

Refresher Module CS231

Introduction to C



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



Learning Objectives:

- Basic C syntax
 - Control Flow
 - Structures
- Pointers
 - Memory structure
 - Pointer arithmetic
- Arrays
 - n-dimensional arrays
 - Arrays vs pointers

Declaring vs Defining variables

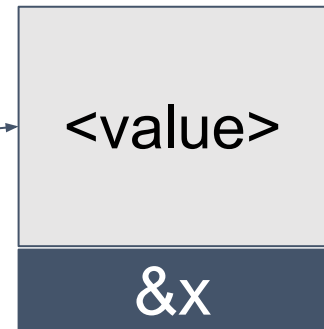
`<return_type> x;`

reserves some memory location
proportional to return type



`x = <value>;`

value is saved at the memory location &x



Flow of control

```
expr1;  
while (expr2){  
    expr3;  
}
```

Flow of control

```
expr1;  
while (expr2){  
    expr3;  
}
```

```
expr1;  
do{  
    expr2;  
}while (expr3);
```

Flow of control

```
expr1;  
while (expr2){  
    expr3;  
}
```

```
expr1;  
do{  
    expr2;  
}while (expr3);
```

```
expr1;  
for(expr2; expr3; expr5;){  
    expr4;  
}
```

Initially

Flow of control

```
expr1;  
while (expr2){  
    expr3;  
}
```

```
expr1;  
do{  
    expr2;  
}while (expr3);
```

```
expr1;  
for(expr2; expr3; expr5){  
    expr4;  
}
```

Initially

```
expr;  
for(expr; expr1; expr3){  
    expr2;  
}
```

Intermediate

Flow of control

```
expr1;  
while (expr2){  
    expr3;  
}
```

```
expr1;  
do{  
    expr2;  
}while (expr3);
```

Pair of braces can be replaced by semicolon



```
expr1;  
for(expr2; expr3; expr5){  
    expr4;  
}
```

Initially

```
expr;  
for(expr; expr1; expr3){  
    expr2;  
}
```

Intermediate

Flow of control

```
expr1;  
while (expr2){  
    expr3;  
}
```

The expressions need not be conditional statements.

Every expression in C has an implicit truth value of 1, other than any expression which evaluates to 0.

```
expr1;  
do{  
    expr2;  
}while (expr3);
```

Pair of braces can be replaced by semicolon

```
expr1;  
for(expr2; expr3; expr5){  
    expr4;  
}
```


Initially

```
expr;  
for(expr; expr1; expr3){  
    expr2;  
}
```

Intermediate

Structures - Basic building blocks

```
typedef struct Node {  
    int data;  
    struct Node *nextptr;  
} Node;
```



Question: What color does the black Node correspond to - green or purple?

Structures - Basic building blocks

Typedef is used to create an *alias*.

```
struct Node {  
    int data;  
    struct Node *nextptr;  
};
```

```
typedef struct Node Node;  
struct Node {  
    int data;  
    Node *nextptr;  
};
```

```
typedef int length_metre;  
typedef int age;
```

```
length_metre x = 5300;  
age y = 82;
```

These two quantities are conceptually different even if they belong to the same datatype.

```
int x = 10;  
int *ip = &x;
```

What **unique** pieces of information can you extract from **ip**?

```
int x = 10;  
int *ip = &x;
```



What **unique** pieces of information can you extract from **ip**?

```
int x = 10;  
int *ip = &x;
```



What **unique** pieces of information can you extract from **ip**?

- We can obtain the **value** of **ip**.
- We can obtain the **address** of **ip**.
- We can **dereference ip** using the * operator to obtain the value at the address of **x**.

Demo of the previous code snippet

```
> cat pointerex1.c
#include <stdio.h>
int main()
{
    address format specifier ip c
    int x=10;
    int* ip=&x;
    printf("Value of x is %d, and value of ip is %p\n",x,ip);
    printf("Address of x is %p, and address of ip is %p\n",&x,&ip);
    printf("Dereferencing ip we get the quantity %d\n",*ip);
    return 0;
}
The commonly used format specifiers in printf() function are:
> ./a.out
Value of x is 10, and value of ip is 0x7ffff0292e5c
Address of x is 0x7ffff0292e5c, and address of ip is 0x7ffff0292e60
Dereferencing ip we get the quantity 10
```

It is used to print the address in a hexadecimal

```
int x = 10;  
int *ip = &x;
```



What **unique** pieces of information can you extract from **ip**?

Operation	Pointer	Normal variable
Obtain value	Yes	Yes
Obtain address using &	Yes	Yes
Dereference using *	Yes	No

Pointer arithmetic

datatype *p = &a;

What is p+n?

Pointer arithmetic

`datatype *p = &a;`

`p+n = &a + n*sizeof(datatype)`

Pointer arithmetic

`datatype *p = &a;`

`p+n = &a + n*sizeof(datatype)`

We are traversing in **blocks of memory** whenever address arithmetic is used.

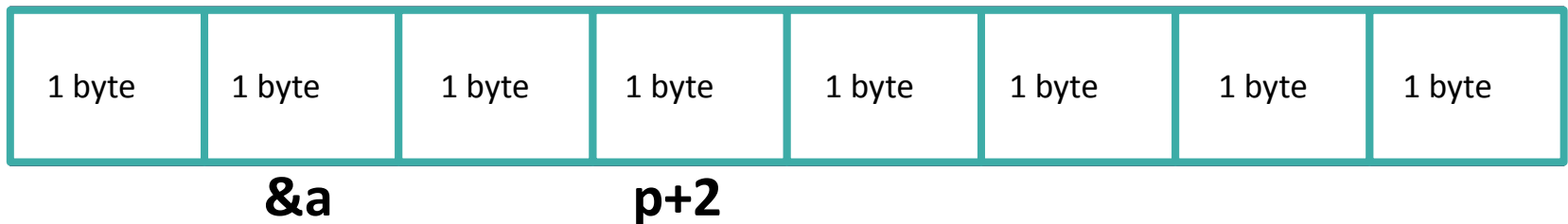
Pointer arithmetic (concrete example)

`char` *p = &a;

$p + 2 = \&a + 2 * \text{sizeof}(\text{char})$

$= \&a + 2 * 1$

$= \&a + 2$



Pointer arithmetic

`datatype *p = &a;`

`p+n = &a + n*sizeof(datatype)`

Pointer arithmetic

`datatype *p = &a;`

`p+n = &a + n*sizeof(datatype)`

Application: Arrays!

What is the output?

```
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;

    printf("%d\n",c1);

    printf("%d\n",c2);
    return 0;
}
```

What is the output?

```
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;

    printf("%d\n",c1);

    printf("%d\n",c2);
    return 0;
}
```

2nd memory location

$$(3251)_{10} = (\text{0000 1100 1011 0011})_2$$

1st memory location

What is the output?

```
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;

    printf("%d\n",c1);

    printf("%d\n",c2);
    return 0;
}
```

Memory representation concepts:
Endianness - Big Endian, Little Endian

Do your own research!

Reference:

<https://www.geeksforgeeks.org/little-and-big-endian-mystery/>

2nd memory location

$$(3251)_{10} = (\text{0000 1100 1011 0011})_2$$

1st memory location

What is the output?

```
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;
    printf("%d\n",c1);

    printf("%d\n",c2);
    return 0;
}
```

$(3251)_{10} = (\text{0000 1100 1011 0011})_2$

2nd memory location

1st memory location

Questions to ask yourself:

- What is cp storing? An address or a value?
- What is the size of the memory location occupied by cp? (Hint: character datatype has ?)
- What is the size of x?
- What is the size of &x?

What is the output?

```
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;

    printf("%d\n",c1);

    printf("%d\n",c2);
    return 0;
}
```

2nd memory location

$$(3251)_{10} = (0000\ 1100\ 1011\ 0011)_2$$

1st memory location

Questions to ask yourself:

- What is cp storing? An address or a value?
cp stores anything that is assigned to it - address or value. If we store an address then we can dereference it.
- What is the size of the memory location occupied by cp? (Hint: character datatype has ?)
- What is the size of x?
- What is the size of &x?

What is the output?

```
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;

    printf("%d\n",c1);

    printf("%d\n",c2);
    return 0;
}
```

2nd memory location

$$(3251)_{10} = (\text{0000 1100 1011 0011})_2$$

1st memory location

Questions to ask yourself:

- What is cp storing? An address or a value?
cp stores anything that is assigned to it - address or value. If we store an address then we can dereference it.
- What is the size of the memory location occupied by cp? (Hint: character datatype has ?)
1 byte or 8 bits
- What is the size of x?
- What is the size of &x?

What is the output?

```
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;

    printf("%d\n",c1);

    printf("%d\n",c2);
    return 0;
}
```

2nd memory location

$$(3251)_{10} = (0000\ 1100\ 1011\ 0011)_2$$

1st memory location

Questions to ask yourself:

- What is cp storing? An address or a value?
cp stores anything that is assigned to it - address or value. If we store an address then we can dereference it.
- What is the size of the memory location occupied by cp? (Hint: character datatype has ?)
1 byte or 8 bits
- What is the size of x?
4 bytes or 32 bits
- What is the size of &x?

What is the output?

```
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;

    printf("%d\n",c1);

    printf("%d\n",c2);
    return 0;
}
```

2nd memory location

$$(3251)_{10} = (0000\ 1100\ 1011\ 0011)_2$$

1st memory location

Questions to ask yourself:

- What is cp storing? An address or a value?
cp stores anything that is assigned to it - address or value. If we store an address then we can dereference it.
- What is the size of the memory location occupied by cp? (Hint: character datatype has ?)
1 byte or 8 bits
- What is the size of x?
4 bytes or 32 bits
- What is the size of &x?
Depends on your OS.

What is the output?

```
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;

    printf("%d\n",c1);

    printf("%d\n",c2);
    return 0;
}
```

Try to reason with the help of the following visual aid what is the value of c1:

$$(3251)_{10} = (\overset{\text{2nd memory location}}{\text{0000 1100}} \overset{\text{1st memory location}}{\text{1011 0011}})_2$$

$(12)_{10}$ $(179)_{10}$

What is the output?

```
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;

    printf("%d\n",c1);

    printf("%d\n",c2);
    return 0;
}
```

Try to reason with the help of the following visual aid what is the value of c1: **179**

$$(3251)_{10} = (\overset{\text{2nd memory location}}{\text{0000 1100}} \overset{\text{1st memory location}}{\text{1011 0011}})_2$$
$$\qquad\qquad (12)_{10} \qquad\qquad (179)_{10}$$

What is the output?

```
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;

    printf("%d\n",c1);

    printf("%d\n",c2);
    return 0;
}
```

What does *cp++ do?

$$(3251)_{10} = (\text{0000 1100 1011 0011})_2$$

2nd memory location 1st memory location

(12)₁₀ (179)₁₀

What is the output?

```
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;

    printf("%d\n",c1);

    printf("%d\n",c2);
    return 0;
}
```

What does *cp++ do?

It is parsed as (*cp) then cp++.

Right now cp points to the 1st memory location

$$(3251)_{10} = (\text{0000 } \text{1100 } \text{1011 } \text{0011})_2$$

(12)₁₀ 1st memory location

(179)₁₀

What is the output?

```
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;

    printf("%d\n",c1);
    printf("%d\n",c2);
    return 0;
}
```

What does *cp++ do?

It is parsed as (*cp) then cp++.

Right now cp points to the 1st memory location

c1 holds the dereferenced value at the 1st memory location.
After cp++ we are pointing to the 2nd memory location.
c2 holds the dereferenced value at the 2nd memory location.

2nd memory location

$$(3251)_{10} = (\text{0000 } \text{1100 } \text{1011 } \text{0011})_2$$

(12)₁₀

1st memory location

(179)₁₀

Demo of the previous code snippet

```
> cat endianex.c
#include <stdio.h>
int main()
{
    unsigned int x = 3251;
    char *cp = &x;
    unsigned char c1 = *cp++, c2 = *cp;

    printf("%d\n", c1);

    printf("%d\n", c2);
    return 0;
}

> ./a.out
179
12
```