Pointer Practice Handout

Exploring int, char, struct pointers, lvalues, rvalues, and function passing

Program 1: Basic int pointer

```
#include <stdio.h>
int main() {
    int a = 10;
    int *p = &a;
    printf("a = %d\n", a);
    printf("*p = %d\n", *p);
    printf("Address of a = %p\n", (void*)&a);
    printf("Value of p = %p\n", (void*)p);
}
```

Concept: Concept: p stores the address of a. *p fetches the value. a is an lvalue, 10 is an rvalue.

■ Try it yourself: Predict the output before running!

Program 2: Pointer reassignment and dereferencing

```
#include <stdio.h>
int main() {
    int x = 5, y = 20;
    int *p = &x;
    printf("*p = %d\n", *p);
    p = &y;
    printf("*p = %d\n", *p);
}
```

Concept: Concept: Pointer can be reassigned to another variable. It just stores addresses.

■ Try it yourself: Predict the output before running!

Program 3: char pointer and dereferencing

```
#include <stdio.h>
int main() {
    char c = 'A';
    char *pc = &c;
    printf("c = %c, *pc = %c\n", c, *pc);
    *pc = 'Z';
    printf("After *pc='Z', c = %c\n", c);
}
```

Concept: Concept: Changing through a pointer changes the original variable.

■ Try it yourself: Predict the output before running!

Program 4: Pointer to struct

```
#include <stdio.h>
struct Point {
    int x, y;
};
int main() {
    struct Point p1 = {10, 20};
    struct Point *ptr = &p1;
    printf("x=%d, y=%d\n", ptr->x, ptr->y);
    ptr->x = 50;
    printf("After change: x=%d, y=%d\n", p1.x, p1.y);
}
```

Concept: Concept: Use -> to access members through a pointer.

■ Try it yourself: Predict the output before running!

Program 5: Passing pointer to function

```
#include <stdio.h>
void increment(int *p) {
    (*p)++;
}
int main() {
    int n = 10;
    increment(&n);
    printf("n = %d\n", n);
}
```

Concept: Concept: &n; passes the address, so actual value changes.

■ Try it yourself: Predict the output before running!

Program 6: Passing array (decay into pointer)

```
#include <stdio.h>
void printArray(int arr[], int size) {
    for(int i=0; i<size; i++)
        printf("%d ", arr[i]);
}
int main() {
    int a[] = {1, 2, 3};
    printArray(a, 3);
}
```

Concept: Concept: Array name 'a' decays into pointer to its first element (int*). arr inside function is just a pointer.

■ Try it yourself: Predict the output before running!

Program 7: Pointer arithmetic

```
#include <stdio.h>
int main() {
    int a[] = {10, 20, 30};
    int *p = a;
    printf("*p = %d\n", *p);
    p++;
    printf("*p = %d\n", *p);
}
```

Concept: Concept: Incrementing a pointer moves to the next element.

■ Try it yourself: Predict the output before running!

Program 8: const pointers and pointer to const

```
#include <stdio.h>
int main() {
    int x = 5, y = 6;
    const int *p = &x;  // pointer to const int
    int *const q = &x;  // const pointer to int
    // *p = 10; // Error
    q[0] = 9; // OK
    printf("x = %d\n", x);
}
```

Concept: Concept: const int *p → can’t change value through p. int *const q → can’t change pointer’s target.

■ Try it yourself: Predict the output before running!

Program 9: Double pointers

```
#include <stdio.h>
int main() {
    int a = 100;
    int *p = &a;
    int **pp = &p;
    printf("**pp = %d\n", **pp);
    **pp = 200;
    printf("a = %d\n", a);
}
```

Concept: Concept: **pp accesses the value indirectly through a pointer to pointer.

■ Try it yourself: Predict the output before running!

Program 10: lvalues and rvalues in pointer assignment

```
#include <stdio.h>
int main() {
    int a = 5;
    int *p;
    p = &a;   // OK: &a is rvalue (address), p is lvalue
    *p = 9;   // OK: *p is lvalue (refers to a)
    // &a = p; // ERROR: &a is not assignable
    printf("a = %d\n", a);
}
```

Concept: Concept: lvalue has location, rvalue is temporary. Pointer dereferencing yields an lvalue.

■ Try it yourself: Predict the output before running!