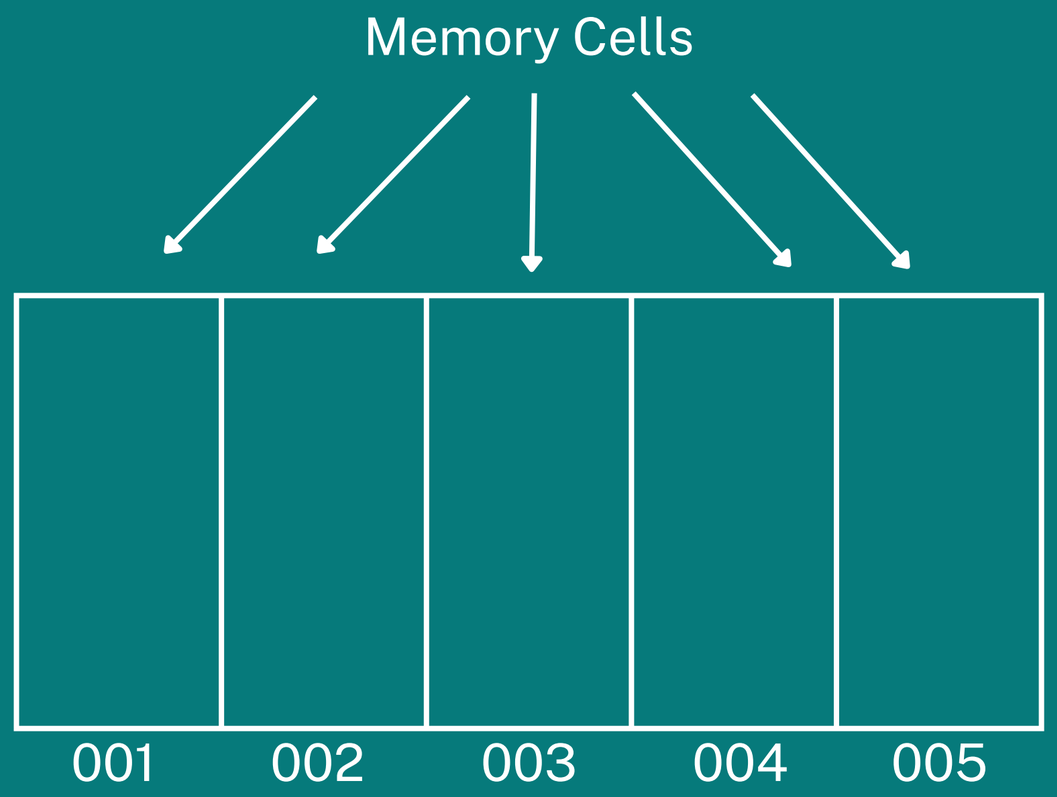
Pointers and Parameter Passing

Memory Addresses



Back to Agenda Page

Pointers

Variable that holds the memory address of other variables

Back to Agenda Page

Syntax to Declare Pointers

<data_type> *<variable_name>;

Example:

- int *ptr;
- char *ptr;
- double *ptr;
- float *ptr;

Back to Agenda Page

2 types of Pointer Operator

Back to Agenda Page

2 types of Pointer Operator

Address Operator

Back to Agenda Page

2 types of Pointer Operator

Address Operator

- Denoted by ‘&’
- Used to obtain the address of a variable.

Back to Agenda Page

2 types of Pointer Operator

Address Operator

- Denoted by ‘&’
- Used to obtain the address of a variable.

Indirection Operator

Back to Agenda Page

2 types of Pointer Operator

Address Operator

- Denoted by ‘&’
- Used to obtain the address of a variable.

Indirection Operator

- Denoted by ‘*’
- Used to access the value of the address held by the pointer.

Back to Agenda Page

Reminders in Using Pointers

A pointer can only point to the data type specified in its declaration

You can give a pointer “no value” by giving it the value NULL.

Without the address operation, the indirect operation y = *p would be semantically incorrect.

Reminders in Using Pointers

A pointer can only point to the data type specified in its declaration

You can give a pointer “no value” by giving it the value NULL.

Without the address operation, the indirect operation y = *p would be semantically incorrect.

```
int num, *intPtr;
float dec, *floatPtr;

intPtr = &num;
floatPtr = &num;
```

Reminders in Using Pointers

A pointer can only point to the data type specified in its declaration

You can give a pointer “no value” by giving it the value NULL.

Without the address operation, the indirect operation y = *p would be semantically incorrect.

```
int num, *intPtr;
float dec, *floatPtr;

intPtr = &num;
floatPtr = &num;
```

Reminders in Using Pointers

A pointer can only point to the data type specified in its declaration

You can give a pointer “no value” by giving it the value NULL.

Without the address operation, the indirect operation y = *p would be semantically incorrect.

```
int num, *intPtr;
float dec, *floatPtr;

intPtr = &num;
floatPtr = &num;
```

Reminders in Using Pointers

A pointer can only point to the data type specified in its declaration

You can give a pointer “no value” by giving it the value NULL.

Without the address operation, the indirect operation y = *p would be semantically incorrect.

```
int num, *intPtr;
float dec, *floatPtr;

intPtr = &num;
floatPtr = &num;
```

```
int *intrPTR = NULL;
```

Reminders in Using Pointers

A pointer can only point to the data type specified in its declaration

```
int num, *intPtr;
float dec, *floatPtr;

intPtr = &num;
floatPtr = &num;
```

You can give a pointer “no value” by giving it the value NULL.

```
int *intrPTR = NULL;
```

Without the address operation, the indirect operation y = *p would be semantically incorrect.

```
int x, *p, **y;

y = *p; //wrong

//correct solution:
p = &x;
y = *p;
```

Pointer-to-Pointer

Start

Used to store the address of another pointer

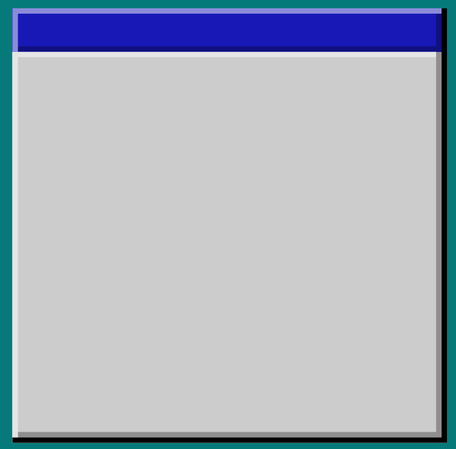
Syntax:

```
<data_type> **<variable_name>;
```

Tips in checking if an operation is valid or not

Back to Agenda Page

Tips in checking if an operation is valid or not



The pointer and the variable have the same base type

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;
x = b;
q = a;
```

The pointer and the variable have the same base type

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;
x = b;
q = a;
```

The pointer and the variable have the same base type

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;
x = b;
q = a;
```

The pointer and the variable have the same base type

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;
x = b;
q = a;
```

The pointer and the variable have the same base type

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;
x = b;
q = a;
```

The pointer and the variable have the same base type

The number of asterisks should be equal on both side

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;
x = b;
q = a;
```

The pointer and the variable have the same base type

```
int p, *x, *y, **z;

x = p;
x = y;
y = z;
```

The number of asterisks should be equal on both side

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;
x = b;
q = a;
```

The pointer and the variable have the same base type

```
int p, *x, *y, **z;

x = p;
x = y;
y = z;
```

The number of asterisks should be equal on both side

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;  ✗
x = b;  ✗
q = a;  ✓
```

The pointer and the variable have the same base type

```
int p, *x, *y, **z;

x = p;  ✗
x = y;  ✓
y = z;
```

The number of asterisks should be equal on both side

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;  ✗
x = b;  ✗
q = a;  ✓
```

The pointer and the variable have the same base type

```
int p, *x, *y, **z;

x = p;  ✗
x = y;  ✓
y = z;  ✗
```

The number of asterisks should be equal on both side

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;  ✗
x = b;  ✗
q = a;  ✓
```

The pointer and the variable have the same base type

```
int p, *x, *y, **z;

x = p;  ✗
x = y;  ✓
y = z;  ✗
```

The number of asterisks should be equal on both side

Address Operator (&)

- adds asterisks

Indirection Operator (*)

- reduces number of astrerisks

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;  ✗
x = b;  ✗
q = a;  ✓
```

The pointer and the variable have the same base type

```
int p, *x, *y, **z;

x = p;  ✗
x = y;  ✓
y = z;  ✗
```

The number of asterisks should be equal on both side

```
int p, *x, *y, **z;

x = &p;
x = y;
y = *z;
```

Address Operator (&)

- adds asterisks

Indirection Operator (*)

- reduces number of astrerisks

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;  ✗
x = b;  ✗
q = a;  ✓
```

The pointer and the variable have the same base type

```
int p, *x, *y, **z;

x = p;  ✗
x = y;  ✓
y = z;  ✗
```

The number of asterisks should be equal on both side

```
int p, *x, *y, **z;

x = &p;  ✓
x = y;
y = *z;
```

Address Operator (&)

- adds asterisks

Indirection Operator (*)

- reduces number of astrerisks

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;  ✗
x = b;  ✗
q = a;  ✓
```

The pointer and the variable have the same base type

```
int p, *x, *y, **z;

x = p;  ✗
x = y;  ✓
y = z;  ✗
```

The number of asterisks should be equal on both side

```
int p, *x, *y, **z;

x = &p;  ✓
x = y;   ✓
y = *z;
```

Address Operator (&)

- adds asterisks

Indirection Operator (*)

- reduces number of astrerisks

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;
float q, a,*b;

p = q;  ✗
x = b;  ✗
q = a;  ✓
```

The pointer and the variable have the same base type

```
int p, *x, *y, **z;

x = p;  ✗
x = y;  ✓
y = z;  ✗
```

The number of asterisks should be equal on both side

```
int p, *x, *y, **z;

x = &p;  ✓
x = y;   ✓
y = *z;  ✓
```

Address Operator (&)

- adds asterisks

Indirection Operator (*)

- reduces number of astrerisks

Back to Agenda Page

Tips in checking if an operation is valid or not

The maximum number of indirection operators on a pointer is the same as its declaration

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;

p = **x;
p = *x;
```

The maximum number of indirection operators on a pointer is the same as its declaration

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;

p = **x;
p = *x;
```

The maximum number of indirection operators on a pointer is the same as its declaration

Back to Agenda Page

Tips in checking if an operation is valid or not

```
int p, *x, *y, **z;

p = **x;
p = *x;
```

The maximum number of indirection operators on a pointer is the same as its declaration

Back to Agenda Page

Practice Questions

11:11PM

Valid or Invalid?

```
int x = 3, y = 2, *t, **v;
char a = ‘a’, b = ‘f’, c = ‘M’, d = ‘D’, *h, *i, **j, **k
```

1. h = &a;
2. t = &b;
3. a = k;
4. v = &t;
5. a = *h;
6. *k = ‘C’
7. k = &i;
8. &h = k;
9. b = **j;
10. **k = ‘h’;

Valid or Invalid?

```
int x = 3, y = 2, *t, **v;
char a = 'a', b = 'f', c = 'M', d = 'D', *h, *i, **j, **k
```

1. h = &a;
2. t = &b;
3. a = k;
4. v = &t;
5. a = *h;
6. *k = 'C'
7. k = &i;
8. &h = k;
9. b = **j;
10. **k = 'h';

Valid or Invalid?

```
int x = 3, y = 2, *t, **v;
char a = 'a', b = 'f', c = 'M', d = 'D', *h, *i, **j, **k
```

✅ 1. h = &a;
❌ 2. t = &b;
3. a = k;
4. v = &t;
5. a = *h;
6. *k = 'C'
7. k = &i;
8. &h = k;
9. b = **j;
10. **k = 'h';

Valid or Invalid?

```
int x = 3, y = 2, *t, **v;
char a = 'a', b = 'f', c = 'M', d = 'D', *h, *i, **j, **k
```

✅ 1. h = &a;
❌ 2. t = &b;
❌ 3. a = k;
4. v = &t;
5. a = *h;
6. *k = 'C'
7. k = &i;
8. &h = k;
9. b = **j;
10. **k = 'h';

Valid or Invalid?

```
int x = 3, y = 2, *t, **v;
char a = 'a', b = 'f', c = 'M', d = 'D', *h, *i, **j, **k
```

✅ 1. h = &a;
❌ 2. t = &b;
❌ 3. a = k;
✅ 4. v = &t;
5. a = *h;
6. *k = 'C'
7. k = &i;
8. &h = k;
9. b = **j;
10. **k = 'h';

Valid or Invalid?

```
int x = 3, y = 2, *t, **v;
char a = 'a', b = 'f', c = 'M', d = 'D', *h, *i, **j, **k
```

✅ 1. h = &a;
❌ 2. t = &b;
❌ 3. a = k;
✅ 4. v = &t;
✅ 5. a = *h;
6. *k = 'C'
7. k = &i;
8. &h = k;
9. b = **j;
10. **k = 'h';

Valid or Invalid?

```
int x = 3, y = 2, *t, **v;
char a = 'a', b = 'f', c = 'M', d = 'D', *h, *i, **j, **k
```

✅ 1. h = &a;
❌ 2. t = &b;
❌ 3. a = k;
✅ 4. v = &t;
✅ 5. a = *h;
❌ 6. *k = 'C'
7. k = &i;
8. &h = k;
9. b = **j;
10. **k = 'h';

Valid or Invalid?

```
int x = 3, y = 2, *t, **v;
char a = 'a', b = 'f', c = 'M', d = 'D', *h, *i, **j, **k
```

✅ 1. h = &a;
❌ 2. t = &b;
❌ 3. a = k;
✅ 4. v = &t;
✅ 5. a = *h;
❌ 6. *k = 'C'
✅ 7. k = &i;
8. &h = k;
9. b = **j;
10. **k = 'h';

Valid or Invalid?

```
int x = 3, y = 2, *t, **v;
char a = 'a', b = 'f', c = 'M', d = 'D', *h, *i, **j, **k
```

✅ 1. h = &a;
❌ 2. t = &b;
❌ 3. a = k;
✅ 4. v = &t;
✅ 5. a = *h;
❌ 6. *k = 'C'
✅ 7. k = &i;
❌ 8. &h = k;
9. b = **j;
10. **k = 'h';

Valid or Invalid?

```
int x = 3, y = 2, *t, **v;
char a = 'a', b = 'f', c = 'M', d = 'D', *h, *i, **j, **k
```

✅ 1. h = &a;
❌ 2. t = &b;
❌ 3. a = k;
✅ 4. v = &t;
✅ 5. a = *h;
❌ 6. *k = 'C'
✅ 7. k = &i;
❌ 8. &h = k;
✅ 9. b = **j;
10. **k = 'h';

Valid or Invalid?

```
int x = 3, y = 2, *t, **v;
char a = 'a', b = 'f', c = 'M', d = 'D', *h, *i, **j, **k
```

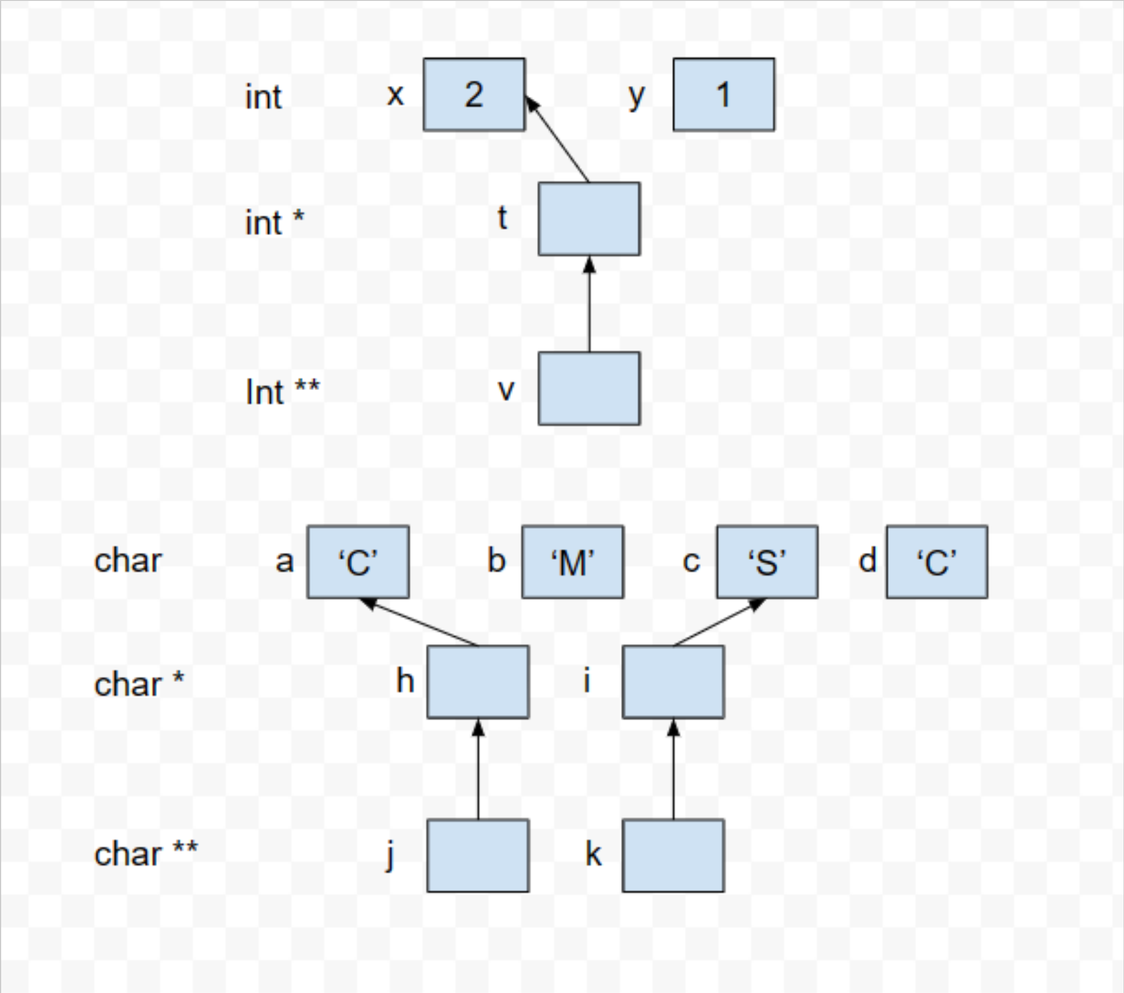
- ✅ 1. h = &a;
- ❌ 2. t = &b;
- ❌ 3. a = k;
- ✅ 4. v = &t;
- ✅ 5. a = *h;
- ❌ 6. *k = 'C'
- ✅ 7. k = &i;
- ❌ 8. &h = k;
- ✅ 9. b = **j;
- ✅ 10. **k = 'h';

Draw the resulting Diagram

```
int x = 3, y = 2, *t, **v;
char a = ‘S’, b = ‘C’, c = ‘M’, d = ‘D’, *h, *i, **j, **k;

t = &x;
a = ‘C’;
h = &c;
i = &b;
k = &h;
j = k;
*i = **j;
k = &i;
*k = h;
*i = ‘S’;
h = &a;
d = **j;
y = *t-y;
v = &t;
**v = y + 1;
```

Draw the resulting Diagram



Parameter Passing

2 types of Parameters

Actual Parameter

- Values passed from the calling function to the function to be executed.

Formal Parameter

- Receivers of the values passed from the function call.

Back to Agenda Page

2 types of Parameters

```
int sample(int a, int b){
    return a;
}

void main(){
    int c = 7;
    sample(c, 8);
}
```

- a and b are formal paramaters.
- The value of c and 8 are the actual parameters.

Back to Agenda Page

2 types of Parameters Passing

Pass by Value

- Passing the actual value to the function

Pass by Reference

- passing the address of the variable or the value of a pointer to a function

Back to Agenda Page

Pass by Value

```c
#include <stdio.h>

int sum(int a, int b){
    return a+b;
}

int main(){

    int num1 = 5;
    int num2 = 9;
    int total = sum (num1,num2);

    return 0;
}

```

Pass by Reference

```c
#include <stdio.h>

int addFive(int *a, int *b){
    *a = *a +5;
    *b = *b + 5;
}

int main(){
    int num1 = 5;
    int num2 = 9;
    addFive(&num1,&num2);

    return 0;
}

```

2 types of Parameters Passing

Pass by Value

- Passing the actual value to the function
- Creates a copy of the original value in the function
- No effect on the actual parameter

Pass by Reference

- passing the address of the variable or the value of a pointer to a function
- A reference (Address) to the variable is passed to the function.
- The original variable is affected by modifications done in the function.

Back to Agenda Page


Thank you!