A1

A2

// original  
// does not check if out of bounds  
// always returns 0  
int setValueAtOrg(double \*x, int i, double value){  
 x[i] = value;  
 return 0;  
}  
  
// improved  
// no return type  
int setValueAt(double \*x, int i, double value){  
 \*x = \*x + i;  
 \*x = value;  
}

A3

printf("Question 3\n");  
long a = 1234567890; /\* Hex: 499602d2 \*/  
long b = 1000000000; /\* Hex: 3b9aca00 \*/  
printf("a %x\n",a);  
printf("b %x\n",b);  
// void pointer to b  
void\*p=&b;  
  
// print address of b since p points to this address  
printf("%x\n", p);  
//printf("Address of b: %p\n", &b);  
  
// void pointer p is incremented and cast to a pointer of type long  
// prints b  
printf("%x\n",\*(long\*)p++);  
  
// void pointer p is incremented and cast to a pointer of type char  
// prints  
printf("%x\n",\*(char\*)p++);  
  
//  
printf("%x\n",\*(unsigned char\*)p++);  
  
//  
printf("%x\n", p);

A4

int increment(int \*x){  
 // increment recieves pointer to i  
 // the value of i is incremented by 1 to 5 + 1 = 6  
 // j will be 6  
 // i will be 6  
 return ++(\*x);  
}  
  
int incrementt(int \*x){  
 // incrementt recieves pointer to i  
 // (\*x)++ will be treated as \*(x++)  
 // this means j will be 5 and i will be 6  
 return (\*x)++;  
}

A5

//prints 1 1  
printf("%i %i\n", \*x, \*(short \*)px);  
px++;  
//prints 1 3  
printf("%i %i\n", \*x, \*(short \*)px);

short x = 3;  
short \*px = &x;  
\*(px--) = 20;  
\*px = 21;  
// prints 20 21  
printf("%i %i\n", x, \*px);

A6

struct {  
 char a[10];  
 char b;  
 char c;  
 short int d;  
}myStruct;  
  
union {  
 char a[12];  
 int b;  
 short int d[4];  
} myUnion;  
  
//struct: each element gets its own disk space -> 14  
// char a[10]: 10 bytes + char b 1 byte + char c 1 byte + short int 2 bytes  
//union: disk space is defined by biggest element -> 12  
// char a[12]: 12 bytes  
printf("%i\n", sizeof (myStruct));  
printf("%i\n", sizeof (myUnion));

A7

#define callA callB(5)  
/\*The #define creates a macro,  
 \* which is the association of an identifier or parameterized identifier  
 \* with a token string. After the macro is defined, the compiler can  
 \* substitute the token string for each occurrence of the identifier  
 \* in the source file.  
 \*/  
  
void callB(int a){  
 // prints 10  
 printf("%i\n", a\*2);  
}  
  
  
int main(int argc, const char \* argv[]) {  
 // this calls is substituted for callB(5)  
 callA;  
 return 0;  
}