یونٹ نمبر5

فنکشنز

س: آپ مسئلہ حل کرنے کے طریقہ کار اور تقسیم کرو اور فتح کرو کی تکنیک کے بارے میں کیا جانتے ہیں؟

مشکل اور پیچیدہ مسائل کو حل کرنے کا عمل مسئلے کا حل کہلاتا ہے۔ کسی مسئلہ کو حل کرنے کے لیے ایک منظم حکمت عملی اختیار کرنا ہوتی ہے۔ تقسیم کرو اور فتح کرو کی حکمت عملی میں ایک پیچیدہ مسئلے کو چھوٹے چھوٹے مسائل میں تقسیم کیا جاتا ہے اور ان کا حل نکال کر ایک پیچیدہ مسئلے کا حل نکال لیا جاتا ہے۔ اس طرح ہمارے لیے ایک وقت میں ایک چھوٹے مسئلے پر توجہ مرکوز کرنا آسان ہو جاتا ہے بجائے اس کے کہ ہم ہر وقت پورے مسئلے کے بارے میں سوچیں۔

س. فنکشنز کیا ہے؟ فنکشنز کی اقسام کی وضاحت کریں۔

فنکشن سٹیٹمنٹس کا ایک بلاک ہے جو ایک خاص کام انجام دیتا ہے جیسے printf ایک فنکشن ہے جو کمپیوٹر اسکرین پر کسی بھی چیز کو ظاہر کرنے کے لیے استعمال ہوتا ہے۔ scanf ایک اور فنکشن ہے جو یوزر سے ان پٹ لینے کے لیے استعمال ہوتا ہے۔ ہر پروگرام میں ایک مین فنکشن ہوتا ہے جو یوزر کے کاموں کو سر انجام دیتا ہے۔ اسی طرح ہم دوسرے فنکشنز لکھ سکتے ہیں اور انہیں کئی بار استعمال کر سکتے ہیں۔

فنکشنز کی اقسام

بنیادی طور پر فنکشنز کی دو قسم ہیں۔

1) بلٹ ان فنکشنز (Built-in Function) 2) یوزرڈیفائنڈ فنکشنز (User-defined Function)

1) بلٹ ان فنکشنز (Built-in Function)

جن فنکشنز کی پہلے سے سی پروگرامنگ لینگویج میں وضاحت کی جاتی ہے انہیں بلٹ ان فنکشن کہا جاتا ہے۔ ان فنکشنز کو لائبریری فنکشن بھی کہا جاتا ہے۔ یہ فنکشنز لینگویج کے ایک حصے کے طور پر دستیاب ہیں۔ یہ فنکشنز ریڈی میڈ پروگرام ہیں۔ یہ فنکشنز عام طور پر ریاضی کے حساب کتاب, سٹرنگ آپریشن, ان پٹ / آؤٹ پٹ آپریشن وغیرہ انجام دیتے ہیں۔ مثال کے طور پر printf اور scanf بلٹ ان فنکشنز ہیں-

2) یوزرڈیفائنڈ فنکشنز (User–defined Function)

وہ فنکشنز جو پروگرامر بناتا ہے انہیں یوزرڈیفائنڈ فنکشنز کہتے ہیں۔ یہ فنکشنز یوزر کی ضرورت کے مطابق بنائے جاتے ہیں۔

س. فنکشنز استعمال کرنے کے فوائد بیان کریں۔

فنکشنز ہمیں درج ذیل فوائد فراہم کرتے ہیں۔

1) دوبارہ استعمال (Reusability)

فنکشنز کے ذریعے ہم کوڈ کو دوبارہ استعمال کر سکتے ہیں۔ اس کا مطلب یہ ہے کہ جب بھی ہمیں ایک فنکشن کی فنکشنیلیٹی (functionality) کی ضرورت ہو ہم اس فنکشن کو کال کر سکتے ہیں۔ ہمیں بار بار سٹیٹمنٹس کا ایک ہی مجموعہ لکھنے کی ضرورت نہیں ہے۔

پھر ہمیں اس مسئلے کو دور کرنے کے لیے پورے پروگرام کو چیک کرنے کی ضرورت نہیں ہے۔ ہمیں صرف ایک فنکشن پر توجہ دینے کی ضرورت ہے۔

3) مسئلے کی پیچیدگی سے نمٹنا (Handling the Complexity of problem)

اگر ہم پورے پروگرام کو ایک پروسیجر کے طور پر لکھیں تو پروگرام کی دیکھ بھال کرنا مشکل ہو جاتا ہے۔ فنکشنز پروگرام کو چھوٹے چھوٹے حصوں میں تقسیم کرتے ہیں اور اس طرح مسئلے کی پیچیدگی کم ہو جاتی ہے۔

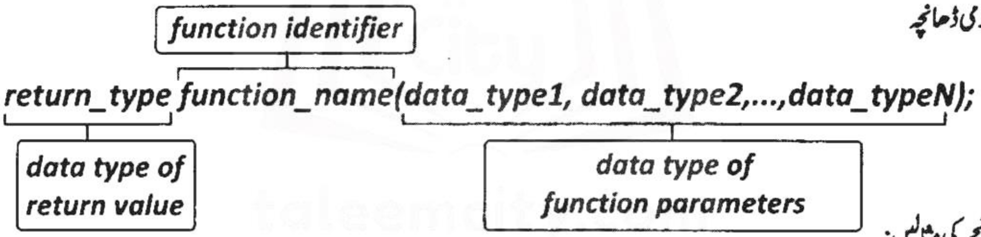
4) پڑھنے کی اہلیت (Readability)

پروگرام کو کئی فنکشنز میں تقسیم کرنا پروگرام کی پڑھنے کی اہلیت کو بہتر بناتا ہے۔

س. فنکشن کا سگنیچر اور اس کا عمومی ڈھانچہ مثالوں کے ساتھ بیان کریں۔

فنکشن سٹیٹمنٹس کا ایک بلاک ہے جو کچھ ان پٹ حاصل کرتا ہے اور کچھ آؤٹ پٹ فراہم کرتا ہے۔ فنکشن کی ان پٹس کو فنکشن کے پیرامیٹرز کہا جاتا ہے اور فنکشن کی آؤٹ پٹ کو اس کی ریٹرن ویلیو کہا جاتا ہے۔ ایک فنکشن میں ایک سے زیادہ پیرامیٹرز ہو سکتے ہیں لیکن یہ ایک سے زیادہ قیمت ریٹرن نہیں کر سکتے۔ فنکشن کے سگنیچر (signature) فنکشن کی ان پٹ اور آؤٹ پٹ کی وضاحت کے لیے استعمال ہوتے ہیں۔

فنکشن کا عمومی ڈھانچہ



فنکشن سگنیچر کی مثالیں:

فنکشن کی تفصیل	فنکشن سگنیچر
ایک فنکشن جو ایک انٹیجر کو بطور ان پٹ لیتا ہے اور اس کا مربع ریٹرن کرے گا۔	int square (int);
ایک فنکشن جو مستطیل کی لمبائی اور چوڑائی کو بطور ان پٹ لیتا ہے اور مستطیل کا احاطہ ریٹرن کرے گا۔	float perimeter (float, float);
ایک فنکشن جو تین انٹیجر کو بطور ان پٹ لیتا ہے اور ان میں سب سے بڑی قیمت ریٹرن کرے گا۔	int largest (int, int, int);
ایک فنکشن جو دائرے کے رداس کو بطور ان پٹ لیتا ہے اور دائرے کا رقبہ ریٹرن کرے گا۔	float area (float);
ایک فنکشن جو ایک کریکٹر کو بطور ان پٹ لیتا ہے اور اگر حرف علت (vowel) ہے تو 1 ریٹرن کرے گا ورنہ 0 ریٹرن کرے گا۔	int isVowel (char);

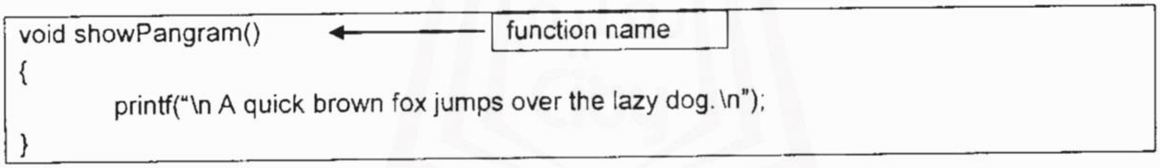
س: فنکشن ڈیفینیشن کے حصوں کی تعریف کریں؟

ایک فنکشن کے سگنیچر سے یہ نہیں پتہ چلتا کہ وہ کس کام کے لیے ہے۔ اس کے لیے فنکشن ڈیفینیشن ہوتی ہے۔ ایک فنکشن ڈیفینیشن کا ڈھانچہ کچھ اس طرح سے ہے۔

```
return_type function_name (data_type_var1, data_type _var2, ...., data_type_varN)
{
        Body of the function
}
```

فنکشن کی باڈی ان سٹیٹمنٹس کا سیٹ ہوتا ہے جنہیں چلا کر فنکشن ایک خاص کام سرانجام دیتا ہے۔ فنکشن سگنیچر کے فورا بعد قوسین { } کے اندر بند سٹیٹمنٹس کا مجموعہ فنکشن کی باڈی بناتا ہے۔

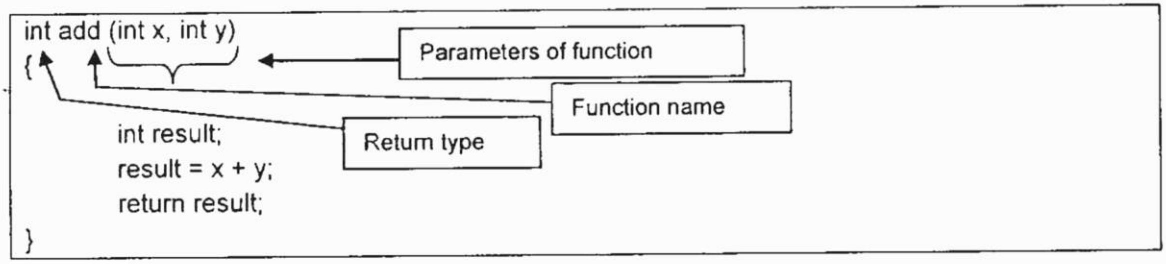
مثال: درج ذیل مثال ایک فنکشن showpangram( ) کی وضاحت کرتی ہے جو کوئی ان پٹ نہیں لیتا اور کچھ بھی ریٹرن نہیں کرتا بس کمپیوٹر سکرین پر "A Quick Brown Fox Jumps over the Lazy Dog " دکھاتا ہے۔



```
void showPangram()
{
        printf("\n A quick brown fox jumps over the lazy dog. \n");
}
```

جیسا کہ مذکورہ فنکشن کچھ ریٹرن نہیں کرتا اس طرح فنکشن کی ریٹرن ٹائپ void ہے۔

مثال: فنکشن کی مثال جو بطور ان پٹ دو انٹیجر لیتی ہے اور دونوں انٹیجر کا مجموعہ ریٹرن کرے گی۔



```
int add (int x, int y)
{
        int result;
        result = x + y;
        return result;
}
```

فنکشن کے اندر return ایک مطلوبہ لفظ ہے جو کالنگ (calling) فنکشن میں ویلیو ریٹرن کرنے کے لیے استعمال ہوتا ہے۔ ایک فنکشن ایک سے زیادہ قیمتیں ریٹرن نہیں کر سکتا۔ مثال کے طور پر مندرجہ ذیل سٹیٹمنٹ ایک کمپائلر ایرر کا نتیجہ ہے۔

```
Return (4 ,5);
```

ایک فنکشن میں ایک سے زیادہ ریٹرن سٹیٹمنٹس ہو سکتی ہیں لیکن جیسے ہی پہلی ریٹرن سٹیٹمنٹ چلتی ہے فنکشن کال ختم ہو جاتی ہے اور فنکشن کی باڈی میں مزید سٹیٹمنٹس پر عملدرآمد نہیں ہوتا ہے۔

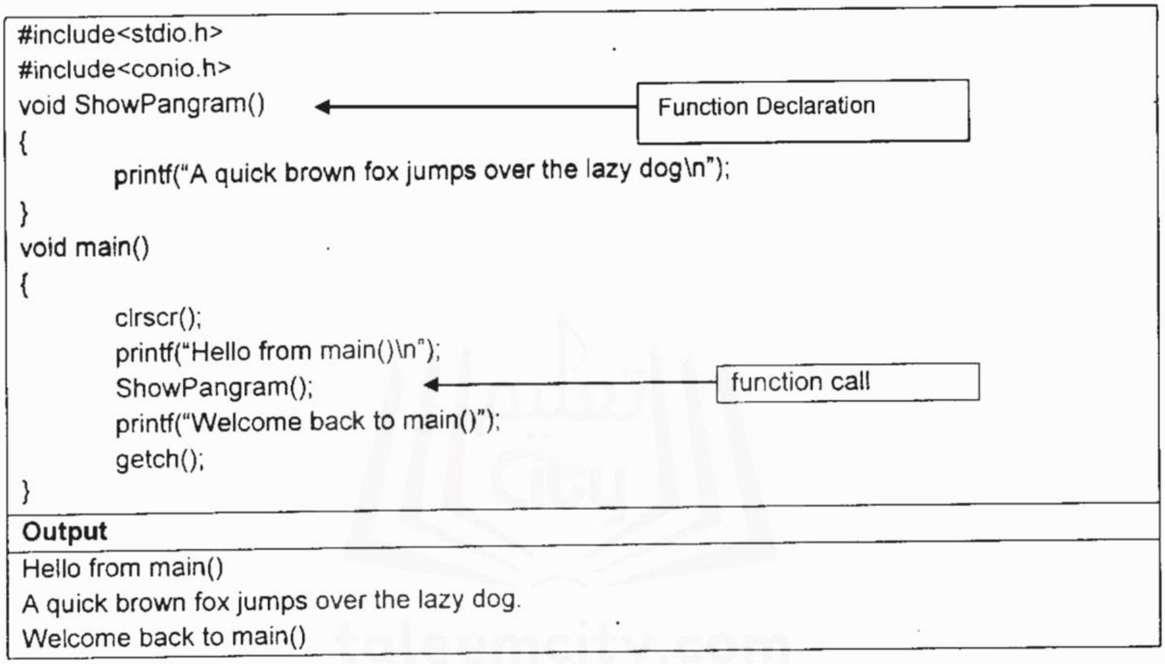
س: فنکشن کال کرنے کے لیے کون سا عمومی ڈھانچہ استعمال کیا جاتا ہے؟

ہمیں کسی فنکشن کو کال کرنے کی ضرورت ہے تاکہ وہ پروگرام کو سونپا گیا کام انجام دے۔

عمومی ڈھانچہ (General Structure)

مندرجہ ذیل عمومی ڈھانچہ ہے جو فنکشن کال کرنے کے لیے استعمال ہوتا ہے۔

function_name(value1, value2, ... , valueN);



```
#include<stdio.h>
#include<conio.h>
void ShowPangram()
{
        printf("A quick brown fox jumps over the lazy dog\n");
}
void main()
{
        clrscr();
        printf("Hello from main()\n");
        ShowPangram();
        printf("Welcome back to main()");
        getch();
}
```

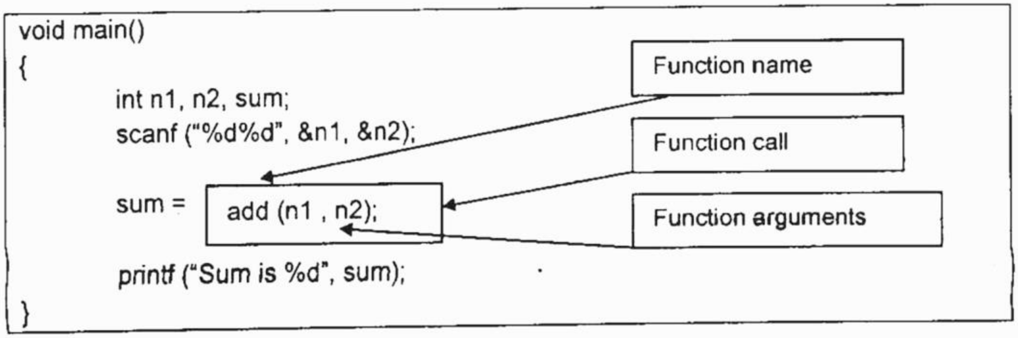
Output

```
Hello from main()
A quick brown fox jumps over the lazy dog.
Welcome back to main()
```

ہم دیکھ سکتے ہیں کہ پروگرام main() فنکشن سے چلنا شروع کرتا ہے۔ جب یہ ایک فنکشن کال (مستطیل کے اندر) آتی ہے تو یہ کنٹرول اس فنکشن میں منتقل ہو جاتا ہے۔ فنکشن کی سٹیٹمنٹس پر عمل کرنے کے بعد کنٹرول ریٹرن کالنگ فنکشن میں منتقل ہو جاتا ہے۔

س: ایک پروگرام لکھیں جو دو نمبران پٹ لے اور ان کا مجموعہ دکھائے۔

سٹیٹمنٹ ;sum = add (n1, n2) کوڈ میں فنکشن "add" کو کال کرتی ہے۔



```
void main()
{
        int n1, n2, sum;
        scanf ("%d%d", &n1, &n2);
        sum =  add (n1 , n2);
        printf ("Sum is %d", sum);
}
```

فنکشن کال میں n1 اور n2 فنکشن add() کے آرگیومینٹس ہیں۔ فنکشن add() سے ریٹرن آنے والے نتائج کو ذخیرہ کرنے کے لیے متغیر sum ڈیکلئیر کیا گیا ہے۔

فنکشن کو بطور آرگیومینٹ پاس کیے گئے ویری ایبلز میں کوئی تبدیلی نہیں آتی۔ فنکشن ویری ایبلز کی ایک کاپی بناتا ہے اور تمام ترامیم صرف اس کاپی میں کی جاتی ہیں۔ درج ذیل مثال میں جب n1 اور n2 پاس کیے جاتے ہیں تو فنکشن ان ویری ایبلز کی کاپیاں بناتا ہے۔ ویری ایبل n1 کی کاپی x ہے اور ویری ایبل n2 کی کاپی y ہے۔

سوال: آرگیومینٹ اور پیرامیٹرز میں کیا فرق ہے؟ ایک مثال دیں۔

جو قیمتیں فنکشن کو پاس کی جاتی ہیں وہ آرگیومینٹس کہلاتی ہے جبکہ فنکشن ڈیفینیشن میں جن ویری ایبلز میں یہ قیمتیں جاتی ہیں انہیں فنکشن کے پیرامیٹرز کہا جاتا ہے۔

مثال: sum = add (n1, n2);

اس مثال میں ویری ایبلز n1 اور n2 آرگومنٹس ہیں جو فنکشن add() کو پاس کیے ہوئے ہیں۔ جبکہ فنکشن add() کے اندر ویری ایبلز x اور y فنکشن کے پیرامیٹرز ہیں۔

س: کیا فنکشن ڈیفینیشن اور فنکشن کال میں ایک جیسی ڈیٹا ٹائپ استعمال کرنا ضروری ہے؟ اپنے جواب کو ایک مثال کے ساتھ بیان کریں۔

یہ ضروری نہیں ہے کہ ویری ایبلز کو انہیں ناموں کے ساتھ فنکشن میں کال کیا جائے جیسا کہ پیرامیٹرز کے نام ہیں۔ تاہم ہم ایک جیسے نام بھی استعمال کرسکتے ہیں۔ یہاں ایک اور اہم نکتہ یہ بھی ہے کہ اگر ہم ایک جیسے نام استعمال کرتے ہیں تب بھی فنکشن میں استعمال ہونے والے ویری ایبلز اصل ویری ایبلز کی ایک کاپی ہوں گے۔ یہ درج ذیل مثال سے واضح کیا گیا ہے۔

```
 #include<stdio.h>
#include<conio.h>
void fun (int x, int y)
{
        x = 20;
        y = 10;
        printf("Values of x and y in fun(): %d%d", x, y);
}
void main()
{
        int x = 10, y = 20;
        fun ( x , y);
        printf("Values of x and y in main(): %d%d", x , y);
        getch();
}
```

Output

```
Values of x and y in fun(): 20 10
Values of x and y in main ():10 20
```

س: پروگرام میں فنکشنز کی ترتیب کے لیے کس نکتہ کو ذہن میں رکھنا چاہیے؟

پروگرام میں فنکشنز کی ترتیب کے لیے درج ذیل نکات کو ذہن میں رکھنا چاہیے۔

1. اگر کال کیے گئے فنکشن کی ڈیفینیشن کال کرنے والے فنکشن کی ڈیفینیشن سے پہلے ظاہر ہوتی ہے تو فنکشن کے سگنیچر کی ضرورت نہیں ہے۔
2. اگر کال کیے گئے فنکشن کی ڈیفینیشن کال کرنے والے فنکشن کی ڈیفینیشن کے بعد ظاہر ہوتی ہے تو کال کیے گئے فنکشن کا سگنیچر اس کال کرنے والے فنکشن سے پہلے لکھنا ضروری ہے۔ مندرجہ ذیل دونوں کوڈ سٹرکچرز درست ہیں۔

a)

```
int add(int, int);
void main()
{
  printf(" %d ",add(4, 5));
}
int add(int a, int b)
{
  return a + b;
}
```

b)

```
int add(int a, int b)
{
      return a + b;
}
void main()
{
printf(" %d ",add(4, 5));
}
```

س: ایک فنکشن prime() لکھیں جو ایک نمبر کو ان پٹ کے طور پر لیتا ہے اور اگر ان پٹ نمبر پرائم ہے تو 1 ریٹرن کرتا ہے ورنہ 0 ریٹرن کرے۔ اس فنکشن کو main() میں استعمال کریں۔

Program:

```
#include <stdio.h>
#include <conio.h>
int prime(int n)
{
        for (int i = 2 ; i < n ; i ++)
        if(n % i == 0)
        return0;
        return1;
}
void main ()
{
        clrscr();
        int x;
        printf (“Please enter a number: ”);
        scanf (“%d”, &x);
        if(prime(x))
                printf(“%d is a Prime Number :” ,x);
        else
                printf(“%d is  not  a  Prime Number ”,x);
        getch();
}
```

س: ایک ایسا فنکشن لکھیں جو مثبت نمبر کو ان پٹ کے طور پر لے اور 0 سے لے کر اس نمبر تک نمبروں کو جمع کرے اور حاصل جمع ریٹرن کرے۔

Program:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int digitsSum(int n)
{
          int sum = 0;
          for(int i = 0; i <= n; i ++)
          {
          sum = sum + i;
          }
          return sum;
}
void main()
{
          clrscr();
          int number;
          printf(“Please enter a positive number: ");
          scanf(“%d", &number);
          if(number >= 0)
          {
                    int sum = digitsSum (number);
                    printf (“The sum of numbers upto given number is %d", sum);
          }
          else
                    printf(“You entered a negative number: ");
          getch();
}
```