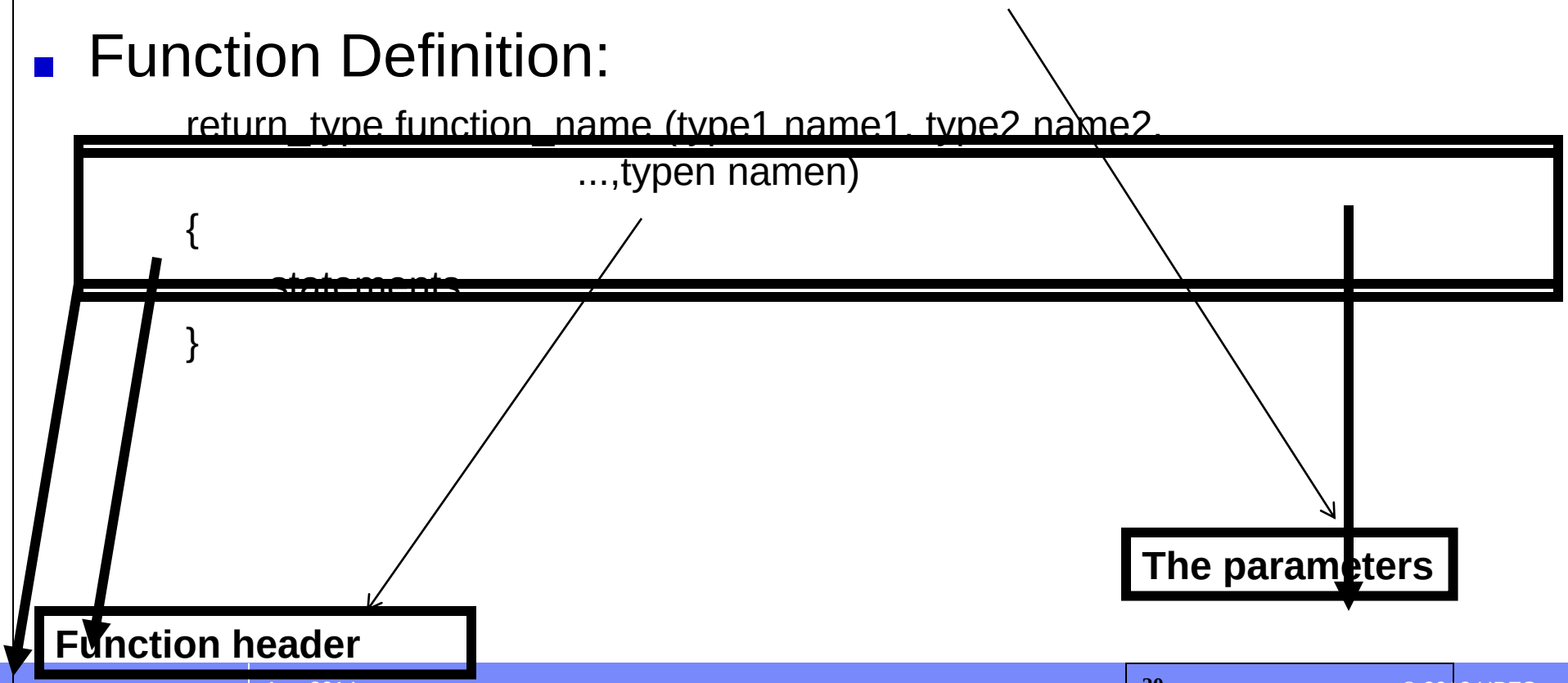
Syntax

- Function Prototype:

```
return_type  function_name (type1  name1, type2  name2,
                            ..., typen  namen);
```

- Function Definition:

```
return_type function_name (type1 name1, type2 name2,
                           ...,typen namen)
{
        statements
}
```



Some Examples

- **Function Prototype Examples**

```
double squared (double number);
void print_report (int);
int get _menu_choice (void);
```

- **Function Definition Examples**

```
double squared (double number)
{
   return (number * number);
}

void print_report (int report_number)
{
    if (report_nmber == 1)
         printf("Printer Report 1");
      else
         printf("Not printing Report 1");
}
```

Passing Arguments

- Function call:

func1 (a, b, c);

- Function header

int func1 (int x, int y, int z)

  - Each argument can be any valid C expression that has a value:
  - For example:

x = func1(x+1,func1(2,3,4),5);

- Parameters **x y z** are initialized by the value of **a b c**
- Type conversions may occur if types do not match.

Parameters are Passed by Value

- **All parameters are passed by value!!**
  - This means they are basically local variables initialized to the values that the function is called with.
  - They can be modified as you wish but these modifications will not be seen in the calling routine!

```
#include<stdio.h>
int twice(int x)
{
    x=x+x;
    return x;
}
int main()
{
    int x=10,y;
    y=twice(x);
    printf("%d,%d\n",x,y);
}
```

Returning a Value

- To return a value from a C function you must explicitly return it with a return statement.
- Syntax:

return <expression>;

  - The expression can be any valid C expression that resolves to the type defined in the function header.
  - Type conversion may occur if type does not match.
  - Multiple return statements can be used within a single function (eg: inside an “if-then-else” statement…)

Local Variables

- Local Variables

```
int func1 (int y)
{
    int a, b = 10;
    float rate;
    double cost = 12.55;
    .......
}
```

- Those variables declared “within” the function are considered “local variables”.
- They can only be used inside the function they were declared in and not elsewhere.

A Simple Example

```
#include <stdio.h>
int x=1;   /* global variable /
void demo(void);
int main() {
  int y=2;  /* local variable to main */
  printf ("\nBefore calling demo(), x = %d and y = %d.",x,y);
  demo();
  printf ("\nAfter calling demo(), x = %d and y = %d.\n",x,y);
  return 0;
}
void demo () {
  int x = 88, y =99;  /* local variables to demo */
  printf ("\nWithin demo(), x = %d and y = %d.",x,y);
}
```

Library Functions

C standard library provides a rich collection of functions for performing I/O operations, mathematical calculations, string manipulation operations etc.

For example, sqrt(x) is a function to calculate the square root of a double number provided by the C standard library and included in the <math.h> header file.

TYPES OF FUNCTION CALLS

- **Call by Value:**

When a function is called by an argument/parameter which is not a pointer the copy of ***the argument*** is passed to the function. Therefore a possible change on the copy does not change the original value of the argument.

Example

- Write a program to calculate and print the area and the perimeter of a circle. Note that theradius is to be entered by the user. (Use ***Call by value*** approach)
- **#include<stdio.h> /*The function calls are Call by Value*/**
- **#define pi 3.14**
- **float area(float);**
- **float perimeter(float);**
- **int main( )**
- **{**
- **float r, a, p;**
- **printf("Enter the radius\n");**
- **scanf("%f",&r);**
- **a = area(r);**
- **p = perimeter(a);**
- **printf("The area = %.2f, \n The Perimeter = %.2f", a, p);**
- **return 0;**
- **}**

Example

```
float area(float x)
{
return pi*x*x;
}
float perimeter(float y)
{
return 2.0*pi*y;
}
```

TYPES OF FUNCTION CALLS

- **Call by Reference:**

When a function is called by an argument/parameter which is a pointer (address of the argument) the copy of ***the address of the argument*** is passed to the function. Therefore a possible change on the data at the referenced address change the original value of the argument.

Example

- Write a program to calculate and print the area and the perimeter of a circle. Note that the radius is to be entered by the user. (Use ***Call by reference*** approach)

- **#include<stdio.h> /*The function calls is Call by Reference*/**
- **#define pi 3.14**
- **void area_perimeter(float, float *, float *);**
- **int main( )**
- **{**
- **float r, a, p;**
- **printf("Enter the radius\n");**
- **scanf("%f",&r);**
- **area_perimeter(r,&a,&p);**
- **printf("The area = %.2f, \n The Perimeter = %.2f", a, p);**
- **return 0;**
- **}**

Example

- **void area_perimeter(float x, float *aptr, float *pptr);**
- **{**
- ***aptr = pi*x*x;**
- ***pptr = 2.0*pi*x;**
- **}**

Library Functions

Ex:

:

```
double a=9.9, b;
b = sqrt(a);
printf("The square root of %f is %f", a, b);
```

:

Other functions such as exp(x) (exponential function e^x) and pow(x,y) (x^y) ... can be used as they are needed. Note that each program in C has a function called main which is used as the root function of calling other library functions.

End